Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Υλοποίηση Εικονικού Μεταγωγέα για Εφαρμογές του Ίντερνετ του Μέλλοντος

Διπλωματική Εργασία

του

Εμμανουήλ Δημογεροντάκη

Επιβλέπων: Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εργαστήριο Διαχείρισης & Βέλτιστου Σχεδιασμού Δικτύων

Αθήνα, Ιούλιος 2011

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Διαχείρισης & Βέλτιστου Σχεδιασμού Δικτύων

Υλοποίηση Εικονικού Μεταγωγέα για Εφαρμογές του Ίντερνετ του Μέλλοντος

Διπλωματική Εργασία

του

Εμμανουήλ Δημογεροντάκη

Επιβλέπων: Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου, 2011.

.....	.....	.....
Βασίλης Μάγκλαρης	Συμεών Παπαβασιλείου	Δημήτρης Καλογεράς
Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Ερευνητής ΕΠΙΣΕΥ

Αθήνα, Ιούλιος 2011

.....

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

Περίληψη

Τα τελευταία χρόνια, όλο και περισσότερο παρουσιάζεται η ανάγκη για υποδομές ελέγχου και αξιολόγησης νέων πρωτοκόλλων και υπηρεσιών. Σε συνδυασμό με την εκρηκτική ανάπτυξη των τεχνικών και των εφαρμογών της εικονικοποίησης έχει προκύψει ένα πλήθος από εικονικές δικτυακές υποδομές, εγκατεστημένες πάνω στο Ίντερνετ. Οι εικονικές δικτυακές υποδομές έχουν σκοπό να αποτελέσουν ένα ρεαλιστικό περιβάλλον πειραμάτων για ερευνητές. Ωστόσο, μέχρι πρόσφατα, αυτές οι υποδομές περιορίζονταν στην παροχή δυνατοτήτων για πειράματα εντοπισμένα από το επίπεδο Δικτύου της δικτυακής στοίβας και πάνω. Μια αρκετά νέα εικονική δικτυακή υποδομή για διενέργεια πειραμάτων, το VINI Trellis, στοχεύει στην προσφορά επιπλέον δυνατότητα για πειράματα που επιτρέπουν παραμετροποιήσιμες τοπολογίες επιπέδου Ζεύξης. Σε ταυτόχρονη ανάπτυξη βρίσκονται και οι εικονικές δικτυακές συσκευές. Το πιο χαρακτηριστικό παράδειγμα ανοιχτού λογισμικού που υλοποιεί λειτουργίες μεταγωγής είναι το Open vSwitch. Σκοπός αυτής της διπλωματικής είναι η ανάπτυξη λογισμικού για την εγκαθίδρυση και διαχείριση τοπολογιών επιπέδου ζεύξης στο περιβάλλον VINI Trellis με χρήση Open vSwitch. Επιπλέον, στόχος είναι η εγκαθίδρυση τοπολογιών επιπέδου Ζεύξης μεταξύ εικονικών πόρων διαφορετικών πειραματικών υποδομών (VINI Trellis - Federica) που κάνουν χρήση διαφορετικών τεχνολογιών επιπέδου Ζεύξης.

Λέξεις Κλειδιά

εικονικοποίηση, επίπεδο Ζεύξης, εικονικές πλατφόρμες δικτυακών πειραμάτων, εικονικά δίκτυα, εικονικός μεταγωγέας, Ίντερνετ του μέλλοντος

Abstract

In recent years,there is an increasing need for infrastructures that enable control and evaluation of new protocols and services. Combined with the explosive development of techniques and applications of virtualization led to an emerge of a variety of virtual network infrastructures located on top of the Internet. These virtual network infrastructures are designed to provide realistic environment for network experiments from researchers. However, until recently, these facilities were limited to providing opportunities for experiments detected by the Network layer of te network stack and up. A fairly new virtual network infrastructure for conducting experiments , the VINI Trellis, aims to provide additional capability for experiments over configurable link-layer topologies. There have also great advances in the field of virtual network devices. The most striking example of open source software that implements switching functions is Open vSwitch. This diploma thesis aims to develop software for the establishment and management of link-layer topologies within the VINI Trellis environment using Open vSwitch. Furthermore, we hope to establish link-layer topologies between virtual resources from separated experimental infrastructures (VINI Trellis - Federica) that deploy different link-layer technologies.

Keywords

virtualization, link layer, virtual network testbeds, virtual networks, virtual switch, Future Internet

Ευχαριστίες

Θα ήθελα κατ' αρχάς να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Βασίλη Μάγκλαρη και τον κ. Λεωνίδα Λυμπερόπουλο για την καθοδήγησή τους κατά την εκπόνηση της παρούσας εργασίας. Επίσης θα ήθελα να ευχαριστήσω τους Δημήτρη Καλογερά, Γιώργο Ανδρουλιδάκη και Αλέξανδρο Σιούγγαρη, που με τις γνώσεις και τις ιδέες τους βοήθησαν στην επιτυχή ολοκλήρωση της εργασίας. Θα ήθελα να δώσω ιδιαίτερες ευχαριστίες στον Χρήστο Αργυρόπουλο, του οποίου η εργατικότητα και ο ενθουσιασμός του αποτέλεσαν και αποτελούν έμπνευση για μένα.

Ευχαριστώ όλα τα παιδιά του Εργαστηρίου Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων, που με βοήθησαν, ο καθένας με τον τρόπο του, σε κάθε βήμα της διαδικασίας.

Τέλος, ευχαριστώ την οικογένειά μου και τους φίλους μου για την κατανόηση και την υποστήριξή τους όλα αυτά τα χρόνια.

Δημογεροντάκης Εμμανουήλ

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Πρόλογος

Η παρούσα αρχιτεκτονική του Ίντερνετ βρίσκεται υπό αμφισβήτηση. Πολλοί ερευνητές του διαδικτύου έχουν αναγνωρίσει αναδυόμενους φραγμούς της δεδομένης αρχιτεκτονικής του Ίντερνετ. Έχει, λοιπόν, φτάσει η στιγμή η διαδικτυακή έρευνα να αποκτήσει μία πιο ευρεία προοπτική ώστε αν και όπου είναι δυνατό να αναθεωρηθούν οι αρχιτεκτονικές αρχές του Ίντερνετ. Οι μελέτες πρέπει να γίνουν σε περιβάλλοντα ευρείας κλίμακας που θα παράγουν ρεαλιστικά και εφαρμόσιμα αποτελέσματα ώστε να αξιολογηθεί η δυνατότητα υλοποίησης των νέων μεθόδων, να επικυρωθούν οι συνολικές συνέπειες και να προκύψουν περαιτέρω απαιτήσεις, προσανατολισμοί και τις δεδομένα για την έρευνα. Με βάσει τα παραπάνω, οι ευρείας κλίμακας ερευνητικές δικτυακές υποδομές για πειράματα αντιπροσωπεύουν το Ίντερνετ του μέλλοντος.

Τα τελευταία χρόνια έχουν διεξαχθεί σημαντικές έρευνες στον τομέα τον ερευνητικών δικτυακών υποδομών. Πιο συγκεκριμένα, οι εικονικές δικτυακές υποδομές έχουν γνωρίσει μεγάλη ανάπτυξη για αρκετούς λόγους. Ο κυρίαρχος λόγος ανάπτυξης τους είναι οι δυνατότητα που παρέχουν οι τεχνικές εικονικοποίησης να μοιράζουν τους φυσικούς πόρους αποτελεί σημαντικό πλεονέκτημα στην δημιουργία μεγάλων υποδομών που μπορούν να χρησιμοποιηθούν ταυτόχρονα από πολλούς χρήστες. Εξίσου σημαντική έρευνα, έχει διεξαχθεί και στον τομέα των ομοσπονδιακών ερευνητικών υποδομών, που επιτρέπουν την χρήση πόρων από διαφορετικές πειραματικές υποδομές. Οι ομοσπονδιακές υποδομές θεωρούνται οι σημαντικότεροι αντιπρόσωποι του Ίντερνετ του μέλλοντος λόγω της υπερ-ευρείας υποδομής τους και της αυτοματοποιημένης δυνατότητας που παρέχουν για επικοινωνία των υποδομών που την συνιστούν.

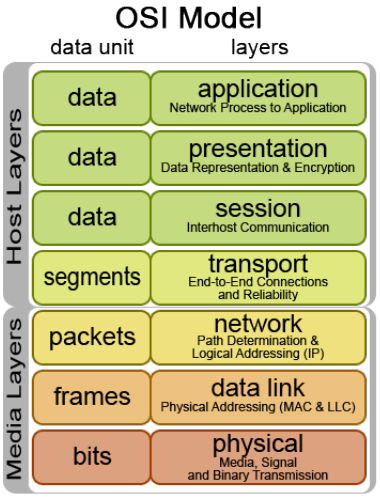
Παρ'όλα αυτά, μέχρι πρόσφατα, οι υποδομές για πειράματα παρείχαν δυνατότητες κυρίως για πειραματισμό με εφαρμογές και πρωτόκολλα του επιπέδου Δικτύου και άνω. Η πλατφόρμα VINI Trellis δημιουργήθηκε με σκοπό να ξεπεράσει αυτό το εμπόδιο και να παρέχει στους ερευνητές δυνατότητα ελέγχου των τοπολογιών επιπέδου Ζεύξης. Η προσέγγιση του Trellis, όμως, στηρίζεται στην έννοια των εικονικών ζεύξεων σημείου προς σημείο. Βασικός στόχος αυτής της διπλωματικής είναι να αλλάξουμε την σημείου προς σημείο προσέγγιση του Trellis προσφέροντας την δυνατότητα δημιουργίας τοπικών δικτύων επιπέδου Ζεύξης, με την βοήθεια του εικονικού μεταγωγέα Open vSwitch. Επιπλεόν, αναγνωρίζοντας την σημασία των ομοσπονδιακών υποδομών παρουσιάζουμε έναν τρόπο επικοινωνίας σε επίπεδο Ζεύξης διαφορετικών υποδομών με σκοπό τον διαμοιρασμό πόρων ανάμεσα σε πειραματικές υποδομές.

Κεφάλαιο 2

Θεωρητική Εισαγωγή

2.1 L2 Δικτύωση

Στο επίπεδο 2 (Layer 2, L2) του δικτυακού προτύπου OSI (σχήμα 2.1) βρίσκεται το επίπεδο Ζεύξης δεδομένων (Data Link Layer). Το L2 παρέχει τα λειτουργικά και διαδικαστικά μέσα για την μεταφορά δεδομένων μεταξύ δικτυακών οντοτήτων , καθώς και την δυνατότητα αναγνώρισης και πιθανώς διόρθωσης σφαλμάτων του Φυσικού επιπέδου (Physical Layer, L1). Παραδείγματα L2 πρωτοκόλλων είναι το Ethernet[1] για τοπικά δίκτυα, το πρωτόκολλο Point-to-Point (PPP, RFC 1661[2]) κ.τ.λ. Το Data Link Layer ασχολείται με την τοπική παράδοση των πλαισίων (Data Link frames) μεταξύ συσκευών που ανήκουν στο ίδιο τοπικό δίκτυο. Τα πλαίσια δεν εξέρχονται από τα όρια του τοπικού δικτύου. Η διαδικτυακή δρομολόγηση και η καθολική διευθυνσιοδότηση είναι λειτουργίες υψηλότερου επιπέδου. Το L2 πρωτόκολλα ασχολούνται με την τοπική παράδοση, την διευθυνσιοδότηση και την "διαιτησία" των συσκευών που θέλουν να αποκτήσουν πρόσβαση στο μέσο μεταφοράς.



Σχήμα 2.1: Το πρότυπο OSI

[1] http://www.networksorcery.com/enp/protocol/ethernet.htm

[2] http://www.ietf.org/rfc/rfc1661.txt

2.1.1 Γέφυρες και Μεταγωγείς

Τα βασικά συστατικά στοιχεία ενός L2 τοπικού δικτύου είναι, πλέον, οι γέφυρες (**bridges**) και οι μεταγωγείς (**switches**). Η γεφύρωση (bridging) αποτελεί μια τεχνική προώθησης πακέτων σε δίκτυα υπολογιστών μεταγωγής πακέτων (packet switched). Αντίθετα με την δρομολόγηση (routing), το bridging δεν κάνει υποθέσεις για το μέρος του δικτύου στο οποίο μπορεί να βρίσκεται μία συγκεκριμένη διεύθυνση. Αντί αυτού, στηρίζεται στην τεχνική της πλημμύρας (flooding) και στον έλεγχο της διεύθυνσης προέλευσης στην επικεφαλίδα των ληφθέντων πλαισίων για τον εντοπισμό άγνωστων συσκευών. Το bridging, λόγω της εξάρτησης του από το flooding, χρησιμοποιείται κυρίως στα δίκτυα τοπικής περιοχής (Local Area Networks, LANs).

Τα **bridges** είναι συσκευές πιο πολύπλοκες από τα hubs και τους repeaters. Μπορούν να αναλύσουν τα εισερχόμενα πλαίσια για να αποφασίσουν αν μπορούν να τα στείλουν σε κάποιο άλλο τμήμα του δικτύου (network segment). Ένα bridge χρησιμοποιεί μία βάση δεδομένων προώθησης (forwarding database) για να στέλνει πλαίσια στα διάφορα network segments. Η forwarding database είναι αρχικά άδεια και γεμίζει με εγγραφές όσο το bridge δέχεται πλαίσια. Προκειμένου να συνδέσει δύο network segments, το bridge διαβάζει την Media Acces Control (MAC) διεύθυνση προορισμού του πλαισίου και αποφασίζει είτε να το προωθήσει είτε να το απορρίψει. Αν το bridge αποφασίσει ότι ο κόμβος προορισμού βρίσκεται σε άλλο segment του δικτύου προωθεί το πακέτο σε αυτό το segment. Αν η διεύθυνση προορισμού ανήκει στο ίδιο segment με την διεύθυνση προέλευσης, τότε το πλαίσιο απορρίπτεται.

Ως L2 **switches**, σε ένα τοπικό δίκτυο, εννοούμε συνήθως ένα bridge με πολλαπλές θύρες (multiport). Δηλαδή μια φυσική συσκευή υλικού (hardware) που χρησιμοποιεί την MAC διεύθυνση της κάρτας δικτύου (Network Interface Card, NIC) ενός υπολογιστικού συστήματος για να αποφασίσει που θα προωθήσει τα πλαίσια. Τα L2 switches διατηρούν πίνακες φιλτραρίσματος (γνωστοί και ως MAC address tables) για την λειτουργία τους. Το L2 switching είναι πολύ αποδοτικό γιατί δεν γίνονται τροποποιήσεις στα πακέτα δεδομένων, αλλά μόνο στο πλαίσιο που τα ενθυλακώνει και μόνο στην περίπτωση διάδοσης σε ανόμοια μέσα (π.χ. από Ethernet σε FDDI[3]). Χρησιμεύει, επίσης στην δημιουργία ομάδων εργασίας (workgroups) και στην μείωση του μεγέθους των τομέων σύγκρουσης (collision domains). Όπως αναφέρθηκε τα απλά L2 switches παρέχουν τις ίδιες λειτουργίες με τα bridges. Το επιτυγχάνουν όμως πιο αποδοτικά αποφεύγοντας περιττούς ελέγχους των πλαισίων και παρουσιάζοντας μια πιο δυναμική συμπεριφορά.

2.1.2 L2 Virtual Private Networks

Ένα εικονικό ιδιωτικό δίκτυο (virtual private network, VPN[15]) είναι ένας ασφαλής τρόπος σύνδεσης σε ένα ιδιωτικό LAN που βρίσκεται σε απομακρυσμένη τοποθεσία, χρησιμοποιώντας το Ίντερνετ ή οποιοδήποτε μη ασφαλές δημόσιο δίκτυο, με σκοπό την μεταφορά δεδομένων ασφαλώς, με χρήση κρυπτογραφίας. Το VPN χρησιμοποιεί επικύρωση (authentication) για να απορρίψει μη εξουσιοδοτημένους χρήστες και κρυπτογράφηση για να εμποδίσει την ανάγνωση πακέτων από μη εξουσιοδοτημένους χρήστες. Πιό συγκεκριμένα, το πρωτόκολλο VPN ενθυλακώνει πακέτα χρησιμοποιώντας μία ασφαλή κρυπτογραφική μέθοδο μεταξύ δύο ή περισσότερων δικτυακών συσκευών που δεν είναι μέλη του ίδιου ιδιωτικού δικτύου. Η κρυπτογράφηση

[3] `http://www.networksorcery.com/enp/protocol/fddi.htm`

εφαρμόζεται ώστε να διατηρηθούν ασφαλή τα δεδομένα όπως διέρχονται από τους συνδεδεμένους κόμβους ενός τοπικού δικτύου ή δικτύων ευρείας περιοχής.

Εμείς θα εστιάσουμε σε υπηρεσίες VPN που λειτουργούν στο L2[**?**].

2.1.2.1 L2 VPN Services

Τα βασικά L2 VPN services είναι[4]:

- **Virtual LAN**: Τα Virtual LAN είναι μία L2 τεχνική που επιτρέπει την συνύπαρξη πολλαπλών LAN broadcast domains, διασυνδεδεμένα με ένα σύνολο από multipoint links (trunks), κάνοντας χρήση του IEEE 802.1Q trunking πρωτοκόλλου.
- **Virtual private LAN service (VPLS)**: Το VPLS είναι ένα L2 Point-to-Point Virtual Private Network (PPVPN) προσομοιώνοντας όλες τις λειτουργίες ενός LAN. Από την προοπτική του χρήστη, ένα VPLS παρέχει την δυνατότητα της διασύνδεσης κάποιων τμημάτων LAN (LAN segments) πάνω από μία υποδομή δικτύου μεταγωγής πακέτων. Η υποδομή αυτή είναι αόρατη στον χρήστη με αποτέλεσμα τα απομακρυσμένα LAN segments να συμπεριφέρονται σαν ένα LAN. Στο VPLS, ο παροχέας της υποδομής προσομοιώνει επιπλέον και ένα bridge που υποστηρίζει learning και, προαιρετικά, VLAN.
- **Pseudo wire (PW)**: Το PW είναι παρόμοια υπηρεσία με το VPWSπαροχή point-to-point προσομοίωση L1 κυκλωμάτων χωρίς data link δομή, αλλά μπορεί επιπλέον να παρέχει διαφορετικά L2 πρωτόκολλα στα δύο άκρα. Συνήθως, το interface του είναι ένα πρωτόκολλο WAN, όπως τα Asynchronous Transfer Mode (ATM) και Frame Relay. Όταν έχουμε σαν σκοπό την παρουσίαση ενός συνεχούς (contiguous) LAN μεταξύ δύο ή περισσότερων τοποθεσιών είναι πιο κατάλληλα τα VPLS και IPLS.
- **IP-only LAN-like service (IPLS)**: Το IPLS είναι ένα υποσύνολο του VPLS με δυνατότητες L3. Χρησιμοποιεί πακέτα και όχι πλαίσια.

2.1.2.2 Tunneling with L2 payload

Η υλοποίηση των VPN services βασίζονται σε υποδομές που υλοποιούνται από tunneling πρωτόκολλα. Στα δίκτυα υπολογιστών ένα πρωτόκολλο tunneling είναι ένα δικτυακό πρωτόκολλο (το delivery protocol) μέσα στο οποίο ενθυλακώνεται ένα άλλο πρωτόκολλο (το payload protocol). Με την χρήση του tunneling μπορούμε για παράδειγμα να μεταφέρουμε ένα πακέτο πρωτοκόλλου μη συμβατού με το δίκτυο παράδοσης, είτε να δημιουργήσουμε ένα ασφαλές κανάλι μέσα σε ένα μη αξιόπιστο δίκτυο.

Ιδιαίτερο ενδιαφέρον, παρουσιάζουν οι τρόποι δημιουργίας VPLS που παρέχει υπηρεσία Ethernet, λόγω της ευρείας χρήσης του Ethernet στα τοπικά δίκτα. Το VPLS μπορεί να προκύψει με την χρήση tunneling πρωτοκόλλων που δέχονται για payload protocol το Ethernet. Ακολουθούν οι βασικότερες τεχνολογίες ενθυλάκωσης πλαισίων Ethernet.

Layer 2 Tunneling Protocol (L2TP): Το L2TP είναι ένα tunneling πρωτόκολλο, αναπτυγμένο από την Cisco Systems[4], που χρησιμοποιείται για την δημιουργία VPNs. Δεν παρέχει κρυπτογράφηση ή εμπιστευτικότητα από μόνο του. Βασίζεται σε κάποιο πρωτόκολλο

[4] `http://www.cisco.com/`

κρυπτογράφησης που μεταφέρεται μέσα στο tunnel για να παρέχει ασφάλεια. Το L2TP ενεργεί σαν L2 πρωτόκολλο αλλά τυπικά ανήκει στο Επίπεδο Συνόδου (Session Layer, L5). Βασίζεται στα παλαιότερα πρωτόκολλα Layer 2 Forwading Protocol (L2F) και Point-to-Point Tunneling Protocol(PPTP). Η τελευταία έκδοση L2TPv3 παρέχει επιπλέον επιλογές ασφαλείας, βελτιωμένη ενθυλάκωση και δυνατότητα να μεταφέρει πλαίσια διαφορετικά του PPP πάνω από ένα διαδίκτυο IP, όπως Ethernet, Frame Relay, ATM κ.τ.λ. Το L2TPv3 μπορεί, λοιπόν, να χρησιμοποιηθεί ως υποδομή ενός VPLS.

Generic Routing Encapsulation (GRE): Το GRE[5] είναι ένα tunneling πρωτόκολλο αναπτηγμένο από την Cisco Systems που μπορεί να δεχτεί σαν payload μία μεγάλη ποικιλία από πρωτόκολλα, μεταφέροντας τα μέσα από εικονικά point-to-point links πάνω από ένα διαδίκτυο πρωτοκόλλου IP. Όπως και το L2TP, το GRE δεν δημιουργεί ασφαλή tunnels αλλά επιτρέπει την ενθυλάκωση πρωτοκόλλων που παρέχουν ασφάλεια, όπως το IPsec. Το GRE, όπως λέει και το όνομά του, έχει ως σκοπό να αποτελέσει ένα γενικό πρωτόκολλο, ικανό να ενθυλακώσει μία μεγάλη ποικιλία από πρωτόκολλα. Ένα από τα πρωτόκολλα που μπορούν να αποτελέσουν το payload protocol είναι και το Ethernet. Η τεχνολογία αυτή ονομάζεται EoGRE και μπορεί να χρησιμοποιηθεί για να στηρίξει ένα VPLS.

Multiprotocol Label Switching (MPLS) Το MPLS είναι ένας υψηλά επεκτάσιμος, ανεξάρτητος από κάθε τηλεπικοινωνιακό πρωτόκολλο μηχανισμός μεταφοράς πακέτων. Σε ένα δίκτυο MPLS ανατίθενται ετικέτες στα πακέτα δεδομένων. Οι αποφάσεις προώθησης των πακέτων γίνονται αποκλειστικά με βάση το περιεχόμενο των ετικετών. Έτσι δημιουργούνται "κυκλώματα" από άκρη σε άκρη (end-to-end) ανεξαρτήτως του τύπου του μέσου μεταφοράς και του πρωτοκόλλου. Το πλεονέκτημα του MPLS είναι ότι εξαλείφει κάθε εξάρτηση σε συγκεκριμένη τεχνολογία Data Link Layer, καθώς και την ανάγκη πολλαπλών δικτύων L2 για την ικανοποίηση διαφορετικών τύπων κίνησης. Το MPLS αποτελεί, ουσιαστικά, έναν τρόπο δημιουργίας εικονικών ζεύξεων (virtual links) μεταξύ απομακρυσμένων κόμβων και μπορεί να ενθυλακώσει πολλά διαφορετικά είδη πρωτοκόλλων.

2.2 Εικονικοποίηση(virtualization)

Ως virtualization μπορούμε να ορίσουμε κάθε τεχνολογία που συνδυάζει η μοιράζει υπολογιστικούς πόρους ώστε να τους αναθέσει σε ένα η περισσότερα λειτουργικά περιβάλλοντα χρησιμοποιώντας μεθοδολογίες όπως: τμηματοποίηση (partition) σε επίπεδο υλικού ή λογισμικού, μερική ή πλήρη προσομοίωση, emulation[6], διαμοιρασμό χρόνου και πολλές άλλες [10]. Μπορούμε λοιπόν μέσω του virtualization να πετύχουμε έναν, ή και περισσότερους, από τους παρακάτω σκοπούς: δημιουργία ενός επιπέδου αφαίρεσης, απόκρυψη χαρακτηριστικών και, τέλος δημιουργία απομονωμένων περιβαλλόντων.

Στην συνέχεια απαριθμούμε κάποια από τα βασικά πλεονεκτήματα του virtualization [10]:

1. **Server Consolidation**: Ενοποίηση του φόρτου εργασίας πολλαπλών υποχρησιμοποιούμενων "υπολογιστών" σε λιγότερους με σκοπό την εξοικονόμηση hardware, λειτουργίες ελέγχου και διαχείρισης των υποδομών.

[5] `http://datatracker.ietf.org/doc/rfc2784/`, `http://datatracker.ietf.org/doc/rfc2890/`

[6] χρήση υλικού(hardware) ή λογισμικού(software) για την αντιγραφή των λειτουργιών ενός συστήματος σε ένα δεύτερο, διαφορετικό σύστημα

2. **Sandboxing**: Οι εικονικές μηχανές(virtual machines, VMs) χρησιμέυουν στην παροχή ασφαλών απομονομένων περιβαλλόντων(sandboxes) για την εκτέλεση ξένων ή λιγότερο αξιόπιστων εφαρμογών.
3. **Multiple Execution Environments**: Δημιουργία πολλαπλών ανεξάρτητων περιβαλλόντων εκτέλεσης και άυξηση του QoS εξασφαλίζοντας συγκεκριμένη ορισμένη "ποσότητα" πόρων.
4. **Virtual hardware**: Παροχή μη διαθέσιμου υλικού (π.χ. εικονικούς προσαρμογείς(virtual adapters) Ethernet, εικονικούς μεταγωγείς(virtual switches) Ethernet κ.τ.λ.).
5. **Software Migration**: Διευκολύνει την μετανάστευση(migration) του λογισμικού, παρέχοντας έτσι ευκινησία(mobility)/
6. **Testing/QA**: Βοηθάει στην παραγωγή αυθαίρετων σεναρίων που είναι δύσκολο να παραχθούν στην πραγματικότητα, διευκολύνοντας έτσι την διαδικασία των δοκιμών.

Το virtualization μπορεί να έχει ως στόχο οποιοδήποτε στοιχείο ενός υπολογιστικού συστήματος, όπως για παράδειγμα το hardware, το software, την μνήμη, είτε άλλα μέσα αποθήκευσης, δεδομένων είτε δικτύων. Απο όλα αυτά εμείς θα παρουσιάσουμε τα server και network virtualization, όπου όπως θα δούμε το πρώτο θα αποτελέσει μέσο για την επίτευξη του δεύτερου.

2.2.1 Εικονικοποίηση Υλικού (Server Virtualization)

Το server virtualization[3] μπορεί να επιτευχθεί με πολλούς τρόπους, καθένας από τους οποίους προσφέρει διαφορετικά πλεονεκτήματα αλλά και μειονεκτήματα. Θα ασχοληθούμε με τεχνικές virtualization σε επίπεδο υπολογιστή, είτε λειτουργικού συστήματος(operating system, OS). Το αποτέλεσμα αυτών των τεχνικών είναι περισσότερο η απόκρυψη των φυσικών χαρακτηριστικών της υποδομής, παρά η δημιουργία μιας αφηρημένης υπολογιστικής πλατφόρμας. Το λογισμικό που ελέγχει το virtualization αποκαλείται συνήθως "υπερπεπόπτης"(hypervisor) ή "ελεγκτής εικονικής μηχανής"(virtual machine monitor) Στην συνέχεια παρουσιάζονται οι βασικότερες τεχνικές αυτού του είδους.

2.2.1.1 Πλήρης Εικονικοποίηση (Full virtualization)

Στην επιστήμη των υπολογιστών, το full virtualization είναι μία μέθοδος virtualization που χρησιμοποιείται ώστε να παρέχει ένα συγκεκριμένο είδος περιβάλλοντος εικονικής μηχανής (virtual machine environment), πιο συγκεκριμένα, αυτό που είναι μία πλήρης εξομοίωση του υποκείμενου hardware. Το full virtualization απαιτεί κάθε ξεχωριστό συστατικό του hardware να απεικονίζεται σε κάποιο από τα διάφορα virtual machines, περιλαμβάνοντας πλήρη: σύνολο εντολών (instruction set), λειτουργίες εισόδου εξόδου (I/O operations), διακοπές (interrupts), πρόσβαση στην μνήμη (memory access) και οποιαδήποτε άλλα στοιχεία χρησιμοποιούνται από το sofware που εκτελείται στο υποκείμενο μηχάνημα και προορίζεται για εκτέλεση στο virtual machine. Σε ένα τέτοιο περιβάλλον κάθε software που είναι ικανό να παράγει εντολές συμβατές με το πραγματικό hardware μπορεί να εκτελεστεί στο virtual machine, ακόμα και οποιοδήποτε λειτουργικό σύστημα (operating system, OS). Αυτή είναι και η βασική διαφοροποίηση με άλλες τεχνικές platform virtualization, δηλαδή ότι δεν χρειάζεται συγκεκριμένο ή τροποποιημένο software να εκτελείται στο virtual machine.

Το full virtualization έχει αποδειχτεί πολύ επιτυχές για:

1. Διαμοιρασμός ενός υπολογιστικού συστήματος μεταξύ πολλαπλών χρηστών.
2. Απομόνωση των χρηστών μεταξύ τους.
3. Emulation καινούργιου hardware για την επίτευξη αυξημένης αξιοπιστίας, ασφάλειας και παραγωγικότητας.

2.2.1.2 Υποβοηθούμενη απο Υλικό εικονικοποίηση (Hardware-assisted virtualization)

Το hardware-assisted virtualization είναι μία προσέγγιση που επιτρέπει αποδοτικό full virtualization εκμεταλλευόμενο ικανότητες του hardware, και κυρίως των επεξεργαστών του οικοδεσπότη (host). Το virtual machine monitor μπορεί, με την βοήθεια του hardware-assisted virtualization, να εικονικοποιήσει αποδοτικά το σύνολο του x86 instruction set, χρησιμοποιώντας για τις "ευαίσθητες" εντολές ένα σύστημα τύπου παγίδευση-και-μίμηση (trap-and-emulate) στο hardware.

2.2.1.3 Μερική εικονικοποίηση (Partial virtualization)

Στο partial virtualization, το virtual machine εξομοιώνει πολλαπλά στιγμιότυπα ενός μεγάλου μέρους του υποκείμενου hardware περιβάλλοντος,και συγκεκριμένα χώρους διευθύνσεων (address spaces). Συνήθως αυτό σημαίνει ότι το virtual machine δεν θα υποστηρίζει την εκτέλεση ενός πλήρους operating system, όπως στο full virtualization, αλλά μπορούν να εκτελεστούν πολλές εφαρμογές. Μία σημαντική μορφή partial virtualization είναι η εικονικοποίηση χώρου διευθύνσεων (address space virtualization), στην οποία το virtual machine αποτελείται από ένα ανεξάρτητο adrress space. Αυτή η δυνατότητα απαιτεί hardware που να υποστηρίζει μετάθεση διευθύνσεων (address relocation).

Το partial virtualization αποτελεί, ιστορικά, ένα πολύ σημαντικό βήμα προς την επίτευξη του full virtualization, και είναι σημαντικά πιο εύκολο να υλοποιηθεί. Παρέχει την δυνατότητα για την δημιουργία αξιόλογων virtual machines, δυνατά να υποστηρίξουν σημαντικές εφαρμογές. Έχει αποδειχτεί ικανό για τον διαμοιρασμό υπολογιστικών πόρων μεταξύ πολλαπλών χρηστών. Ωστόσο, σε σύγκριση με το full virtualization, μειονεκτεί σε καταστάσεις που απαιτούν συμβατότητα προς τα πίσω (backward compatibility) ή φορητότητα (portability). Εφ'όσον υπάρχει δυσκολία στην ακριβή πρόβλεψη των χαρακτηριστικών που χρησιμοποιούνται από μία δεδομένη εφαρμογή, οπότε σε περίπτωση που κάποια χαρακτηριστικά του hardware δεν έχουν εξομοιωθεί οποιοδήποτε software τα χρησιμποιεί θα αποτυγχάνει.

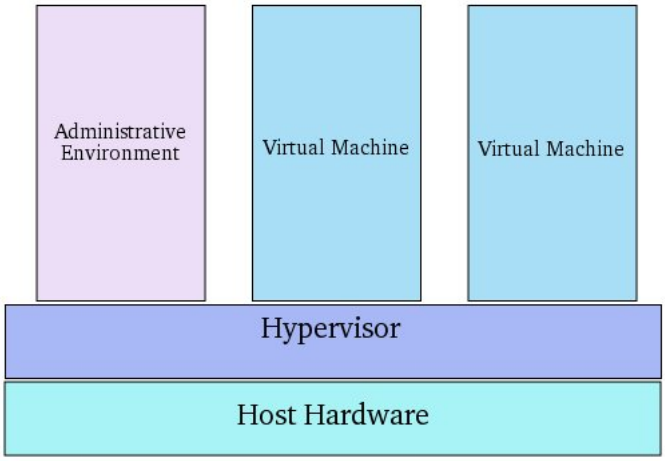
2.2.1.4 Παραεικονικοποίηση (Paravirtualization)

Είναι μία τεχνική virtualization που παρουσιάζει στα virtual machines μια διεπαφή λογισμικού (software interface) που είναι παρόμοια αλλά όχι πανομοιότυπη με αυτή του υποκείμενου hardware.

Ο στόχος της τροποποιημένης διεπαφής (interface) είναι να μειώσει το μερίδιο χρόνου εκτέλεσης του φιλοξενούμενου (guest) που αφορά τις λειτουργίες που είναι ουσιωδώς πιο δύσκολη η εκτέλεση τους σε ένα εικονικό περιβάλλον σε σύγκριση με το μη-εικονικό περιβάλλον. Το paravirtualization παρέχει ειδικά καθορισμένα "άγκιστρα"(hooks) που επιτρέπουν στους guests και στον host να ζητούν και να αναγνωρίζουν αυτές τις διεργασίες, που σε άλλη

περίπτωση θα εκτελούνταν στον εικονικό τομέα (virtual domain) όπου η απόδοση είναι χειρότερη. Μία επιτυχημένη paravirtualized πλατφόρμα μπορεί να απλουστέψει τον διαχειριστή των εικονικών μηχανών (virtual machine manager, VMM) νμεταφέροντας την εκτέλεση των κρίσιμων διεργασιών από το virtual domain στο host domain και μειώνοντας την συνολική υποβάθμιση απόδοσης των εκτελέσεων στον guest.

Το paravirtualization απαιτεί το operating system του guest να είναι ρητά τροποποιημένο ώστε να είναι συμβατό με την διεπαφή προγραμματισμού της εφαρμογής (application programing interface, API) του paravirtualization. Ενά συμβατικό OS που δεν αναγνωρίζει το paravirtualization δεν μπορεί να εκτελεστεί πάνω σε έναν paravirtualizing VMM. Εν τούτοις, ακόμα και σε περιπτώσεις όπου το OS δεν γίνεται να τροποποιηθεί, μπορεί να υπάρχουν διαθέσιμα συστατικά που επιτρέπουν πολλά από τα σημαντικά πλεονεκτήματα επίδοσης του paravirtualization.



Σχήμα 2.2: Hypervisor Virtualization[7]

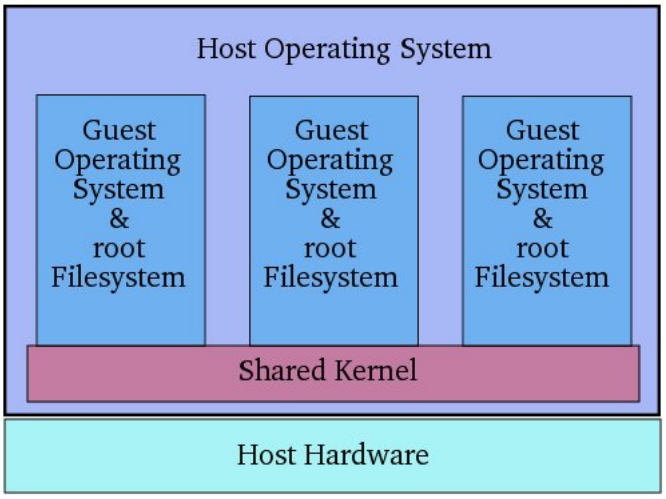
2.2.1.5 Εικονικοποίηση σε επίπεδο λειτουργικού συστήματος (Operating system-level virtualization)

Είναι μία μέθοδος server virtualization όπου ο πυρήνας (kernel) του OS επιτρέπει την δημιουργία πολλαπλών απομονωμένων στιγμιοτύπων του σε επίπεδο χρήστη (user-space). Τέτοια στιγμιότυπα, που αποκαλούνται δοχεία(containers) ή φυλακές (jails), έχουν την ικανότητα να φαίνονται σαν πραγματικοί servers από την πλευρά των χρηστών τους. Σε συστήματα Unix, αυτή η τεχνολογία μπορεί να παρομοιαστεί με τον τυπικό chroot[8] μηχανισμό. Εκτός από τους μηχανισμούς isolation, ο πυρήνας παρέχει επίσης δυνατότητες διαχείρισης πόρων, ώστε να οριοθετήσει την επιρροή των δραστηριοτήτων του ενός container στα υπόλοιπα.

Το OS-level virtualization χρησιμοποιείται συνήθως σε περιβάλλοντα εικονικής φιλοξενίας (virtual hosting), όπου χρησιμεύει στην δέσμευση περιορισμένων πόρων hardware ανάμεσα σε έναν μεγάλο αριθμό αμοιβαία μη εμπιστευόμενων (mutually-distrusting) χρηστών. Χρησιμοποιείται επίσης, σε μικρότερη κλίμακα, για την ενοποίηση του hardware του server μετακινώντας υπηρεσίες (services) απομονωμένων hosts σε έναν server. Άλλα τυπικα σενάρια περιλαμβάνουν

[8] http://en.wikipedia.org/wiki/Chroot

εφαρμογές σε διαχωρισμένα containers για αυξημένη ασφάλεια, ανεξαρτησία από το hardware και τις επιπρόσθετες λειτουργίες διαχείρισης πόρων. Τέλος, υλοποιήσεις OS-level virtualization που παρέχουν ενεργή μετανάστευση (live migration) μπορούν να χρησιμοποιηθούν για δυναμική εξισορρόπηση φόρτου (load balancing) μεταξύ κόμβων σε μία συστοιχία υπολογιστών (cluster).



Σχήμα 2.3: OS-level Virtualization[9]

2.3 Εικονικοποιήση δικτύων (Network Virtualization)

Εικονικοποιήση δικτύων (Network Virtualization) είναι η διαδικασία του συνδυασμού πραγματικών και εικονικών δικτυακών πόρων σε μια διαχειριστική οντότητα λογισμικού. Το τελικό προϊόν είναι το εικονικό δίκτυο(virtual network). Ο σκοπός της εικονικοποίησης δικτύων είναι ο διαμοιρασμός δικτυακών πόρων σε συστήματα και χρήστες με αποδοτικό, ελεγχόμενο και ασφαλή τρόπο. Τα εικονικά δίκτυα χωρίζονται σε δύο μεγάλες κατηγορίες[2]: **εσωτερικά** και **εξωτερικά**.

Τα **εξωτερικά** εικονικά δίκτυα αποτελούνται από κάποια τοπικά δίκτυα που διαχειρίζονται με λογισμικό ως ενιαία οντότητα. Τα συστατικά ενός κλασσικού εξωτερικά εικονικού δικτύου είναι τα switches, σε επίπεδο hardware, και η τεχνολογία VLAN, σε επίπεδο software. Παραδείγματα χρήσης τους περιλαμβάνουν μεγάλα εταιρικά δίκτυα και data centers.

Ένα **εσωτερικό** εικονικό δίκτυο αποτελείται από ένα σύστημα που χρησιμοποιεί virtual machines ή παρεμφερείς τεχνολογίες προσαρμοσμένα πάνω σε ένα τουλάχιστον ψευδο-δικτυακή διεπαφή (pseudo-network interface). Τα virtual machines αυτά, μπορούν να επικοινωνούν μεταξύ τους σαν να συνυπήρχαν στο ίδιο τοπικό δίκτυο, δημιουργώντας έτσι ένα εικονικό δίκτυο. Τα συστατικά στοιχεία του είναι εικονικές δικτυακές διεπαφές (virtual network interface) ή virtual NICs (VNICs[10]) και εικονικοί μεταγωγείς (virtual switches).

Επίσης, υπάρχει η δυνατότητα να συνδυαστούν δικτυακοί πόροι ώστε να συνυπάρξουν εσωτερικά και εξωτερικά εικονικά δίκτυα. Για παράδειγμα, μπορούμε να προσαρμόσουμε προσωπικά

[10]ψευδο-δικτυακό interface που που ρυθμίζεται πάνω σε έναν φυσικό δικτυακό προσαρμογέα του συστήματος (NIC)

συστήματα με εσωτερικά εικονικά δίκτυα σε LANs (Local Area Networks) που ανήκουν σε ένα ευρύτερο, εξωτερικό εικονικό δίκτυο. Στην συνέχεια, κάθε αναφορά στον όρο virtual network θα αφορά τον συνδυασμό εσωτερικών και εξωτερικών εικονικών δικτύων.

2.3.1 Συστατικά Εικονικών Δικτύων

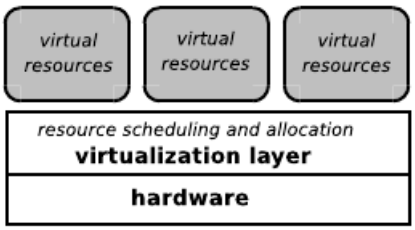
Αντίθετα με πολλές προτάσεις (π.χ.[8]) πιστεύουμε ότι ένα virtual network στηρίζεται πάνω σε 4 βασικά στοιχεία[7]:

- **virtual hosts**, που εκτελούν λογισμικό και προωθούν πακέτα,
- **virtual links**, που μεταφέρουν τα πακέτα μεταξύ των virtual hosts,
- **virtual switches**, που εκτελούν λειτουργίες για την δημιουργία εικονικών τοπικών δικτύων και
- **logical routers**, που εκτελούν τις υπηρεσίες δρομολόγησης σε εικονικό διαδικτυακό επίπεδο.

Ένας **virtual host** παρέχει την ψευδαίσθηση ενός αφιερωμένου φυσικού οικοδεσπότη (physical host), ενώ στην πραγματικότητα μπορεί να υπάρχουν ταυτόχρονα και άλλοι virtual hosts στο ίδιο φυσικό υλικό (physical hardware). Αν δούμε αφαιρετικά έναν virtual host μπορούμε να τον απεικονίσουμε σαν ένα "κουτί" που περιέχει πόρους, όπως στο σχήμα 2.4. Ένας virtual host φαίνεται να έχει αφιερωμένους φυσικούς πόρους (pysical resources) ή λογικούς πόρους (logical resources). Παραδείγματα των physical resources είναι η ΚΜΕ (CPU), η μνήμη και το εύρος ζώνης (bandwidth), ενώ τα logical resources είναι πόροι που υλοποιούνται από το OS, όπως ο πίνακας διεργασιών (process table), ο πίνακας σελίδων (page table), ο πίνακας προώθησης IPv4 (IPv4 forwarding table) και η ενδιάμεση μνήμη (memory buffer). Στην πραγματικότητα, όλοι αυτοί οι πόροι είναι "εικονικοί" από την άποψη ότι διατίθενται από ένα virtualization layer που υλοποιεί την αφαίρεση του virtual host. Το virtualization layer δημιουργεί εικονικούς πόρους από τους φυσικούς χρησιμοποιώντας μηχανισμούς δέσμευσης πόρων και χρονοδρομολόγησης, έτσι ώστε κάθε αφηρημένος virtual host να δέχεται τους αναμενόμενους πόρους που ζητήθηκαν. Ακόμη, η αφαίρεση του virtual host οριοθετεί το έυρος των logical resources που αντιστοιχούν στο "κουτί". Με αυτόν το τρόπο, κάθε virtual host μπορεί αφαλώς να διαχειριστεί τα δικά του logical resources. Το virtualization layer μπορεί να μην εικονικοποιεί όλους τους διαθέσιμους πόρους, με πιθανό αποτέλεσμα την ύπαρξη πόρων στο φυσικό μηχάνημα αλλά εκτός του "κουτιού". Αυτοί οι πόροι είτε θα είναι μή διαθέσιμοι, είτε θα υποστηρίζουν περιορισμένη "αλληλεπίδραση" με το εσωτερικό του virtual host, είτε χρησιμοποιούνται και από άλλες εφαρμογές ταυτόχρονα.

Οι virtual hosts μπορούν να επιτύχουν δύο είδη isolation:

- *Απομόνωση πόρων* (Resource isolation): Εξασφαλίζει ότι κανένας virtual host δεν μπορεί έχει πρόσβαση στους πόρους άλλου virtual host. Οι δεμευτές μνήμης (resource allocators) στο virtualization layer πολυπλέκουν και προγραμματίζουν τους φυσικούς πόρους ώστε να παρέχουν εικονικούς πόρους μέσα σε έναν virtual host.
- *Απομόνωση Χώρου Ονομάτων* (Namespace isolation): Εξασφαλίζει ότι κάθε virtual host μπορεί να ονομάσει και να αναφερθεί σε πόρους (όπως διεργασίες, αρχεία, μνήμη,



Σχήμα 2.4: Host Virtualization

δικτυακά interface, δικτυακές διευθύνσεις, πίνακες προώθησης (forwarding tables)) και δεν δύναται να αναφερθεί σε πόρους άλλων πλαισίων. Για παράδειγμα, μία εφαρμογή σε έναν virtual host δεν μπορεί να προσθέσει διαδρομές στην Βάση Πληροφοριών Προώθησης (Forward Information Base, FIB) ενός άλλου virtual host, και δύο ή περισσότεροι virtual hosts μπορούν να χρησιμποιούν την ίδια διεύθυνση IP για να ονομάσουν διαφορετικά εικονικά δικτυακά interfaces.

Ένα **virtual link** φαίνεται σαν ένα physical link, αλλά πολλά virtual links μπορεί να μοιράζονται ένα physical link. Από την άλλη, ένα virtual link μπορεί να αποτελείται από πολλά hops στο υποκείμενο φυσικό δίκτυο.

Τα **virtual switches** είναι software που επιτρέπει την επικοινωνία μεταξύ πολλών VMs. Όπως και ένα φυσικό Ethernet switch, ένα virtual switch μπορεί να κάνει περισσότερα από την απλή προώθηση πακέτων. Μπορεί να διαχειριστεί έξυπνα την δικτυακή επικοινωνία ελέγχοντας τα πακέτα πριν τα προωθήσει. Κάποιοι πωλητές, ενσωματώνουν virtual switches στο λογισμικό για virtualization που παρέχουν, είτε μπορεί να περιέχονται στο firmware ενός φυσικού server.

Σχετικά πρόσφατα, οι πωλητές δρομολογητών (routers) ξεκίνησαν να υποστηρίζουν virtualization του hardware των δρομολογητών τους. Οι λογικοί δρομολογητές (**logical routers**) χωρίζουν έναν φυσικό router σε πολλαπλούς logical routers που διατηρούν τους δικούς τους πίνακες δρομολόγησης (routing tables), interfaces, πολιτικές (policies), και στιγμιότυπα πρωτοκόλλων δρομολόγησης (routing protocols). Το βασικό ζητούμενο των logical routers είναι η ενοποίηση πολλαπλών δικτυακών στοιχείων σε μία συσκευή hardware, απλοποιώντας έτσι τις ρυθμίσεις σε φυσικό επίπεδο (π.χ. καλώδια και racks) και μειώνοντας τον απαιτούμενο φυσικό χώρο καθώς και τα κόστη. Επιπλέον, η ύπαρξη των logical routers επιτρέπει προσαρμοσμένες ρυθμίσεις δρομολόγησης ή και διαφορετικά πρωτόκολλα δρομολόγησης για συγκεκριμένες εφαρμογές.

2.3.2 Τεχνικές Εικονικοποίησης Δικτύων

Η έννοια των πολλαπλών συνυπαρχόντων λογικών δικτύων είναι αυτή που περιγράφει, ουσιαστικά τον όρο εικονικό δίκτυο. Τα εικονικά δίκτυα μπορούν κατηγοριοποιηθούν σε τέσσερις βασικές κλάσεις ως εξής[9]:

- **Virtual Local Area Networks (VLANs)**: Ένα VLAN είναι ένα σύνολο από λογικά δικτυωμένους hosts με μοναδικό broadcast domain ανεξάρτητα της φυσικής συνδεσιμότητας, σύμφωνα με το πρωτόκολλο 802.1Q.Όλα τα πλαίσια σε ένα VLAN

περιέχουν ένα VLAN ID στην επικεφαλίδα MAC, και οι μεταγωγείς με δυνατότητες VLAN χρησιμοποιούν τόσο την διεύθυνση MAC όσο και το VLAN ID για να προωθήσουν τα πλαίσια. Εφ'όσον τα VLANs βασίζονται σε λογικές αντί για φυσικές ζεύξεις η διαχείριση του δικτύου, ο έλεγχος του και η επαναρύθμιση των VLANs είναι ευκολότερη από αυτές των των αντίστοιχων φυσικών. Επιπρόσθετα, τα VLANs παρέχουν αυξημένα επίπεδα απομόνωσης.

- **Virtual Private Networks(VPNs)**: Ένα VPN, όπως περιγράφηκε στο §2.1.2, είναι ένα δεσμευμένο(dedicated) δίκτυο που συνδέει πολλαπλές τοποθεσίες χρησιμοποιόντας ιδιωτικές και ασφαλείς σήραγγες(tunnels) τοποθετημένες πάνω σε δημόσια δίκτυα επικοινωνίας όπως το Ίντερνετ. Στις περισσότερες περιπτώσεις τα VPNs ενώνουν απομακρυσμένες γεωγραφικά τοποθεσίες μίας εταιρικής επιχείρησης. Κάθε τοποθεσία VPN περιέχει μία ή περισσότερες συσκευές άκρου πελάτη(customer edge) που συνδέονται με έναν ή περισσότερους δρομολογητές άκρου παροχέα(provider edge).

- **Active and Programmable Networks**: Η έρευνα πάνω στα Active and Programmable Networks πυροδοτήθηκε από την ανάγκη για δημιουργία, ανάπτυξη και διαχείριση νέων υπηρεσιών γρήγορα με βάση τις ανάγκες των χρηστών. Εκτός από προγραμματισιμότητα, προσφέρουν έννοιες όπως απομονωμένα περιβάλλοντα που επιτρέπουν πολλαπλούς χρήστες να εκτελούν πιθανώς συγκρουόμενους(conflicting) κώδικες στα ίδια δικτυακά στοιχεία χωρίς να προκαλούν δικτυακές αναστάθειες.

- **Overlay Networks**: Αποτελούν λογικά δίκτυα τοποθετημένα πάνω σε ένα ή περισσότερα υπαρκτά φυσικά δίκτυα. Το ίδιο το Ίντερνετ ξεκίνησε σαν overlay πάνω από το τηλεπικοινωνιακό δίκτυο. Στο σημερινό Ίντερνετ τα overlays δημιουργούνται, κυρίως, στο επίπεδο εφαρμογής. Υπάρχουν, όμως, και υλοποιήσεις σε χαμηλότερα επίπεδα της δικτυακής στοίβας (network stack). Τα overlay δεν χρειάζονται, ούτε προκαλούν, αλλαγές στο (δίκτυο βάση???)underlay δίκτυο. Σαν αποτέλεσμα, έχουν χρησιμοποιηθεί εκτενώς ως σχετικά απλός και ανέξοδος τρόπος για την υλοποίηση νέων χαρακτηριστικών και διόρθωση λαθών στο Ίντερνετ. Ένα πλήθος overlay αρχιτεκτονικών έχουν προταθεί τα τελευταία χρόνια με σκοπό την επίλυση ποικίλων προβλημάτων, περιλαμβανομένων των: δυνατότητα multicasting, παροχή υπηρεσιών QoS, διαθεσιμότητα δρομολόγησης στο Ίντερνετ, προστασία από επιθέσεις DOS(Denial of Service), δίκτυα για Content Distribution[11](CDNs) και διαμοιρασμό αρχείων. Ακόμα μία χρήση τους είναι στην δημιουργία testbed(§2.4) με σκοπό τον σχεδιασμό και την αξιολόγηση νέων αρχιτεκτονικών.

2.4 Πλατφόρμες Δικτυακών Πειραμάτων

Η σημερινή φύση του Διαδικτύου αποτελεί φραγμό για την διενέργεια πειραμάτων, αλλά και την ανάπτυξη και δοκιμή νέων υπηρεσιών και τεχνολογιών. Αυτό το πρόβλημα έρχονται να λύσουν οι πλατφόρμες που υλοποιούν **Network testbeds**.

Ένα testbed είναι μια περιβάλλον για πειραματισμό (test environment). Τα testbeds επιτρέπουν αυστηρό, διαφανή και αναπαράξιμο τρόπο δοκιμής νέων τεχνολογιών και εργαλείων. Έτσι, ένα Network Testbed είναι ένα περιβάλλον για έλεγχο νέων δικτυακών ή διαδικτυακών εφαρμογών και αρχιτεκτονικών (π.χ. πειραματικά πρωτόκολλα). Δηλαδή, δικτυακά περιβάλλοντα

[11]σύστημα από υπολογιστές που περιέχουν αντίγραφα δεδομένων τοποθετημένα σε διάφορους κόμβους του δικτύου

που προσομοιώνουν την λειτουργία πραγματικών δικτύων. Σε αυτά, ο χρήστης έχει την δυνατότητα να πειραματιστεί κάτω από πραγματικές συνθήκες, χωρίς όμως να αλληλεπιδρά, άρα και να επηρεάζει, το πραγματικό Διαδίκτυο[5].

2.4.1 Κατηγορίες Network Testbed

Μπορούμε να ορίσουμε τέσσερα βασικά είδη Network Tesbed[11]:

- *Cluster testbeds*: Βασίζονται στην έννοια του network emulation. Δηλαδή την τεχνική όπου ένα υπαρκτό δίκτυο προσομοιώνεται με σκοπό να αξιολογηθεί η επίδοση, να προβλεφθεί η επίδραση κάποιων αλλαγών ή και για να βελτιστοποιηθούν αποφάσεις αρχιτεκτονικής. Τέτοια testbeds ενώνουν ένα μεγάλο αριθμό δικτυακών συστατικών (π.χ. ζεύξεις,μεταγωγείς, δρομολογητές κ.τ.λ.) σε μία ενιαία εγκατάσταση που μπορεί να προσπελαστεί απομακρυσμένα από χρήστες μέσω ενός web interface.
- *Overlay testbeds* : Βασίζονται στην έννοια του overlay network, που περιγράψαμε παραπάνω. Μπορούμε να φανταστούμε ότι οι κόμβοι του overlay είναι συνδεδεμένοι μεταξύ τους με εικονικές, είτε λογικές ζεύξεις. Κάθε μία από τις αυτές αντιστοιχεί σε ένα μονοπάτι, ενός η περισσότερων, φυσικών ζεύξεων του underlay network.
- *Federated testbeds* : Μπορούν να δημιουργηθούν από την διασύνδεση άλλων ανεξάρτητων τοπικά διαχειριζόμενων testbed.
- *Networking research kits* : Συλλογές συστατικών λογισμικού και υλικού που μπορούν να χρησιμοποιηθούν από ερευνητές για να εγκαταστήσουν τα δικά τους τοπικά δικτυακά εργαστήρια.

2.4.2 Εικονικά Network Testbeds

Αρχικά εξηγούμε τα πλεονεκτήματα των εικονικών Network Testbeds και στην συνέχεια παρουσιάζουμε κάποια χαρακτηριστικά παραδείγματα εικονικών Network Testbeds.

2.4.2.1 Η σημασία της εικονικοποίησης για τα Network Testbeds

Στο πλαίσιο των Network testbeds, η έννοια του virtualization αναφέρεται σε μία ποικιλία από μεθόδους που έχουν αναπτυχθεί για την επίλυση προβλημάτων στην κατασκευή και την χρήση των testbeds. Για παράδειγμα οι εικονικές ζεύξεις των cluster testbeds, η εικονικοποίηση φυσικών πόρων έτσι ώστε να διαμοιράζονται από πολλούς ερευνητές, ακόμα και η προσομοίωση δικτύων. Μπορούμε να εντοπίσουμε τρία βασικά προβλήματα που έρχεται να λύσει το virtualization στα Network Testbeds[5]:

1. Οι ερευνητές πρέπει να μπορούν να πειραματιστούν εύκολα με νέες αρχιτεκτονικές σε live κίνηση.
2. Πρέπει να υπάρχει ένα εύλογος τρόπος να τοποθετούνται επικυρωμένες αρχιτεκτονικές σε εφαρμογή.
3. Πρέπει οι νέες προτεινόμενες αρχιτεκτονικές να μπορούν να απευθυνθούν στην επίλυση όχι μόνο μεμονωμένων προβλημάτων, αλλά στο ευρύ φάσμα προβλημάτων αρχιτεκτονικής που μπορεί να αντιμετωπίζει το Ίντερνετ. the Internet.

2.4.2.2 Planetlab

Το Planetlab[12] είναι ένα ανοιχτό, παγκόσμιο και κατανεμημένο virtual overlay network testbed.Σκοπός του είναι αποτελέσει την υποδομή για μία νέα γενιά εφαρμογών και υπηρεσιών για το Ίντερνετ. Εξυπηρετεί δύο βασικές λειτουργίες:

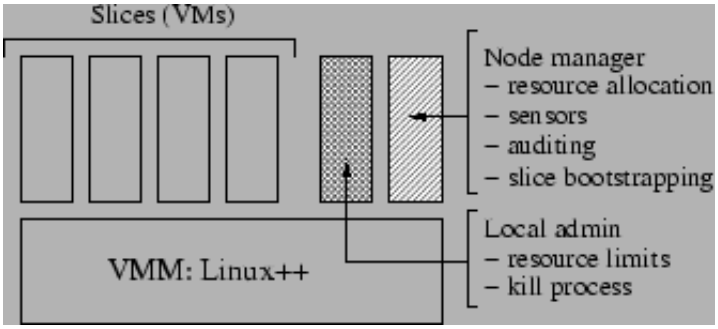
- Πλατφόρμα για μακρόχρονες υπηρεσίες.
- Testbed για δικτυακά και διαδικτυακά πειράματα.

Ουσιαστικά το Planetlab είναι μια συλλογή υπολογιστών κατανεμημένων πάνω στο Ίντερνετ. Ο κάθε ένας από αυτούς είναι ένας κόμβος. Οι κόμβοι του Planetlab έχουν σαν λειτουργικό μία κανονική έκδοση Linux με με κάποιες βελτιώσεις για την υποστήριξη πολλαπλών εφαρμογών και χρηστών.Οι εφαρμογές μοιράζονται έναν κόμβο Planetlab διαμένοντας σε ξεχωριστούς virtual servers. Από την προοπτική της εφαρμογής, στο περιβάλλον εκτέλεσης φαίνεται σαν ένας προσωπικός υπολογιστής Linux. Το Planetlab αποτελεί ιδανική πλατφόρμα για την εκτέλεση κατανεμημένων, αποκεντρωμένων εφαρμογών και υπηρεσιών πάνω από πολλούς κόμβους. Υποστηρίζει τόσο βραχυπρόθεσμες όσο και μακροπρόθεσμες υπηρεσίες ικανές να στηρίξουν μια πελατειακή βάση. Πρόκειται δηλαδή για ένα testbed αλλά και πλατφόρμα εφαρμογής που υποστηρίζει την μετάβαση μίας εφαρμογής από το δοκιμαστικό στάδιο απευθείας σε μία υπηρεσία που εξελίσσεται συνεχώς.

Από τους ερευνητές, που μέσω του Planetlab δοκιμάζουν υπηρεσίες τελικών χρηστών, αναμένεται, επίσης, η ανάπτυξη υπο-υπηρεσιών που με επανάδραση θα χρησιμοποιούνται στο ίδιο το Planetlab για την ανάπτυξη άλλων. Μακροπρόθεσμος στόχος είναι να προσδιοριστούν υπηρεσίες κοινών δομικών στοιχείων βάση των οποίων μπορούν να κατασκευαστούν άλλες υπηρεσίες και εφαρμογές. Η προοπτική αυτή έχει σαν κίνητρο την γενική διερεύνηση σχετικά με την καλύτερη δυνατή επίδραση της δικτυακής ερευνητικής κοινωνίας στο Ίντερνετ. Το Planetlab αποτελεί υλοποίηση του οράματος να εισαχθούν νέες τεχνολογίες στο Ίντερνετ μέσω της ανάπτυξης των δικτύων overlay.

Βασική ορολογία:

- *Site*: Μία φυσική τοποθεσία όπου βρίσκονται nodes του Planetlab.
- *Node*: Ένας ειδικού σκοπού server που παρέχει υπηρεσίες του Planetlab.
- *Slice*: Ένα σύνολο από δεσμευμένους πόρους, που είναι κατανεμημένοι μέσα στο Planetlab. Για τους περισσότερους users, ένα slice σημάινει πρόσβαση σε ένα Unix shell σε έναν αριθμό κόμβων του Planetlab. Τα slices έχουν συγκεκριμένο χρόνο ζωής και πρέπει να ανανεώνονται περιοδικά προκειμένου να παραμένουν έγκυρα.
- *Sliver*: Οι δεσμευμένοι πόροι ενός *node* που ανατίθενται σε έναν *user*. Ένα slice αποτελείται από πολλά slivers.
- *Principal Invsetigator (PI)*: Οι PIs σε κάθε site είναι υπεύθυνοι για την διαχείριση των slices και των users.
- *User*: Οποιοσδήποτε που αναπτύσσει και εκτελεί εφαρμογές στο Planetlab.



Σχήμα 2.5: Planetlab Node Architecture

Αξίζει να σημειωθεί ότι οι όροι Site, Node, Slice και Sliver χρησιμοποιούνται γενικότερα στον χώρο των virtual network testbeds.

[12] [13]

2.4.2.3 VINI

Το VINI[13] είναι μια εικονική δικτυακή υποδομή, η οποία επιτρέπει σε ερευνητές δικτύων να αξιολογήσουν πρωτόκολλα και υπηρεσίες που έχουν αναπτύξει σε μία ευρύτερη δικτυακή περιοχή. Το VINI επιτρέπει στους ερευνητές να αναπτύξουν και να αξιολογήσουν τις ιδέες τους με την χρήση λογισμικού routing, κίνηση πακέτων και δικτυακών γεγονότων. Με σκοπό να παρέχει στους ερευνητές ευελιξία στον σχεδιασμό των πειραμάτων τους, το VINI υποστηρίζει ταυτόχρονα πειράματα με αυθαίρετες δικτυακές τεχνολογίες πάνω από μία διαμοιραζόμενη φυσική υποδομή.

Το βασικό πρόβλημα στο οποίο απευθύνεται το VINI είναι ότι οι ερευνητές στην προσπάθεια αξιολόγησης των προτάσεων τους πρέπει να επιλέξουν ανάμεσα στις προσομοιώσεις, οδηγούμενες είτε από συνθετικά μοντέλα τοπολογίας είτε απο μετρήσεις των υπαρχόντων πρωτοκόλλων, ή σε μικρής κλίμακας testbeds. Ιδανικά, για την αξιολόγηση θα έπρεπε οι ερευνητές να έχουν την δυνατότητα να διεξάγουν πειράματα που είναι ταυτόχρονα και ρεαλιστικά και ελεγχόμενα.

Σύμφωνα με τους δημιουργούς του VINI[6], η ερευνητική κοινότητα χρειάζεται μία υποδομή για την διεξαγωγή πειραμάτων που ικανοποιεί τους τέσσερις παρακάτω στόχους:

- *Εκτέλεση πραγματικού software δρομολόγησης*: Οι ερευνητές πρέπει να έχουν τη δυνατότητα να εκτελούν τυπικό software δρομολόγησης στα πειράματά τους, ώστε να αποτιμούν τις επιδράσεις των επεκτάσεων στα πρωτόκολλα και να αξιολογούν νέες υπηρεσίες σε συμβατικά εμπορικά δικτυακά στοιχεία.

- *Έκθεση σε ρεαλιστικές δικτυακές συνθήκες*: Οι ερευνητές πρέπει να μπορούν να κατασκευάζουν πειράματα σε ρεαλιστικές τοπολογίες και συνθήκες δρομολόγησης. Τα πειράματα θα πρέπει να μπορούν να μελετάνε την συμπεριφορά ενός συστήματος σχετικά με εξωτερικά ερεθίσματα, όπως μηνύματα πρωτοκόλλων δρομολόγησης από το Ίντερνετ.

[12] `http://www.planet-lab.org/`

[13] `http://www.vini-veritas.net`

- *Έλεγχος δικτυακών γεγονότων*: Οι ερευνητές πρέπει να μπορούν να παρεμβάλλουν δικά τους δικτυακά γεγονότα (π.χ. αποτυχίες ζεύξης(link failures) και flash crowds[14]) που δεν εμφανίζονται συχνά στην πράξη, την διεξαγωγή ελεγχόμενων πειραμάτων και ακριβείς μετρήσεις αυτών των γεγονότων.

- *Μεταφορά πραγματικής κίνησης*: Οι ερευνητές πρέπει να έχουν τη δυνατότητα να αξιολογούν τα πρωτόκολλά τους και υπηρεσίες που μεταφέρουν κίνηση εφαρμογών μεταξύ πραγματικών κόμβων, ώστε να επτραπούν μετρήσεις της επίδοσης από άκρο σε άκρο και οι επιπτώσεις στους τερματικούς κόμβους.

Για την ικανοποίηση αυτών των στόχων χρειάζονται εργαλεία για την δημιουργία virtual networks αλλά και της υποδομής πάνω στην οποία θα στηρίζονται. Το Planetlab για παράδειγμα είναι μία υποδομή που υποστηρίζει την εκτέλεση πολλαπλών κατανεμημένων υπηρεσιών σε εκατοντάδες κόμβους στον κόσμο. Ωστόσο, η διενέργεια ελεγχόμενων και ρεαλιστικών πειραμάτων στο Planetlab δεν είναι εύκολη, λόγω των τριών πρώτων λόγων που αναφέρθηκαν. Από την άλλη, συλλογές εργαλείων (toolkits), όπως τα X-Bone[15] και Violin, αυτοματοποιούν την δημιουργία των overlay networks χρησιμοποιώντας tunnels μεταξύ των hosts, ειπτρέποντας σε ερευνητές να αξιολογήσουν νέα πρωτόκολλα και υπηρεσίες. Εν τούτοις, αυτά τα εργαλεία δεν συνδέονται με κάποια μεγάλου εύρους φυσική υποδομή που να αντικατοπτρίζει ένα φυσικό δίκτυο. Ο σκοπός του VINI είναι να συνδυάσει αυτές τις δύο τεχνολογίες.

2.4.2.4 Federica

Η υποδομή FEDERICA είναι αδιάφορη προς τον τύπο των πρωτοκόλλων, των υπηρεσιών και των εφαρμογών που μπορεί να τεθούν υπό δοκιμή, ενώ ταυτόχρονα επιτρέπει την εκτέλεση καινοτομικών πειραμάτων. Η multi-domain, μεγάλης έκτασης υποδομή του FEDERICA παρέχει ένα περιβάλλον πραγματικών συνθηκών για την από άκρο σε άκρο υλοποίηση δικτυακών πειραμάτων.

Τα εικονικά slices της υποδομής FEDERICA ανατίθενται σε ερευνητές ύστερα από αίτηση, για να πραγματοποιήσουν τα πειράματά τους σε ένα ευρύ υπόστρωμα. Ένα εικονικό slice είναι ένας συνδυασμός κυκλωμάτων και εικονικών κόμβων. Μπορεί να περιέχει routed IP circuits, συστήματα και δρομολογητές.

Σε επίπεδο εικονικοποιήσης δικτυακών συσκευών, όλοι οι πόροι που βασίζονται στην διαστρωμάτωση πρωτοκόλλων κατά το μοντέλο OSI μπορούν να εικονικοποιηθούν και να δωθούν στους χρήστες. Στο L2, τα physical devices μπορούν να παραμετροποιηθούν ώς εικονικά Ethernet switches. Όπως και στους λογικούς δρομολογητές, τα εικονικά switches είναι, επίσης, ανεξάρτητα devices, ενώ ο χρήστης δεν αντιλαμβάνεται την διαφορά ανάμεσα σε εικονικό και πραγματικό Ethernet switch.

Για την εφαρμογή εικονικών λειτουργιών του L3 υπάρχουν δύο επιλογές. Σύμφωνα με την πρώτη επιλογή τα physical network devices μπορούν να μεταφερθούν σε παραμετροποιημένους logical/virtual routers και να τοποθετηθούν στο εικονικό περιβάλλον οποιουδήποτε FEDERICA node. Στην δεύτερη επιλογή παραμετροποιούμε τα υπάρχοντα logical/virtual routers και τα πειράμτα εκτελούνται κάτω από διαφορετικές συνθήκες απόδοσης.

[14] μεγάλο κύμα κίνησης προς κάποιον server που προκαλεί δραματική αύξηση στον φόρτο εργασίας, προκαλεί σοβαρές πιέσεις στις ζεύξεις που οδηγούν στον server και οδηγεί σε σημαντική απώλεια πακέτων και συμφόρηση

[15] `http://www.isi.edu/xbone/`

Για την επίτευξη εικονικοποίησης >L3, τα FEDERICA nodes περιλαμβάνουν υπολογιστές με λογισμικό και λειτουργικό σύστημα εικονικοποίησης. Σε ένα ανεξάρτητο στιγμιότυπο ενός VM μπορεί να εφαρμοστεί οποιαδήποτε αναπαράσταση και κρυπτογράφηση δεδομένων (L7 και L6), intra-host επικοινωνία (L5) και QoS από άκρο σε άκρο.

Κεφάλαιο 3

Πλατφόρμες που χρησιμοποιήθηκαν

Για να πετύχουμε στον στόχο μας πρέπει να στηριχτούμε πάνω σε κάποια αποδοτική τεχνολογία για virtual network testbeds, καθώς και σε κάποια τεχνολογία που να υλοποιεί το virtual switching. Στην συνέχεια περιγράφονται οι τεχνολογίες που τελικώς επιλέχτηκαν και αποτέλεσαν την βάση για την υλοποίηση μας.

3.1 Virtual Network Testbed: Vini

Στην συνέχεια παρουσιάζουμε το Vini [1], το Virtual Network testbed που αποτέλεσε την βάση για για το σύστημα που υλοποιήθηκε.

3.1.1 Επισκόπηση του Vini

Το VINI είναι ένα testbed όπως το Planetlab(∮ 2.4.2.2), όπου χρήστες μπορούν να δημιουργήσουν εικονικές τοπολογίες μέσα στα δικά τους slices . Μία εικονική τοπολογία VINI αποτελείται από virtual machines (όπως τα slices του Planetlab σε τεχνικό επίπεδο) συνδεδεμένα από point-to-point(σημείο προς σημείο) virtual links. Οι εφαρμογές που εκτελούνται μέσα σε ένα VINI slice έχουν την δυνατότητα να στείλουν και να λάβουν κίνηση διαμέσου της της εικονικής τοπολογίας, καθώς και να ελέγχουν πως τα πακέτα θα προωθούνται μέσα στην τοπολογία. Με αυτό τον τρόπο, δίνεται η δυνατότητα να εκτελείται ανοιχτό λογισμικό για routing, όπως το Quagga, μέσα στο VINI slice ώστε να δημιουργεί ένα πλήρες routing overlay.

Το VINI συνδέεται άμεσα με το Planetlab. Από την ανάπτυξη του VINI έχουν παραχθεί τρία βασικά επιτεύγματα:

1. Ένα testbed ξεχωριστό από το δημόσιο Planetlab. Το VINI, ουσιαστικά, είναι ένα στιγμιότυπο ενός "private Planetlab" που χρησιμοποιεί το MyPLC (βλέπε ∮3.1.3.1) για την διαχείριση των κόμβων.

[1] `http://www.vini-veritas.net/`

2. Ένα σύνολο από επεκτάσεις στον Planetlab kernel και εργαλείων που ονομάζεται Trellis. Το Trellis συνδέει τις εικονικές τοπολογίες με τα slices. Επί του παρόντος, το Trellis εκτελείται μόνο σε κόμβους που ανήκουν στην υποδομή του VINI, και όχι στην υποδομή του PLanetlab. Υπάρχει, όμως συνεργασία με το Planetlab ώστε στο μέλλον να παρέχει την δυνατότητα για δημιουργία τοπολογιών όπως του VINI.

3. Το αρχικό proof-of-concept toolkit PL-VINI. Το toolkit αυτό χρησιμοποιεί τις τεχνολογίες Click[2], User-Mode Linux[3], XORP[4] και Quagga[5] προκειμένου να δημιουργήσει routing overlays μέσα σε slices που ανήκουν στο public Planetlab. Η ανάπτυξη του PL-VINI έχει σταματήσει.

3.1.2 Trellis

Το Trellis είναι μια πλατφόρμα software για την φιλοξενία πολλαπλών εικονικών δικτύων σε μοιραζόμενο εμπορικό hardware. Το Trellis επιτρέπει σε κάθε εικονικό δίκτυο να ορίσει την δική του τοπολογία, πρωτόκολλα ελέγχου και forwarding tables, διευκολύνοντας την ανάπτυξη προσαρμοσμένων υπηρεσιών σε ένα απομονωμένο, παραμετροποιήσιμο και προγραμματίσιμο δίκτυο και μειώνοντας το κόστος με τον διαμοιρασμό μίας φυσικής υποδομής. Οι στόχοι του Trellis είναι οι εξής:

- *Ταχύτητα*: Ένα εικονικό δίκτυο πρέπει να είναι ικανό να μεταφέρει πακέτα σε ταχύτητες πολλών Gigabits.

- *Isolation*: Προκειμένου να αποτραπούν οι παρεμβάσεις μεταξύ των εικονικών δικτύων, η υποδομή θα πρέπει να παρέχει namespace isolation και isloation πόρων, τόσο του συστήματος (π.χ. PIDs, αρχεία, CPU) όσο και δικτυακών πόρων (π.χ. forwarding tables, link, bandwidth).

- *Ευελιξία*: Μία υπηρεσία που εκτελείται σε ένα εικονικό δίκτυο πρέπει να μπορεί να καθορίσει το δικό της πρωτόκολλο δρομολόγησης. Η πλατφόρμα πρέπει να παρέχει έναν δυνατό και κατανοητό περιβάλλον ανάπτυξης δικτυακών υπηρεσιών.

- *Επεκαταστιμότητα (scalability)*: Η πλατφόρμα πρέπει να μπορεί να υποστηρίξει, ταυτόχρονα, όσα εικονικά δίκτυα όσα κοστίζει η εγκατάσταση και η συντήρησή της.

- *Χαμηλό κόστος*: Το κόστος της φιλοξενίας ενός εικονικού δικτύου πρέπει να είναι πολύ χαμηλό. Το Trellis μπορεί να εκτελείται πάνω σε συμβατικό εμπορικό hardware (π.χ. υπολογιστές προορισμένοι για servers) ώστε να μειωθούν τα κόστη και τα εμπόδια αποδοχής από τους χρήστες. Η χρήση συμβατικού εμπορικού hardware επιτρέπει επίσης στην υποδομή να συμβαδίζει με προόδους στην τεχνολογία (π.χ. πολυπύρηνοι επεξεργαστές).

Με βάση αυτά ζητούμενα προέκυψε η παρακάτω αρχιτεκτονική από τους δημιουργούς του Trellis[7].

[2] `http://read.cs.ucla.edu/click/click`

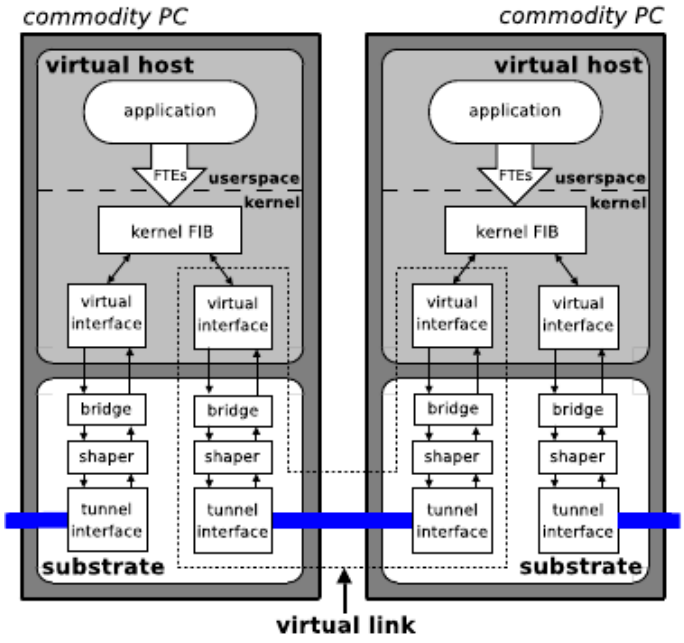
[3] `http://user-mode-linux.sourceforge.net/`

[4] `http://www.xorp.org/`

[5] `http://www.quagga.net/`

3.1.2.1 Trellis Architecture

Το Trellis δεν αποτελεί προσπάθεια ανάπτυξης ενός Virtual Network Testbed από το μηδέν. Είναι ένας συνδυασμός από ένα σύνολο τεχνολογιών αιχμής που θεωρήθηκαν κατάλληλες, με σκοπό να προκύψει ένα νέο, αποδοτικό και ολοκληρωμένο Virtual Network Testbed. Η αρχιτεκτονική του Trellis χωρίζεται σε δύο βασικά κομμάτια, αυτά του εκάστοτε εικονικού δικτύου. Στους virtual hosts και στα virtual links.



Σχήμα 3.1: Επισκόπηση της Αρχιτεκτονικής του Trellis

Στην συνέχεια παρουσιάζουμε τις πιο βασικές αποφάσεις των δημιουργών του Trellis που αφορούν αυτά τα στοιχεία.

Virtual Host

Η δημουργία των virtual hosts θα γίνεται με βάση Container-based εικονικοποίηση.
Η πρώτη πιο σημαντική απόφαση είναι ο τρόπος εικονικοποίησης του virtual host. Οι πιο ενδεδειγμένες μέθοδοι για τέτοιες περιπτώσεις είναι το full virtualization, το paravirtualization και το OS-level virtualization, όπως αυτά εξηγήθηκαν στο § 2.2.1. Συνήθως, τα fully virtualized και paravirtualized συστήματα παρέχουν καλύτερο επίπεδο isolation από τα containers, τα οποία όμως έχουν καλύτερες επιδόσεις καθώς αποτελούν πιο ελαφριές αφαιρέσεις από τα virtual machines. Έτσι, οι απαιτήσεις για καλές επιδόσεις, επεκτασιμότητα σε συνδυασμό με ικανοποιητικό isolation και ευελιξία κάνουν επιτακτική την χρήση της εικονικοποίησης βασισμένης σε containters(container based virtualization). Γίνεται συνδυασμός δύο διαφορετικών τέτοιων τεχνολογιών. Του Linux VServer και του NetNS για του λόγους που εξηγούνται στην συνέχεια, μαζί με μία συνοπτική παρουσίαση τους.

Linux VServer: Το Linux-Vserver[6] είναι μία open source υλοποίηση virtual server που εφαρμόζει os-level virtualization στον Linux kernel. Ουσιαστικά αποτελεί έναν μηχανισμό

[6] `http://linux-vserver.org/`

jail που μπορεί να χρησιμοποιηθεί για τον ασφαλή διαμοιρασμό των πόρων ενός υπολογιστικού συστήματος (όπως το file system, η CPU, η μνήμη κ.τ.λ.). Κάθε jail ονομάζεται security context και το εικονικοποιημένο σύστημα που περιέχει είναι ο virtual server. Παρέχεται ένα εργαλείο για την μετακίνηση ανάμεσα στα διάφορα security context. Έτσι η εκκίνηση ενός virtual server είναι απλώς η εκκίνηση του λειτουργικού μέσα σε ένα νέο security context, ενώ το κλείσιμο είναι τερματισμός όλων των διεργασιών που ζουν στο security context. Ένα από τα σημαντικά κριτήρια επιλογής του Linux-Vserver για το server virtualization του Trellis είναι ότι υποστηρίζεται από το Planetlab, άρα και το MyPLC στο οποίο στηρίζεται το Trellis. Η χρήση του όμως συνεπάγεται το σημαντικό μειονέκτημα ότι δεν εικονικοποιεί την δικτυακή στοίβα (network stack) του Linux αλλά στηρίζεται στην απομόνωση. Συνεπώς, δεν μπορεί ο κάθε virtual server να δημιουργεί τις δικές του εσωτερικές ρυθμίσεις για routing και firewalling. Για να καλύψει αυτή την απαίτηση το Trellis χρησιμοποιεί το NetNS.

NetNS: Το Trellis χρησιμοποιεί τα Linux Network NameSpaces[7] (NetNS) για να απομονώσει το network stack του πυρήνα. Μπορεί έτσι να δημιουργήσει πολλαπλά στιγμιότυπα του network stack και επιπλέον να τα απομονώσει μεταξύ τους, ώστε να αποτρέπονται παρεμβάσεις μεταξύ των διαφορετικών virtual networks. Πιο συγκεκριμένα, το NetNS εικονικοποιεί κάθε προσβάσιμο δικτυακό πόρο(π.χ. διευθύνσεις IP, routing tables, interfaces, port numbers κ.τ.λ.), επιτρέποντας έτσι σε κάθε virtual host την επιθυμητή δικτυακή πρόσβαση. Ένα namespace με τους δικτυακούς του πόρους είναι μοναδικό και "δένεται" σε συγκεκριμένες διεργασίες. Μετά την δημιουργία του, γίνεται αόρατο στις αλλές διεργασίες που εκτελούνται εκτός του virtual host.

Virtual Links

Στο Trellis τα virtual links μεταφέρουν την κίνηση μεταξύ δύο virtual hosts. Ένας virtual host παραδίδει το πλαίσιο σε ένα virtual interface, το οποίο το στέλνει σε ένα virtual link. Αφού το πακέτο εξέλθει από τον virtual host διαμέσω του virtual interface μπορεί να ελέγχεται ο ρυθμός μετάδοσης του από κάποιον διαμορφωτή κίνησης (traffic shaper), με σκοπό την επίτευξη του μέγιστου bitrate. Στην συνέχεια, το πλαίσιο παραδίδεται στο tunnel προκειμένου να μεταδοθεί στο άλλο άκρο του virtual link. Σύμφωνα με τους δημιουργούς του Trellis τα virtual links πρέπει να :

- Παρουσιάζονται σαν εικονικά Ethernet links. Ο στόχος αυτής της επιλογής είναι διότι αποτελεί μία ευρέως χρησιμοποιούμενη και οικεία L2 τεχνολογία. Ο αντίκτυπος της στην υλοποίηση είναι ότι ένα virtual link πρέπει να περιέχει την σημασιολογία του Ethernet (π.χ. broadcast domains, point-to-multipoint τοπολογίες) και να υποστηρίζει την μορφή των πλαισίων Ethernet.

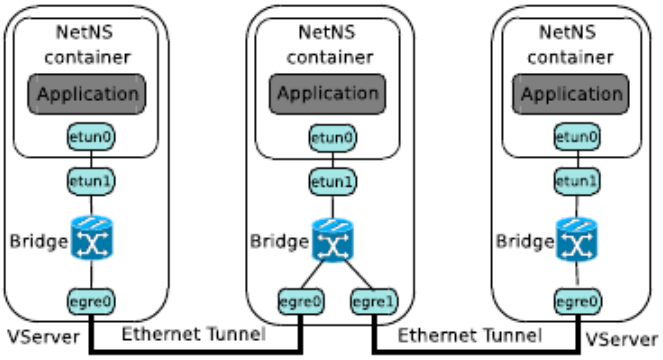
- Χρησιμοποιούν ελαφρείς μηχανισμούς ενθυλάκωσης και αποπολυπλεξίας. Λόγω των πολλαπλών εικονικών συσκευών Ethernet (και των πολλαπλών virtual hosts) που πρέπει να μοιράζονται ένα physical device, το υπόστρωμα πρέπει να διασφαλίζει ότι τα πακέτα αποπολυπλέκονται προς την σωστή εικονική συσκευή.

- Μπορούν να επιβάλλουν περιορισμό του εύρους ζώνης εκτός του πλαισίου του virtual host. Κάθε virtual link μπορεί να έχει ένα όριο εύρους ζώνης, με σκοπό την διασφάλιση του isolation μεταξύ εικονικών δικτύων. Το υπόστρωμα πρέπει να απαγορεύει στο virtual link την υπέρβαση αυτού το καθορισμένου ορίου.

[7] `http://lxc.sourceforge.net/index.php/about/kernel-namespaces/network/`

Υλοποίηση των virtual links με την αποστολή πλαισίων ethernet μέσα από GRE tunnels. Για να καλυφθούν οι στόχοι που περιγράφηκαν έγινε χρήση μίας αρκετά σύγχρονης τεχνολογίας tunneling. Πρόκειται για μία επέκταση του πρωτοκόλλου για tunneling GRE, ώστε να υποστηρίζει την δυνατότητα μεταφοράς πακέτων Ethernet, γνωστή και ως Ethernet over GRE (EoGRE, βλέπε $\oint$2.1.2.2). Το Trellis χρησιμοποιεί το EoGRE σαν μηχανισμό tunneling γιατί ένα σταθερό και μικρό κόστος ενθυλάκωσης και επιπλεόν χρησιμοποιεί ένα κλειδί μεγέθους τεσσάρων byte για να αποπολυπλέξει τα πακέτα και να τα προωθήσει στο σωστό tunnel interface. Λόγω της key-based πολυπλεξίας, τα EoGRE tunnels επιτρέπουν σε ένα virtual network να χρησιμοποιήσει επικαλλυπτόμενο χώρο διευθύνσεων και προσφέρουν την δυνατότητα για την πακέτων πρωτοκόλλων διαφορετικών του IP.

Τερματισμός των tunnels στο "root context", έξω από τα containers των virtual hosts. Τα virtual links του Trellis πρέπει να είναι απομονωμένα από το υπόλοιπο virtual network και πρέπει να είναι ευέλικτα. Για να ικανοποιήσει αυτούς τους στόχους, το Trellis τερματίζει τα virtual links στο root context. Πιο συγκεκριμένα τα EoGRE tunnels τερματίζονται σε root context. Στην συνέχεια μία ενδιάμεση συσκεύη ουρών αναμονής διαμορφώνει την κίνηση κάνοντας χρήση του tc, το Linux traffic control module[8]. Η εικονική συσκευή που βρίσκεται στον virtual host συνδέεται μέσω μίας γέφυρας (bridge) με το άκρο του tunnel. Αυτή η αρχιτεκτονική (σχήμα 3.2) επιτρέπει την εφαρμογή πολιτικών διαμόρφωσης της κίνησης και την υλοποίηση αλγορίθμων χρονοδρομολόγησης που προσφέρουν εγγυήσεις εξυπηρέτησης για κάθε virtual interface. Τέλος, διατηρείται η δυνατότητα των χρηστών να επιβάλλουν τις δικές τους πολιτικές διαμόρφωσης της κίνησης όσον αφορά την δική τους κίνηση.



Σχήμα 3.2: Λεπτομέρειες της τοπολογίας του Trellis

[8]http://tcng.sourceforge.net/

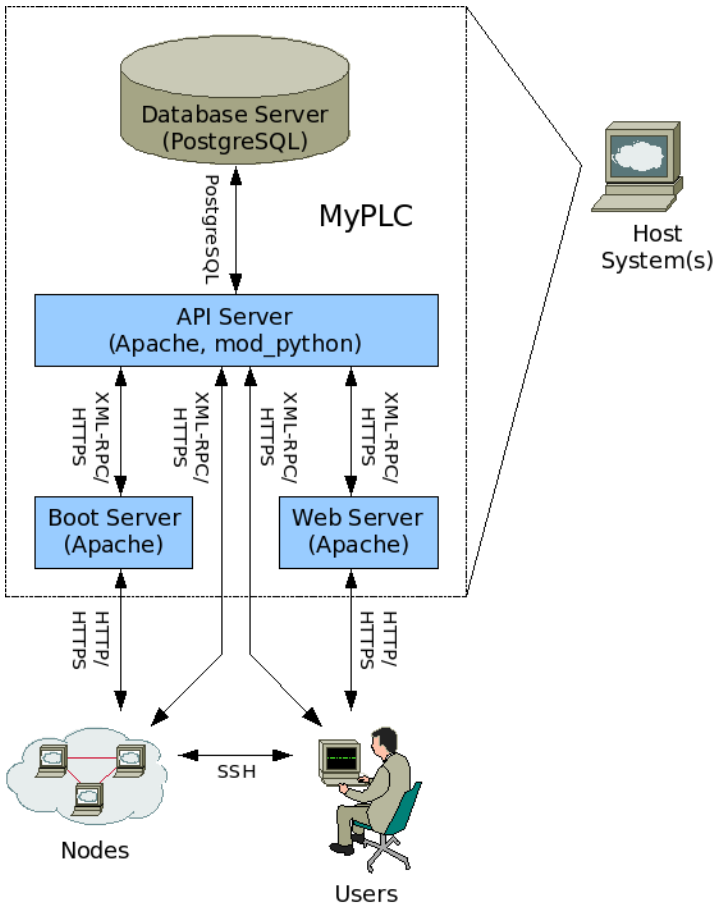
3.1.3 Trellis Control Plane

Αναφέραμε ότι το VINI αποτελεί ουσιαστικά ένα στιγμιότυπο ενός "private Planetlab". Ως αποτέλεσμα, το Control Plane του Trellis είναι βασισμένο στο Control Plane με ενός "private Planetlab", το MyPLC.

3.1.3.1 MyPLC

Το MyPLC είναι μία πλήρης φορητή εγκατάσταση του PlanetLab Central. Η προεπιλεγμένη εγκατάσταση αποτελείται από έναν web server, έναν XML-RPC API server, έναν boot server και έναν database server, τα βασικά στοιχεία δηλαδή του PLC (σχήμα

Το βασικό χαρακτηριστικό του λογισμικού είναι ότι υποστηρίζει κατανεμημένη εικονικοποίηση (distributed virtualization), την δυνατότητα, δηλαδή, να ανατίθεται ένα slice των υλικών πόρων του δικτύου Planetlab σε κάποια εφαρμογή. Έτσι η εφαρμογή αυτή μπορεί να εκτελείται σε όλες ή σε κάποιες από τις μηχανές που βρίσκονται σε όλο τον κόσμο, και την ίδια χρονική στιγμή πολλές άλλες εφαρμογές να εκτελούνται σε διαφορετικά slices.



Σχήμα 3.3: Αρχιτεκτονική του MyPLC

3.1.3.2 Trellis Management Plane

Internet In A Slice (IIAS) Η συλλογή εργαλείων IIAS[6][1] διευκολύνει την δημιουργία ενός πλήρους εικονικού δικτύου μέσα σε ένα slice. Είναι ένα παράδειγμα αρχιτεκτονικής δικτύου που μπορεί να εφαρμοστεί στο PL-VINI και στο Trellis. Οι ερευνητές μπορούν χρησιμοποιήσουν το IIAS προκειμένου να διεξάγουν ελεγχόμενα πειράματα που αξιολογούν τα υπάρχοντα πρωτόκολλα δρομολόγησης IP και τους μηχανισμούς προώθησης κάτω από ρεαλιστικές συνθήκες. Εναλλακτικά, το IIAS μπορεί να χρησιμοποιηθεί σαν μια υλοποίηση αναφοράς (όπως και στην δική μας περίπτωση), η οποία μπορεί να τροποποιηθεί προκειμένου να αξιολογηθούν επεκτάσεις σε σημερινά πρωτόκολλα και μηχανισμούς. Ένα IIAS αποτελείται από πέντε συστατικά:

1. μία μηχανή προώθησης για τα πακέτα που μεταφέρονται από το overlay network (ένας overlay router),
2. μία έξυπνη μέθοδο ρύθμισης των forwarding tables της μηχανής προώθησης (ένα control plane),
3. έναν μηχανισμό προκειμένου οι πελάτες να μπορούν εισάγουν κίνηση στο overlay network ώστε να υπάρχει πραγματική κίνηση (overlay ingress),
4. ένα μέσο ανταλλαγής πακέτων με severs που δεν γνωρίζουν την υπάρξη του overlay network, έτσι εφ'όσον το μεγαλύτερο κομμάτι του διαδικτύου βρίσκεται έξω από το overlay network,
5. μία συλλογή από κατανεμημένους υπολογιστές στους οποίους θα στηρίζεται το overlay network.

Το αρχικό IIAS συνδύαζε τις τεχνολογίες Click, XORP, User-Mode Linux και OpenVPN για να επιτρέψει την εκτέλεση ενός "εικονικού παροχέα Ίντερνετ" (virtual Internet Service Provider(ISP)) σε κάθε slice. Στην συνέχεια όμως το IIAS εξελιχθηκε προκειμένου να υποστηρίζει το Trellis και μετατράπηκε σε framework ελέγχου του VINI. Η νέα έκδοση του IIAS εκτελεί Quagga και OpenVPN επάνω στην εικονική τοπολογία του Trellis. Οι πελάτες συνδέονται σε έναν κόμβο εισόδου (ingress node) μέσω του OpenVPN, και δρομολογούν κίνηση στο IIAS. Από την πλευρά του, το IIAS δρομολογεί την κίνηση στον ορισμένο κόμβο εξόδου (egress node) για το πρόθεμα δικτύου (network prefix) του, όπου η κίνηση εξέρχεται από το NAT. Το Quagga εκτελεί το πρωτόκολλο OSPF για να δημιουργεί τις διαδρομές (routes).

Vsys[9] Το Vsys, ένα εναλλακτικό sudo, είναι ένα εργαλείο που παρέχει την δυνατότητα σε μη προνομιούχους χρήστες να εκτελέσουν εντολές που χρειάζονται αυξημένα δικαιώματα (privileges), όπως το πρόγραμμα sudo στο Linux. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη σε εικονικοποιημένα περιβάλλοντα, όπου οι χρήστες δεν είναι μόνο περιορισμένοι αλλά και απομονωμένοι. Το Vsys υλοποιήθηκε στα πλαίσια της ανάπτυξης του Planetlab με σκοπό να επιτρέπει στους χρήστες να εκτελούν χρήσιμες προκαθορισμένες υπηρεσίες για τις οποίες δεν έχουν εξουσιοδότηση.

Οι διαθέσιμες στον χρήστη (είτε virtual guest) εντολές είναι ένα προκαθορισμένο σύνολο από εκτελέσιμα αρχεία, που μπορούν με ακρίβεια να ελέγχουν το βαθμό πρόσβασης που το slice

[9] `http://www.cs.princeton.edu/~sapanb/vsys/`

έχει σε ξένα πλαίσια. Τα εκτελέσιμα αυτά αρχεία βρίσκονται τοποθετημένα σε συγκερκιμένο κατάλογο (directory) στο πλαίσιο εξυπηρέτησης. Στα slices που εγγράφονται σε αυτές της υπηρεσίες δημιουργείται ένα ζεύγος από fifo pipes (ή ένα unix domain socket) για κάθε υπηρεσία. Αυτά τα pipes (ή τα sockets αντίστοιχα) αποτελούν τα κανάλια επικοινωνίας (εισόδου και εξόδου) με τις υπηρεσίες. Ο μηχανισμός πιστοποίησης του Vsys βασίζεται στο isolation του συστήματος αρχείων (filesystem) είτε usenix-permission-based[10] isolation και δεν απαιτεί καμμία ρητή συναλλαγή όταν καλούνται οι privileged υπηρεσίες.

Τα βασικά πλεονεκτήματα του Vsys είναι η δυνατότητα της ανάπτυξης scipts για τον χρήστη σε οποιαδήποτε γλώσσα προγραμματισμού, η δυνατότητα να την ενεργοποίησης των scritps αυτών δυναμικά και η δυνατότητα τού αποτελεσματικού και με λεπτομέρεια περιορισμού δικαιωμάτων (π.χ. μερική πρόσβαση σε ένα αρχείο). Επιπρόσθετα, τα Vsys scritps μπορούν να χρησιμοποιηθούν με απλά εργαλεία UNIX όπως τα cat, echo και grep.

3.2 Virtual Switching: Open vSwitch

Τα δίκτυα, σε εικονικοποιημένα περιβάλλοντα, παρουσιάζουν νέες ευκαρίες και προβλήματα. Ωστόσο, το τυπικό διαδικτυακό μοντέλο σε αυτά τα περιβάλλοντα αποτελείται από τον κλασσικό L2 switch ή L3 router σε επίπεδο hypervisor ή στο επίπεδο διαχείρισης υλικού του εικονικού περιβάλλοντος. Αυτά τα εικονικά δικτυακά στοιχεία διαχειρίζονται την επικοινωνία μεταξύ virtual machines που βρίσκονται στην ίδια φυσική τοπθεσία, καθώς και την επικοινωνία με την φυσική NIC. Είναι υλοποιημένα σε software και συνήθως τοποθετούνται στο πλαίσιο του host.

Το Open vSwitch είναι μια προσπάθεια να προσαρμοστεί το εικονικό επίπεδο δικτύου στο σύνολο των ιδιοτήτων του[14]. Χρησιμοποιήθηκε, λοιπόν, για να διατελεί τον ρόλο του εικονικού switch.

3.2.1 Open vSwitch Platform

Το Open vSwitch[11], είναι ένας λογικός δικτυακός μεταγωγέας (vswitch) ειδικά προορισμένος για εικονικά περιβάλλοντα.

Η διαφορά του σε σχέση με άλλες παρόμοιες προσεγγίσεις έγκειται στο γεγονός ότι παρέχει μία εξωτερική διεπαφή (interface) για συμπαγή δομικά (fine-grained), έλεγχο της συμπεριφοράς προώθησης, που μπορεί να υποστηρίξει λειτουργίες QoS, tunneling kai filtering κανόνων. Επίσης, υποστηρίζει μία απομακρυσμένη διεπαφή που επιτρέπει την μετανάστευση (migration) του παραμετροποιημένου στιγμιοτύπου (configuration state) (χρήσιμο στην πρόσαψη δικτυακων πολιτικών στα virtual machines). Επιπλέον, η υλοποίηση του παρέχει μια ευέλικτη μηχανή προώθηση, βασισμένη σε πίνακες, που μπορεί να χρησιμοποιηθεί για την λογική τμηματοποιήση (partition) του forwarding plane. Τέλος, έχει την δυνατότητα συνεργασίες με τα περισσότερα Linux-based περιβάλλοντα εικονικοποίησης όπως Xen[12],XenServer[13], KVM[14] και QEMU[15].

[10] `http://www.usenix.org/`

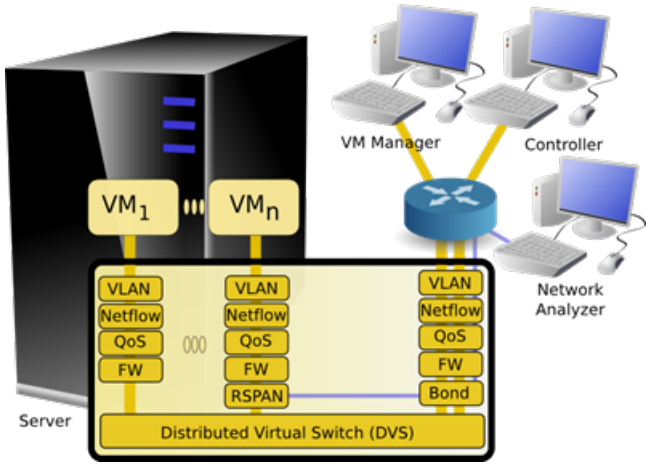
[11] `http://www.openvswitch.org/`

[12] `http://www.xen.org/`

[13] `http://www.citrix.com/English/ps2/products/product.asp?contentID=683148`

[14] `http://www.linux-kvm.org/`

[15] `http://qemu.org/`



Σχήμα 3.4: Επισκόπηση του Open vSwitch

Ένα ακόμη πολύ σημαντικό χαρακτηριστικά του Open vSwitch, αν και εκτός των πλαισίων αυτής της διπλωματικής είναι η συμβατότητα του με OpenFlow switches και controllers. Αυτό το χαρακτηριστικό διευρύνει σημαντικά της ικανότητες του Open vSwitch ως εργαλείο για το Ίντερνετ του Μέλλοντος.

Πιθανές χρήσεις του είναι για την επίλυση προβλημάτων όπως την απομόνωση σε διαμοιραζόμενα περιβάλλοντα, κινητικότητα μεταξύ υποδικτύων, κατανεμημένων τοπολογιών και ορατότητας μεταξύ κόμβων[14].

3.2.2 Αρχιτεκτονική του Open vSwitch

3.2.2.1 Επισκόπηση

Το Open vSwitch[14] είναι λογισμικό που τοποθετείται στον επίπεδο του mangement domain (hypervisor ή σε άλλες περιπτώσεις host kernel space). Παρέχει συνδεσιμότητα μεταξύ των virtual machines και των φυσικών interfaces. Υλοποιεί το τυπικό Ethernet switching με δυνατότητες για VLAN, RSPAN και βασικό ACL. Μπορεί να χρησιμοποιηθεί και αυτόνομα, όπως στην περίπτωση που μελετάμε εμείς, σαν ένας standard L2 μεταγωγέας. Για υποστηρίξει, όμως, την σύνδεση με εικονικά περιβάλλοντα παρέχει διεπαφές για την διαχείριση του forwarding state και managing configuration state κατά σε περιβάλλον εκτέλεσης.

Παραμετροποίηση(configuration):Διαμέσου της διεπαφής παραμετροποίησης μία απομακρυσμένη διαδικασία μπορεί να διαβάσει αλλά και να μεταβάλλει το configuration state (ζεύγη κλειδί/τιμή) και να δημιουργήσει triggers που ενεργοποιούνται από ασύγχρονα γεγονότα και μεταβάλλουν το configuration state. Υπάρχει δυνατότητα για χρήση των ιδιοτήτων που περιγράφονται παρακάτω στο Παράρτημα Β΄.2. Επιπλέον, αυτή η διεπαφή παρέχει την δυνατότητα σύνδεσης δικτυακών ports με το γενικότερο εικονικό περιβάλλον. Για παράδειγμα, η διεπαφή παρέχει την δυνατότητα ορισμού καθολικά μοναδικών αναγνωριστικών (UUIDs) για εικονικά interfaces του switch. Αυτή η δυνατότητα είναι απαραίτητη για την ανεξαρτητοποίηση της παραμετροποίησης από χαρακτηριστικά τοπολογίας.

Μονοπάτι Προώθησης (Forwarding Path): Η διαχείριση της παραμετροποίησης του μεταγωγέα, όπως περιγράφηκε παραπάνω, είναι κάτι συνηθισμένο και στους φυσικούς μεταγωγείς. Το

Open vSwitch παρέχει επιπλέον την δυνατότητα της απομακρυσμένης διαχείρισης του forwarding path. Με αυτόν τον τρόπο, δίνεται σε εξωτερικές διεργασίες άμεση πρόσβαση στο forwarding table, προσδιορίζοντας την συμπεριφορά πακέτων βάσει των L2,L3 και L4 επικεφαλίδων. Το lookup μπορεί να αποφασίσει να προωθήσει τα πακέτα σε μία ή περισσότερες πόρτες, να απορρίψει το πακέτο, είτε να του εφαρμόσει en/decapsulation. Η διεπαφή του forwarding path υλοποιεί ένα υπερσύνολο των λειτουργιών του πρωτόκολλου OpenFlow.

Διαχείριση Συνδεσιμότητας (Connectivity management): Το Open vSwitch παρέχει μία τοπική διεπαφή διαχείρισης, μέσα από το οποία το virtualization layer μπορεί να διαχειριστεί την τοπολογική παραμετροποίηση. Η διεπαφή επιτρέπει την δημιουργία εικονικών μεταγωγέων, διαχείριση της συνδεσιμότητας των Virtual Interfaces (VIFs) (για κάθε συνδεδεμένο VIF προστίθεται ένα logical port στο switch), αλλά και διαχείριση συνδεσιμότητας των Physical Interfaces (PIFs).

Κάτω από το Open vSwitch βρίσκεται ένα flow-table forwarding model. παρόμοιο με αυτό που χρησιμοποιείται από το OpenFlow. Το rule-based forwarding αποσκοπεί στην επίτευξη ενός σχεδόν αυθαίρετου logical partitioning των διαδικασιών του forwarding. Πιο συγκεκριμένα, επιτρέπει την σύνδεση μεταξύ του δικτυακού configuration state και των διαδικασιών του forwarding με ένα υποσύνολο της κίνησης, είτε από ένα VIF, ένα VM, είτε ένα σύνολο από VMs.

Στην απλούστερη χρήση του, το Open vSwitch συμπεριφέρεται όπως ένα παραδοσιακό physical switch μέσα στο virtualization layer. Κάθε στιγμιότυπο διαχειρίζεται ξεχωριστά διαμέσου των διεπαφών διαχείρισης, παρέχοντας ορατότητα και έλεγχο πάνω σε inter-VM επικοινωνίες, που είναι ορατές στο first hop physical switch. Ωστόσο, η συμπερίληψη των interfaces για καθολικό managing configuration και forwarding state επιτρέπει την κατανομή των λειτουργιών του switch σε πολλαπλούς servers, αποσυνδέοντας έτσι αποτελεσματικά την λογική δικτυακή τοπολογία από την φυσική. Για παράδειγμα, μια απομακρυσμένη διαδικασία, εφόσον είναι ενσωματωμένη στην πλατφόρμα ελέγχου του virtualization, μπορεί να κάνει migrate το network configuration state ταυτόχρονα με τα(VMs), όπως αυτά μετακινούνται μεταξύ των φυσικών servers.

Επιπρόσθετα, η δυνατότητα της διαχείρισης του forwarding table διαμέσω ενός εξωτερικού interface επιτρέπει στην χαμηλού επιπέδου flow state να κάνει migrate μαζί με το VM. Αυτό θα ήταν χρήσιμο την μεταφορά των υπαρχόντων flow counters και ACLs. Επίσης επιτρέπει το migration κανόνων που αφορούν tunneling, δυνατότητα χρήσιμη στο migration μεταξύ διαφορετικών υποδικτύων IP.

3.2.2.2 Υλοποίηση

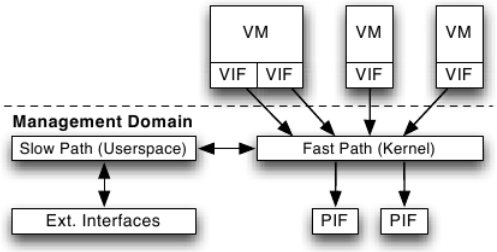
Σε αυτή την παράγραφο κάνουμε μια στοιχειώδη παρουσίαση της υλοποίησης του Open vSwitch, καθώς αυτό θα μας επιτρέψει να κατανοήσουμε σε μεγαλύτερο βάθος την λειτουργία του.

Στην υλοποίηση υπάρχουν δύο βασικά συστατικά: ένα "γρήγορο μονοπάτι"(fastpath) που εντοπίζεται στο kernel-space καθώς και ένα "αργό μονοπάτι"(slowpath) στο user-space.

Στο **fast path** υλοποιείται η μηχανή του forwarding που είναι υπεύθυνη για το ανά πακέτο lookup, την τροποποίηση και το forwarding. Επιπλέον, διατηρεί και τους μετρητές για κάθε εγγραφή του πίνακα forwarding. Αυτό το κομμάτι του συστήματος είναι το πιο επίφοβο,

όσον αφορά την ταχύτητα. Αυτός είναι ένας από τους βασικούς λόγους που επιλέχθηκε να τοποθετηθεί στο fast path.

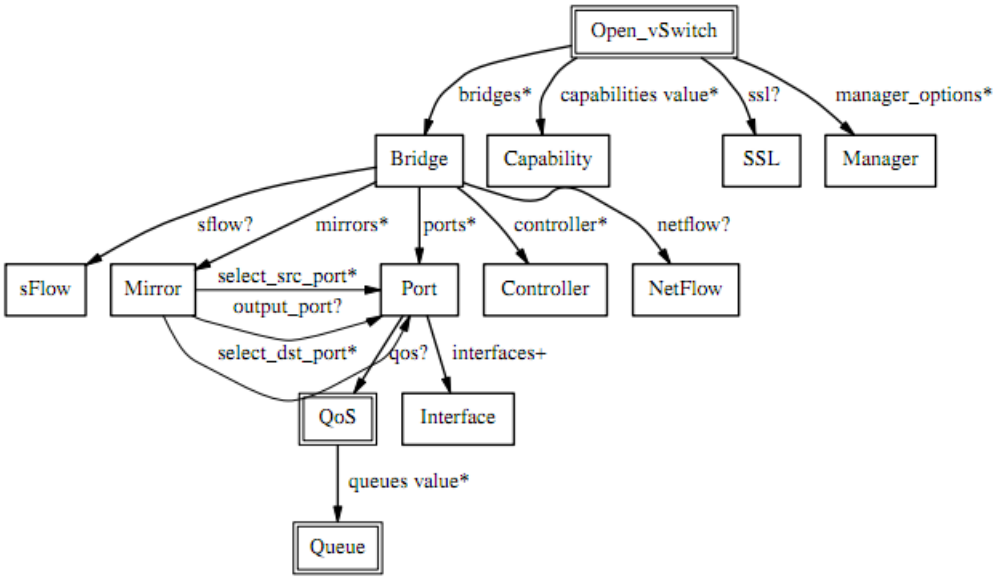
Το μεγαλύτερο μέρος της λειτουργικότητας υλοποιείται στο **slow path**, που εκτελείται στο domain της διαχείρισης του VM, ώστε να μειωθεί το μέγεθος του system specific κωδικά (kernel-space), χωρίς μεγάλο κόστος σε επιδόσεις. Υλοποιεί το forwarding logic, συμπεριλαμβανομένων των MAC learing και load-balancing σε bonded interfaces. Ακόμα, υλοποιεί την απομακρυσμένη ορατότητα (remote visibility) και configuration interfaces, όπως για τα NetFlow, OpenFlow και πρωτόκολλα απομακρυσμένης διαχείρισης.



Σχήμα 3.5: Αρχιτεκτονική του Open vSwitch

Open vSwitch Database Schema

Μία βάση δεδομένων, με δομή όπως απεικονίζεται παρακάτω(3.6), διατηρεί τις παραμέτρους για κάθε Open vSwitch daemon. Η υψηλότερου επιπέδου ρύθμιση του daemon καθορίζεται από τον πίνακα Open vSwitch. Εγγραφές στους υπόλοιπους πίνακες έχουν νόημα μόνο εφ'όσον μπορούμε να φτάσουμε σε αυτές άμεσα ή έμμεσα διαμέσω του πίνακα Open vSwitch.



Σχήμα 3.6: Open vSwitch schema

Οι πίνακες που θα μας απασχολήσουν στα πλαίσια της διπλωματικής είναι οι Bridge, Port και Interface, των οποίων τα χαρακτηριστικά που χρησιμοποιήθηκαν παρουσιάζονται στο

Παράρτημα Β′.

Κεφάλαιο 4

Αρχιτεκτονική Εγκαθίδρυσης L2 Εικονικών Τοπολογιών

4.1 Προσθήκη εικονικού μεταγωγέα στο Trellis

Σκοπός μας είναι η δημιουργία μίας πλατφόρμας που θα παρέχει L2 connectivity και θα επιτρέπει στους ερευνητές τον πειραματισμό με υπηρεσίες και επεκτάσεις πρωτοκόλλων και σε αυτό το επίπεδο. Μέχρι πρόσφατα οι ερευνητές δεν είχαν την δυνατότητα να διεξάγουν τέτοια πειράματα σε μεγάλης κλίμακας δίκτυα, ώστε να παράγουν αξιόπιστα αποτελέσματα. Το Trellis είναι μια προσπάθεια να ξεπεραστεί αυτό το εμπόδιο. Η τρέχουσα αρχιτεκτονική και υλοποίηση του όμως έχει σαν βάση την δημιουργία ζεύξεων σημείου προς σημείο (point-to-point). Η προοπτική αυτή εμποδίζει τους χρήστες-ερευνητές να μελετήσουν συμπεριφορές σε Ethernet switched LANs, που αποτελούν την κυρίαρχη δομή, όσον αφορά τα τοπικά δίκτυα υπολογιστών.

Η προσέγγιση μας λοιπόν αποσκοπεί στην προαιρετική ανάθεση των switching λειτουργιών ενός slice σε ένα sliver, το switch sliver. Αυτός το sliver θα εντοπίζεται σε έναν τυπικό Trellis node, και η τροποίηση αφορά το switch sliver και θα είναι συνδεδεμένο με όλα τα υπόλοιπα slivers που ανήκουν στο ίδιο slice. Η μετατροπή του sliver σε switch sliver δεν συνίσταται μόνο στην εκτέλεση διεργασιών ελέγχου του switching (π.χ. ACLs, QoS) στο sliver αυτό αλλά και στην αυτόματη δημιουργία της συνδεσμολογίας του LAN, συμπεριλαμβάνοντας όλους τους κόμβους του slice

4.1.1 Αρχιτεκτονική

Η λήψη των αποφάσεων για την αρχιτεκτονική που υλοποιήθηκε στηρίχθηκε στα παρακάτω βασικά κριτήρια.

- **Πλήρης L2 Συνδεσιμότητα**: Είναι πολύ σημαντικό η πλατφόρμα που θα προκύψει να παρουσίαζει στους χρήστες όσο το δυνατόν πιο πλήρη L2 συνδεσιμότητα. Δηλαδή η υποδομή να είναι καθορισμένη με τέτοιο τρόπο ώστε να μην κληρονομεί περιορισμούς στους χρήστες καθώς αυτό θα την καθιστά "δύσπεπτη" και θα αυξήσει την πιθανότητα για λανθασμένη χρήση.

- **Επεκτασιμότητα**: Αποτελεί από τους πιο σημαντικούς παράγοντες σε περιβάλλοντα εικονικοποίησης καθώς, ένας από τους σκοπούς της είναι η ενοποίηση πολλών συστημάτων.
- **Ομοιογένεια**: Η ομοιογένεια σε μία τέτοια πλατφόρμα παίζει σημαντικό ρόλο καθώς κάνει απλούστερη και πιο σωστή την διαχείριση της. Η ομοιογένεια πρέπει να διατηρείται στα πλαίσια κάθε αφηρημένης οντότητας(π.χ. simple node, switch node). Σε ευρύτερο πλαίσιο η ομοιογένεια κάνει την διαχείριση της πλατφόρμας πιο ομαλή και αποδοτική.
- **Ταχύτητα**: Η ταχύτητα είναι βασικός παράγοντας, αλλά όχι από την άποψη της επιτάχυνσης. Δεν έχουμε σαν σκοπό να επιταχύνουμε το υπάρχον σύστημα αλλά να διατηρήσουμε αμετάβλητες τις αρχικές επιδόσεις (π.χ. του Trellis). Έτσι θα έχουμε καταφέρει να προσθέσουμε επιπλέον λειτουργικότητα χωρίς να επιβαρύνουμε το υπάρχον σύστημα. Φυσικά, κάθε απόφαση που αποφέρει και επιτάχυνση και λειτουργικότητα είναι ευπρόσδεκτη.

Με βάση αυτούς του στόχους προέκυψαν οι αποφάσεις για την αρχιτεκτονική της εκτεταμένης πλατφόρμας. Στην συνέχεια παρουσιάζουμε και αιτιολογούμε αναλυτικά τις αποφάσεις

4.1.1.1 Ορισμός του switch sliver

Δημιουργούμε μία καινούργια αφηρημένη οντότητα. Αυτή του switch sliver. Αυτό το sliver δεν πρέπει να αντιμετωπίζεται από τον χρήστη σαν ένα απλό sliver. Ο ρόλος του είναι αυτός της διαχείρισης του virtual switch με χρήση των εργαλείων που θα παρέχονται. Ιδανικά, ο ιδιοκτήτης του slice, θα μπορεί να συνδέεται στο sliver αυτό και να κάνει monitoring της κίνησης του virtual switch (με χρήση NetFlow, sFlow) , να εφαρμόζει πολιτικές QoS και να διαχειρίζεται την υπερκείμενη τοπολογία (π.χ. με χρήση flow based forwarding). Όλες αυτές είναι δυνατότητες που αυξάνουν σε πολύ μεγαλό βαθμό την ευελιξία και τις δυνατότητες πειραματισμού του ερευνητή.

Η προεπιλεγμένη αρχιτεκτονική του IIAS δημιουργεί virtual point-to-point links δύο μεμονωμένων slivers. Εμείς, χρησιμοποιούμε την αρχή του Trellis να βασίζεται σε point-to-point links για να επεκτείνουμε το L2 broadcast domain του κάθε sliver. Ενώ πριν, κάθε κόμβος συνδεόταν μόνο με τους γειτονικούς του, μέσω του switch sliver προσθέτει στο broadcast domain του κάθε sliver που ανήκει στο ίδιο slice.

4.1.1.2 Open vSwitch αντί Linux bridge

Το Linux OS περιλαμβάνει ένα ενσωματωμένο L2 switch (το Linux Bridge) που μπορεί να χρησιμοποιηθεί από τα VMs για συνδεσιμότητα μεταξύ των VMs, ακόμα και αν βρίσκονται σε διαφορετική φυσική υποδομή. Παρ'όλα αυτά το Open vSwitch έχει υλοποιηθεί για χρήση σε εικονικές διατάξεις πολλαπλών server, ένας χώρος στον οποίο η υπάρχουσα δομή δεν αποδίδει τόσο καλά. Αυτά τα περιβάλλοντα συχνά χαρακτηρίζονται από δυναμικά end-points, την διαχείριση λογικών αφαιρέσεων και μερικές φορές την συνεργασία με ειδικού σκοπού switching hardware. Στην συνέχεια απαριθμούμε τις σχεδιαστικές αρχές του Open vSwitch που το καθιστούν επικρατέστερο για την αρχιτεκτονική μας από το Linux Brigde:

1. **mobility of state**: Το Open vSwitch υποστηρίζει την ρύθμιση και την μεταφορά τόσο των ρυθμίσεων όσο και του στιγμιοτύπου της κατάστασης του δικτύου. Για παράδειγμα,

στην περίπτωση που ένα virtual machine μετακινείται μεταξύ end-hosts, είναι δυνατή η μεταφορά μαζί του όχι μόνο των αντίστοιχων ρυθμίσεων (κανόνες SPAN, ACLs, QoS) αλλά και όλη η τρέχουσα δικτυακή κατάσταση (π.χ. το στιγμιότυπο της κατάστασης, που μπορεί να είναι δύσκολο να ανακασκευαστεί). Επιπλέον, η κατάσταση του Open vSwitch καταγράφεται και αποθηκεύεται από ένα πραγματικό μοντέλο δεδομένων (data-model) επιτρέποντας την ανάπτυξη δομημένων συστημάτων αυτοματισμού.

2. **Responding to network dynamics**: Τα εικονικά περιβάλλοντα, συχνά, χαρακτηρίζονται από μεγάλο βαθμό μεταβολών. Τα virtual machines μεταφέρονται, επαναφέρονται σε παλιότερες και νεότερες χρονικές στιγμές, και το λογικά περιβάλλον μεταβάλλεται. Το Open vSwitch έχει έναν αριθμό από χαρακτηριστικά προκειμένου να προσαρμόζεται στις αλλαγές αυτές. Εκτός από την υποστήριξη για NetFlow και sFlow που μπορούν να έχουν ορατότητα των μεταβολών, η βάση δεδομένων δικτυακής κατάστασης (OVSDB) υποστιρίζει απομακρυσμένη ενεργοποίηση γεγονότων (remote triggers). Αυτό χρησιμεύει στην δημιουργία software που παρακολουθεί το δίκτυο και δρα ανάλογα με τις αλλαγές. Αυτή η τεχνική χρησιμοποιείται ευρέως σήμερα, για παράδειγμα στην διαχείριση των μεταφορών των virtual machines. Επιπλέον, η υποστήριξη για OpenFlow δίνει επίσης δυνατότητα για απομακρυσμένο έλεγχο της κίνησης.

3. **Maintenance of logical tags**: Distributed virtual switches (οπώς το VMware vDS και το Nexus 1000 V της Cisco) συχνά διατηρούν ένα λογικό επίπεδο στο δίκτυο προσθέτωντας ή με τον χειρισμό ετικετών (tags) στα πακέτα του δικτύου. Η τεχνική αυτή μπορεί να χρησιμοποιηθεί για να προσδιοριστεί μοναδικά ένα virtual machine (με τρόπο ανθεκτικό στο hardware spoofing), ή διατηρήσει κάποιο λογικό πλαίσιο σχετικό μόνο με το λογικό επίπεδο. Έτσι, η διαχείριση των tags μπορεί να παρέχει την ίδια λειτουργία που παρέχει ένα distributed virtual switch. Επιπλέον, οι κανόνες του tagging αποθηκεύονται σε μία βλετιστοποιημένη μορφή και όχι σε μη αποδοτικό συνδυασμό με κάποια δικτυακή συσκευή, οπότε και είναι δυνατή η δημιουργία, τροποποίηση και μεταφορά χιλιάδων κανόνων.

4. **EoGRE support**: Το Open vSwitch παρέχει μία δική του υλοποίηση για EoGRE tunnels, που αποτελούν βασικό κομμάτι της αρχιτεκτονικής του Trellis. Υποστηρίζει ακόμα πλήρη απομακρυσμένη διαχείριση των tunnels, χρήσιμο για την σύνδεση εικονικών δικτυών με διαφορετικά data centers.

5. **Hardware integration**: Το μονοπάτι προώθησης (forwarding path) του Open vSwitch βρίσκεται σε kernel context και έχει σχεδιαστεί ώστε μεταφέρει την διαχείριση των πακέτων στο hardware. Έτσι, το μονοπάτι ελέγχου (control path) μπορεί να διαχειριστεί μία software υλοποίηση αλλά και ένα hardware switch. Το hardware integration δεν προσφέρει μόνο βελτίωση των επιδόσεων στα virtualized environments αλλά και επιτρέπει (σε περίπτωση που το Open vSwitch ελέγχει και φυσικές συσκευές) την δημιουργία ενός ενοποιημένου αυτόματου δικτυακού ελέγχου των εικονικών και μη περιβαλλόντων.

Βλέπουμε, λοιπόν, ότι το Open vSwitch απευθύνεται σε διαφορετικά σενάρια χρήσης από το Linux Bridge και το Linux networking stack, εστιάζοντας στην ανάγκη για αυτοματοποιημένο και δυναμικό έλεγχο του δικτύου σε ευρείας κλίμακας εικονικοποιημένα περιβάλλοντα βασισμένα στο Linux. Σε αυτό το πλαίσιο, αποφασίζουμε να υλοποιήσουμε το virtual switch κάνοντας χρήση της τεχνολογίας Open vSwitch.

4.1.1.3 Χρήση ενός vswitch σε κάθε node

Απόρροια της προηγούμενης απόφασης είναι να αντικαταστήσουμε στην υπάρχουσα αρχικτεκτονική τα linux bridges με vswitches. Σύμφωνα με του δημιουργούς του Open vSwitch, είναι εξίσου αποδοτικό με Linux bridges. Ωστόσο, όπως δείξαμε παραπάνω πρόκειται για ένα εργαλείο με αυξημένες δυνατότητες, πιο "έξυπνο" από ένα Linux bridge. Έτσι αποφασίσαμε να αντικαταστήσουμε όλα τα linux bridge instances κάθε κόμβου με ένα μοναδικό Open vSwitch instance. Η απόφαση αυτή δημιουργεί προβλήματα στο isolation της κίνησης των διαφορετικών slices. Τον τρόπο επίτευξης του isolation τον συζητάμε στην §4.1.1.5. Με αυτή την αντικατάσταση δίνουμε μία ομοιογενή μορφή στο συνολικό σύστημα και εξοικονομούμε πόρους στο πλαίσιο του host. Επίσης δίνουμε την δυνατότητα στον network administrator της υποδομής να μπορεί να καθορίσει περιοριστικές πολιτικές στα virtual links, λειτουργία που είναι χρήσιμη για την ισοκατανομή των πόρων στους χρήστες, καθώς και να αποκτήσει ολοκληρωτικό έλεγχο πάνω στην κίνση των nodes, διευκολύνοντας με αυτό το τρόπο την διαχείριση του συστήματος.

4.1.1.4 Υλοποίηση του switching σε host context

Η υλοποίηση του switching μπορεί να γίνεται είτε σε host context, δηλαδή στην υποδομή του virtual network, είτε σε user context, μέσα στο sliver.

Δεδομένης, της απόφασης να χρησιμοποιούμε ένα vswitch σε κάθε κόμβο στο host context, επιλέγουμε αυτό το vswitch να υλοποιεί και το switching. Ώς αποτέλεσμα, ένα Open vSwitch instance υλοποιεί ταυτόχρονα την σύνδεση (stitching) των EoGRE end-points και των κόμβων, αλλά και τις λειτουργίες που αντιστοιχούν και ζητούνται από το switch sliver. Η διεπαφή ελέγχου του χρήστη στο switching θα γίνεται με Vsys εντολές προκειμένου να εξασφαλιστεί το isolation.

Εναλλακτικά θα μπορούσαμε να εγκαταστήσουμε ένα καινούργιο vswitch μέσα στο sliver προκειμένου ο χρήστης να έχει άμεση πρόσβαση σε αυτό. Κάτι τετοιο όμως θα ήταν αρκετά μη αποδοτικό καθώς ο κώδικας του vswitch θα εκτελείτο σε εικονικό περιβάλλον. Επιπλέον, μειώνεται η ρεαλιστικότητα και το επίπεδο ελέγχου των πειραμάτων, καθώς με αυτή την αρχιτεκτονική η κίνηση θα έπρεπε να περνάει μέσα από δύο vswitch, με πιθανώς διαφορετικές ρυθμίσεις.

4.1.1.5 Isolation με χρήση της τεχνολογίας VLAN

Η χρήση ενός vswitch για όλα τα slivers στην υποδομή ενός node προκαλεί εμφανή προβλήματα σχετικά με το isolation της κίνησης του κάθε slice. Χρειαζόμαστε λοιπόν μία τεχνική προκειμένου να ομαδοποιήσουμε τα ports του vswitch που αφορούν το ίδιο slice. Σαν λύση στο πρόβλημα αυτό στο πλαίσιο το Open vSwitch διακρίνουμε δύο βασικές τεχνικές: την χρήση κανόνων flow και την χρήση VLAN.

Η ομαδοποιήση των port ενός switch με την χρήση VLAN αποτελεί σήμερα μία από τις σημαντικότερες λειτουργίες των physical switches. Είναι ο κυρίαρχος τρόπος δημιουργίας L2 broadcast domains από network segments που συνδέονται στο switch καθώς και εξασφάλισης ενός επιπέδου ασφαλείας στα τοπικά υποδίκτυα. Θέλοντας να προσομοιώσουμε, λοιπόν, την λειτουργία ενός L2 broadcast domain η χρήση των VLANs μοιάζει ιδανική. Εφ 'οσον, όμως αναφερόμαστε σε μία πλατφόρμα που παρέχει την δυνατότητα πειραμάτων πρέπει να λάβουμε

υπ'όψην μας το γεγονός ότι χρησιμοποιόντας VLANs για την δημιουργία της υποδομής εμποδίζουμε τους πειραματισμούς με τεχνολογίες VLAN.

Η εναλλακτική προσέγγιση είναι να οριστούν κανόνες flow στο vswitch με σκοπό την προώθηση πακέτων με βάση τις MAC προορισμού και διεύθυνσης, χωρίς την διενέργεια lookup σε κάποια άλλη δομή του vswitch παρά μόνο εκείνη των flows. Οι κανόνες flow του Open vSwitch αποτελούν ένα υπερσύνολο των κανόνων flow του OpenFlow. Αυτή η λύση παρέχει L2 connectivity, όμως οδήγεί σε αλλαγή της αφαίρεσης του virtual switch σε virtual hub. Δηλαδή, οδήγεί σε απώλεια της έννοιας του MAC learning, αφήνωντας σαν μόνη λύση προώθησης το flooding. Η απώλεια αυτή λειτουργικότητας μας οδήγησε στο συμπέρασμα ότι ο καθορισμός του isolation με την χρήση κανόνων flow χρειάζεται την διαχείριση από μία πιο "έξυπνη" οντότητα, για παράδειγμα ενός OpenFlow controller. Μία τέτοια προσέγγιση υπερβαίνει τον αρχικό μας σκοπό καθώς υπάρχουν απλούστερες εναλλακτικές.

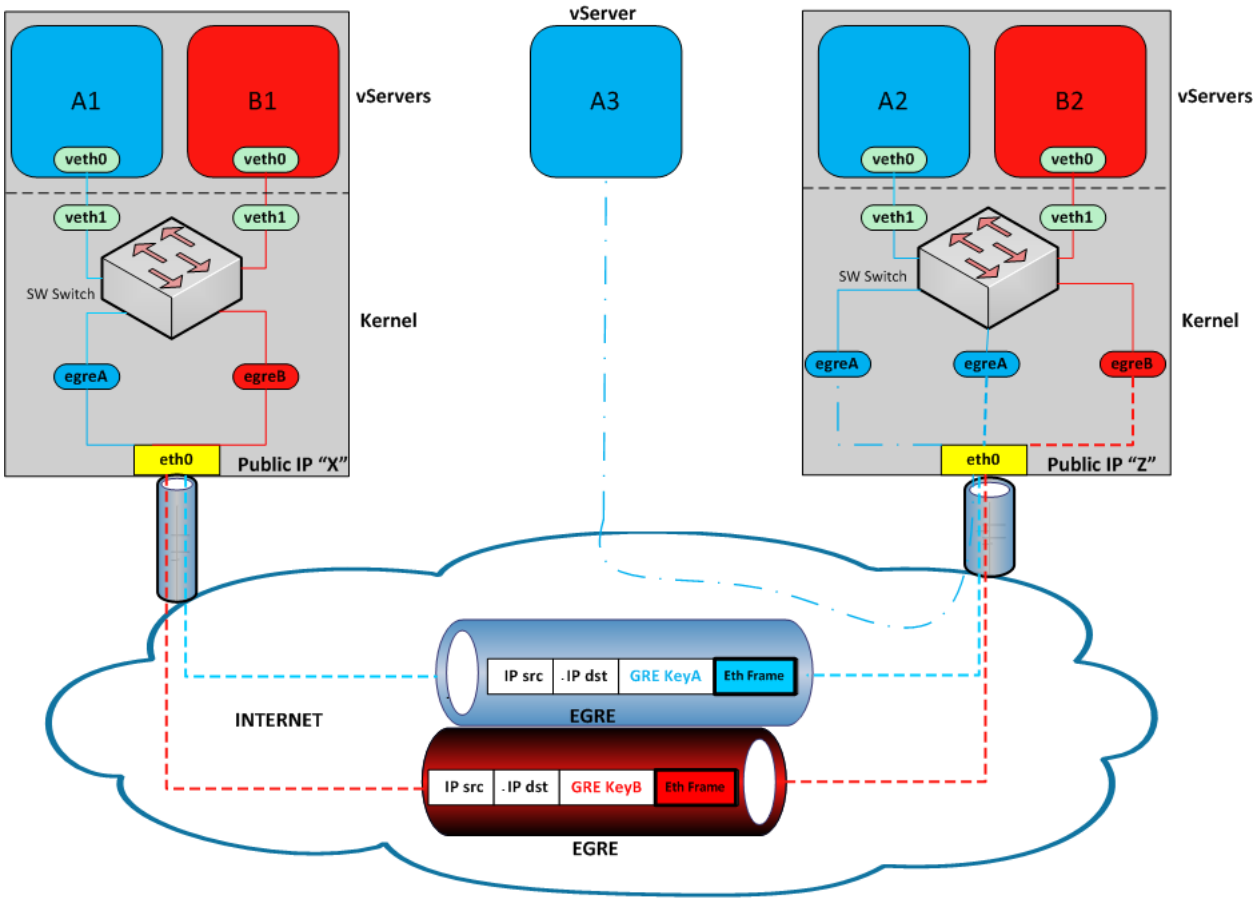
Καταλήγουμε λοιπόν στην χρήση VLANs παρα τον περιορισμό που επιβάλλουν. Η άρση του εμποδίου για χρήση των VLAN από τους users αποτελεί μία από τις προτάσεις για μελλοντική εργασία. Ο ιδανικός τρόπος τρόπος να γίνει αυτό θα ήταν με την επέκταση του Open vSwitch για την υποστήριξη του πρωτοκόλλου 802.1QinQ[1] (γνωστό και ως 802.1ad), το επιτρέπει διπλό VLAN tagging.

4.1.1.6 Διαγραφή των πλεοναζόντων δικτυακών συσκευών στην υποδομή του switch sliver

Από την αρχιτεκτονική που έχει προκύψει μπορούμε να παρατηρήσουμε ότι στον switch sliver κάθε slice χρειάζεται μόνο ένα μονοπάτι σύνδεσης με το vswitch. Έτσι, χωρίς βλάβη της λειτουργία του συστήματος, μπορούμε να διαγράψουμε τα επιπλέον μονοπάτια. Η επιλογή αυτή δυναμώνει επίσης την αφαίρεση του switch sliver, καθώς του δίνει ξεχωριστά χαρακτηρίστικα από τα υπόλοιπα nodes. Το switch slice δεν χρησιμοποιείται σαν άλλος ένας κόμβος του πειράματος αλλά σαν το switch μιας τοπολογίας με της αντίστοιχες δυνατότητες. Το μονοπάτι που διατηρούμε είναι για να επιτρέπουμε στον χρήστη να κάνει monitor την κίνηση που περνάει από το switch.

Μετα, από όλες τις αποφάσεις καταλήγουμε στην αρχικτεκτονική που περιγράφεται στο σχήμα 4.1.

[1] http://www.ieee802.org/1/pages/802.1ad.html



Σχήμα 4.1: Τροποιημένη δικτυακή υποδομή στο περιβάλλον Trellis

4.2 L2 connectivity μεταξύ κόμβων από federated testbeds

Μέχρι τώρα παρουσιάσαμε μία αρχιτεκτονική που παρέχει L2 connectivity μεταξύ κόμβων που ανήκουν στο ίδιο συγκεκριμένο testbed, το VINI Trellis. Ωστόσο, υπάρχουν πολλές network testbed πλατφόρμες για ανάπτυξη και έλεγχο νέων δικτυακών τεχνολογιών. Κάθε πλατφόρμα έχει το δικό της πλαίσιο ελέγχου που διαχειρίζεται τοπικά. Ως αποτέλεσμα, είναι δύσκολο να δημιουργηθούν testbeds παγκόσμιας κλίμακας. Την λύση σε αυτό το πρόβλημα μπορούν να δώσουν οι συνενώσεις (federation) πολλών testbed. Ένα σύνολο από federated testbeds μπορεί να παρέχει παγκόσμιας κλίμακας, ρεαλιστικό περιβάλλον διεξαγωγής πειραμάτων.

Η υποδομή για virtual links, και κατ'επέκταση ενός L2 broadcast domain, στα virtual network testbeds συνήθως βασίζεται σε μία από τις τεχνολογίες VLAN ή EoGRE. Για παράδειγμα, στην περίπτωση του VINI έχουμε L2 τοπολογίες που καθορίζονται και διαχωρίζονται βάσει IP end-points και GRE tunnels ενώ στην περίπτωση του FEDERICA τα L2 broadcast domains και οι L2 τοπολογίες καθορίζεται και διαχωρίζονται με VLANs. Η επίτευξη L2 connectivity σε ένα federation από testbeds όπου όλα χρησιμοποιούν για την δημιουργία της υποδομής EoGRE είναι τετριμμένη. Σε τέτοια περίπτωση η δημιουργία του L2 broadcast domain που περιλαμβάνει κόμβους από διαφορετικά testbeds απαιτεί την ίδια διαδικασία που θα ακολουθούσαμε για την δημιουργία ενός L2 broadcast domain σε ένα από αυτά τα testbeds, όπως π.χ. παρουσιάσαμε για το VINI Trellis. Στην περίπτωση όμως του federation

από testbeds που χρησιμοποιούν είτε EoGRE είτε VLAN για την δημιουργία των virtual links τότε ο τρόπος δημιουργίας ενός L2 broadcast domain μεταξύ κόμβων των διάφορων testbeds δεν είναι προφανής.

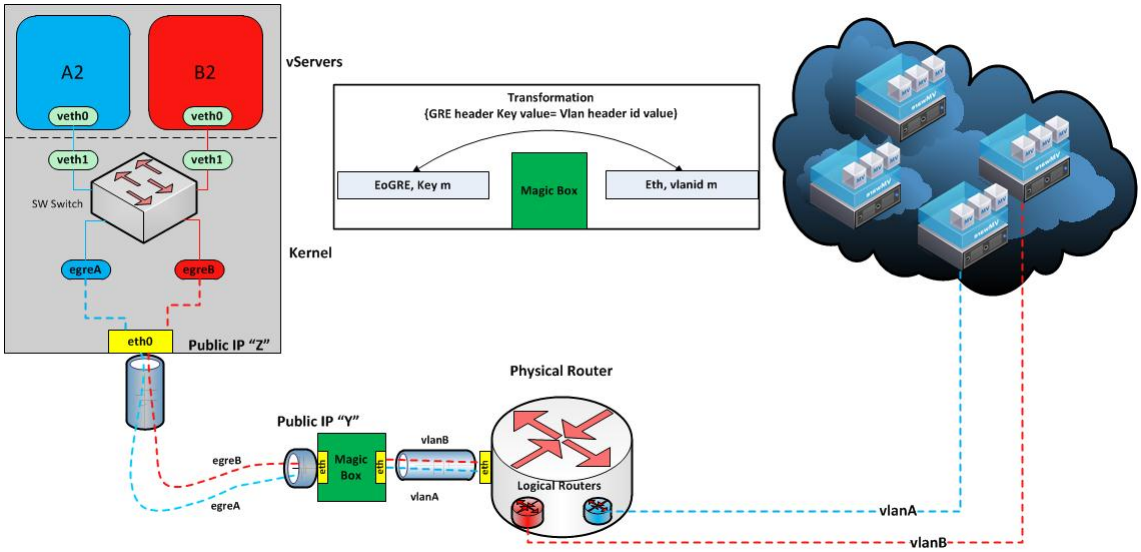
4.2.1 Αρχιτεκτονική

Σε μία προσπάθεια να προσαρμοστούμε στο κλίμα των εξελίξεων θα κάνουμε μία επιπλέον προσθήκη στην αρχιτεκτονική που ήδη παρουσιάσαμε. Η προσθήκη αποσκοπεί στην δυνατότητα της παροχής L2 connectivity ενός sliver σε VINI Trellis nodes με sliver σε κάποιο άλλο testbed που στηρίζεται στην τεχνολογίας VLAN για να δημιουργήσει L2 broadcast domains. Το αποτέλεσμα θα είναι ένα L2 stiching των τεχνολογιών EoGRE και VLAN.

Στην περίπτωση της εκτεταμένης αρχιτεκτονικής του VINI Trellis που δημιουργήσαμε, το σύνολο του L2 broadcast domain ενός slice μπορεί να προσεγγιστεί μέσω του node όπου εντοπίζεται το switch sliver. Για την επίτευξη, λοιπόν, του L2 connectivity μεταξύ των κόμβων του VINI και των κόμβων του co-federated testbed, προυποτίθεται η ύπαρξη L2 connectivity μεταξύ των κόμβων του co-federated και το node όπου είναι τοποθετημένο το switch sliver. Προσπαθώντας να καλύψουμε αυτή την προυπόθεση συντάμε τα εξής προβλήματα:

- το co-federated testbed δεν υποστηρίζει EoGRE, άρα δεν μπορούμε να συνδέσουμε απευθείας τα nodes μέσω διαδικτύο κάνοντας χρήση των public IPs τους,
- τα nodes μπορεί να μην δέχονται public IPs (π.χ. FEDERICA) και
- δεν ειναι αναγκαίο οι φυσικές υποδομές των testbeds να βρίσκονται στην ίδια φυσική τοποθεσία (co-located), άρα δεν μπορούμε απλά να δημιουργήσουμε μια φυσική L2 τοπολογία που να παρέχει την απαραίτητη συνδεσιμότητα.

Απαντάμε σε αυτά τα προβλήματα ορίζοντας ένα **Magic Node**. Το Magic Node είναι ένα φυσικό υπολογιστικό σύστημα co-located με το co-federated testbed που αναλαμβάνει να παρέχει την υπηρεσία του L2 stitching των federated testbeds. Ο Network Administrator μπορεί να θεωρήσει το Magic Node σαν ένα μαύρο κουτί που δέχεται κίνηση EoGRE και την μετατρέπει σε VLAN.



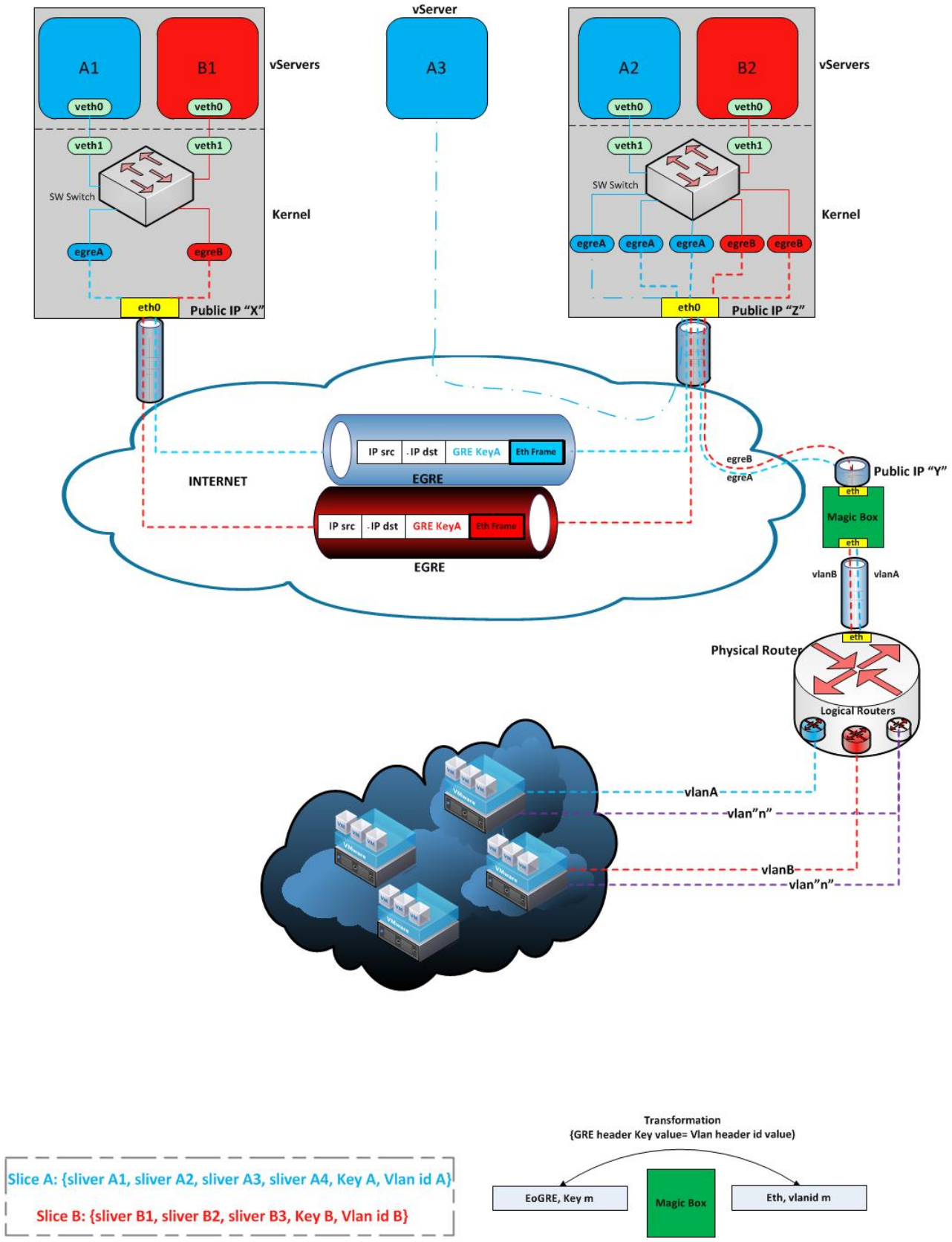
Σχήμα 4.2: Μετάβαση της κίνησης από GRE σε VLAN

4.2.1.1 Περιγραφή του Magic Node

Τον ρόλο Magic Node μπορεί να αποτελέσει οποιοδήποτε υπολογιστικό σύστημα co-located στο co-federated testbed site. Δέχεται EoGRE πλαίσια από την πλευρά του VINI Trellis testbed. Αφού εξάγει το payload Ethernet frame το προωθεί σε ένα logical swith, που έχει δυνατότητα VLAN tagging στα ports, όπου και παράγεται ένα 802.1Q πλαίσιο με το ορισμένο VLAN tag. Τέλος, το VLAN tagged πλαίσιο προωθείται προς τα nodes του co-federated testbed. Ως logical switch την επιλέγουμε να εκτελείται το Open vSwitch, καθώς αποτελεί ιδανικό εργαλείο για την προσωμοίωση ενός switch σε επίπεδο software. Το εργαλείο Linux bridge δεν αποτελεί εναλλακτικη, καθώς, ενώ προσφέρει παραπάνω λειτουργικότητα από ένα απλό bridge η προσωμοίωση ενός logical switch με υποστήριξη για VLAN ports ξεπερνάει τις δυνατότητες του.

Στο βασικό σενάριο χρήσης ενός Magic Node , καταλήγει σε ένα port ενός physical switch στο οποίο καταλήγει και η on-site φυσική υποδομή του co-federated testbed. Αναθέτοντας λοιπόν τα δύο ports σε ένα logical switch έχουμε την ολοκλήρωση του L2 stitching. Ωστόσο, μπορούμε πλέον να βασιστούμε στο L2 connectivity μεταξύ slices των federated testbeds και να το χρησιμοποιήσουμε για να παρέχουμε υπηρεσίες για οποιοδήποτε δικτυακό επίπεδο ανώτερο του L2. Έτσι, σε ένα εξελιγμένο σενάριο χρήσης του Magic Node, συνδέεται σε ένα physical router, όπου συνδέεται και η φυσική υποδομή του co-federated testbed. Στον physical router, η inter-slice κίνηση περνάει μέσα από logical routers. Κάθε logical router λειτουργεί πάνω από ένα L2 broadcast domain. Αυτό το εξελιγμένο σενάριο περιγράφεται στο σχήμα 4.2.

Η συνολική αρχιτεκτονική λοιπόν είναι:



Σχήμα 4.3: Δικτυακή υποδομή σε federated περιβάλλον

Κεφάλαιο 5

Υλοποίηση

Αρχικά θα παρουσιάσουμε το βασικό σενάριο χρήσης του Trellis και πως χρησιμοποιεί τα εργαλεία Vsys και τα IIAS scripts. Στην συνέχεια θα εξηγήσουμε τις προσθήκες μας.

Υπάρχουν τέσσερις βασικοί μέθοδοι δημιουργίας εικονικών δικτύων στο Trellis. Η επιλογή τους γίνεται ορίζοντας την τιμή του attribute "vini_topo" στο slice. Οι δυνατές τιμές είναι οι εξής:

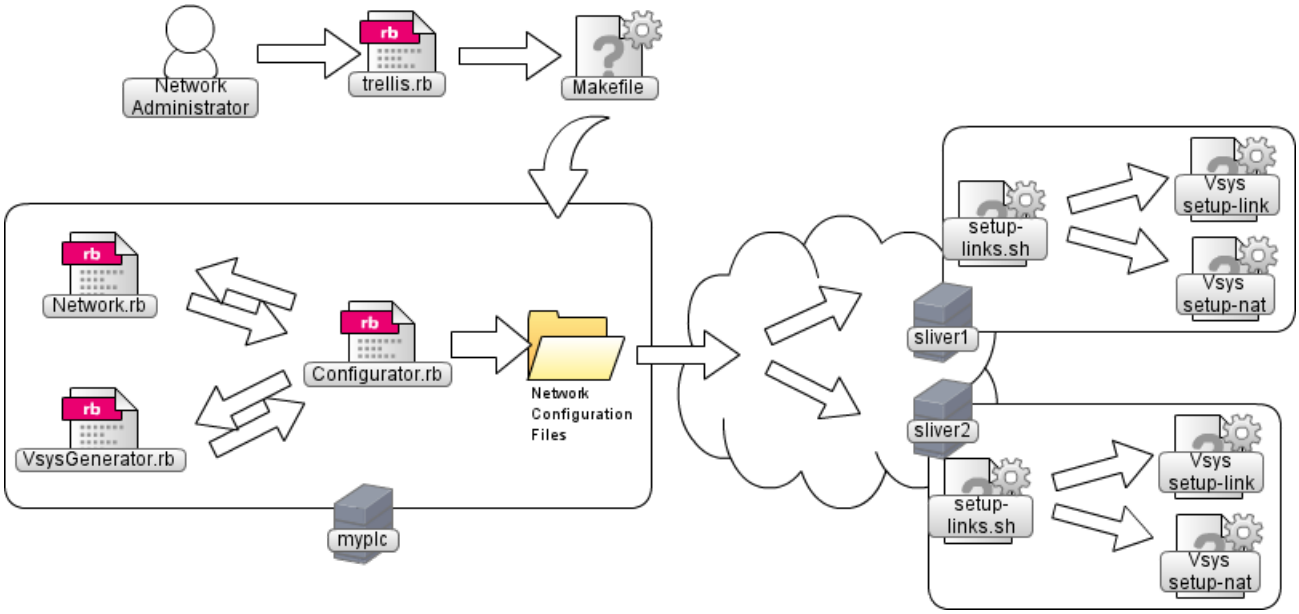
- **none**: Η μέθοδος **none** δεν επιτρέπει την δημιουργία virtual networks σε αυτό το slice. Όλα τα slices που χρησιμοποιούν αυτή την μέθοδο χρησιμοποιούν το φυσικό NIC του κόμβου, όπως στο κλασσικό Planetlab.
- **vsys**: Η μέθοδος **vsys** επιτρέπει την δημιουργία εικονικών τοπολογιών στο slice με την χρήση των IIAS scripts.
- **iias**: Η μέθοδος **iias** δημιουργεί και ρυθμίζει αυτόματα τα virtual links μεταξύ των κόμβων σε ένα slice, εφ'όσον οι κόμβοι ανήκουν σε sites άμεσα συνδεδεμένα στην φυσική τοπολογία.
- **manual**: Η μέθοδος **manual** επιτρέπει στον administrator του MyPLC να ορίσει τα slice attributes "topo_rspec".

Η καλύτερη μέθοδος για πειραματισμό και επέκταση των εικονικών τοπολογίων του Trellis παρέχεται από την μέθοδο **vsys**. Τα IIAS scripts περιέχουν τους μηχανισμούς που έχουν χρησιμοποιηθεί από το Trellis για την υλοποίηση και των υπολοιπων μεθόδων. Έτσι, μπορούμε να πειραματιστούμε με τις υπάρχουσες δυνατότητες αλλά και να τις επεκτείνουμε, με αντίκρυσμα όχι μόνο στην μέθοδο **vsys**, αλλά στο σύνολο των παρεχόμενων μεθόδων του Trellis.

5.1 Υπάρχουσα διαδικάσια δημιουργίας δικτύου υποδομής

Με χρήση των εργαλείων IIAS scripts και Vsys μπορούμε, σαν διαχειριστές της δικτυακής υποδομής να δημιουργήσουμε την εικονική τοπολογία που θα είναι διαθέσιμη στους χρήστες, αφού εξηγήσουμε συνοπτικά πως λειτουργεί η διαδικασία και συνδυάζονται τα εργαλεία.

Οι δημιουργοί του Trellis παρέχουν ένα σύνολο από scripts τα οποία είναι υπεύθυνα για την δημιουργία της IIAS αρχιτεκτονικής. Αυτά τα scripts χρησιμοποιούνται από τον Network administrator, και τοποθετούνται στο MyPLC VM. Ο Network administrator προσδιορίζει σε ένα configuration file τις επιθυμητές από τους users τοπολογίες και τα scripts. Με αυτό τον τρόπο δημιουργούνται bash scripts για κάθε sliver κάθε node, που αντιγράφονται στα slivers. Αυτά τα scripts εκτελούνται σε κάθε sliver και καλούν κάποια προκαθορισμένα Vsys scripts, η εκτέλεση των οποίων δημιουργεί την ζητούμενη τοπολογία. Τα βασικά scripts για την δημιουργία των εικονικών τοπολογιών, καθώς και η κύρια διαδικασία φαίνονται στο σχήμα 5.1.



Σχήμα 5.1: Back-end λειτουργίες για την δημιουργία IIAS δικτυακής τοπολογίας στο Trellis

Χωριζούμε τα αρχεία της τοπολογίας σε δύο βασικές κατηγορίες το front-end και το back-end. Το front-end αποτελείται από τα αρχεία τα οποία χρησιμοποιεί άμεσα ο Network Administrator για να ρυθμίσει και να δημιουργήσει την εικονική τοπολογία. Το back-end αναλαμβάνει να δεχτεί την τοπολογία που έχει ορίσει ο Network Administrator και να την δημιουργήσει.

Back-end:

IIAS `Network.rb`: Το αρχείο `Network.rb` είναι το πρώτο που μελετάμε γιατί ορίζει τις βασικές κλάσεις που χρησιμοποιούνται στην δημιουργία της τοπολογίας. Οι βασικές κλάσεις είναι : `Slice`, `Iface`, `Node` και `Link`. Ο στόχος του script είναι να δημιουργήσει αντικείμενα (objects) των παραπάνω κλάσεων που περιέχουν το σύνολο των πληροφοριών την ορισμένης από τον Network Administrator τοπολογία. Αποτελεί το πρώτο βήμα της διαδικασίας του back-end για την δημιουργία των εικονικών τοπολογιών.

- `Slice`: Στιγμιότυπα της κλάσης `Slice` περιέχουν τις βασικές πληροφορίες του slice όπως το όνομα του, το κλειδί για τα GRE tunnels (gre key), το ζητούμενο περιβάλλον της πλατφόρμας (PL-VINI ή Trellis), καθώς και ρυθμίσεις που αφορούν το routing σε L3.
- `Iface`: Η κλάση `Iface` χρησιμοποιείται αποκλειστικά από το back-end και περιγράφει

ένα network interface που εγκαθίσταται μέσα σε sliver και αποτελεί end-point του virtual link.

- `Node`: Η κλάση `Node` διατηρεί πληροφορίες για τον κόμβο που φιλοξενεί ένα sliver αλλά και για το ίδιο το sliver. Περιέχει τα δικτυακά χαρακτηριστικά του κόμβου (π.χ. dns name, IP), αλλά και χαρακτηριστικά του sliver όπως αν θα εκτελεί λειτουργίες routing, αν θα εκτελεί OpenVPN[1] server κ.τ.λ.

- `Link`: Ο σκοπός της κλάσης `Link` είναι να περιγράψει ένα virtual link και να δημιουργήσει τα κατάλληλα στιγμιότυπα της κλάσης `Iface` που θα αποτελούν τα end-points του virtual link. Για κάθε virtual link στο οποίο συμμετέχει το sliver δημιουργείται ένα νέο Iface και προσδιορόζεται η IP του.

IIAS `Configurator.rb`: Το αρχείο `Configurator.rb` αποτελεί τον συνδετικό κρίκο όλων των αρχείων. Αρχικοποιεί όλες τις κλάσεις από τα απαραίτητα αρχεία και καλεί τις μεθόδους που δημιουργούν τα scripts που θα εκτελεστούν στα slivers για την κατασκευή της τοπολογίας. Αποτελεί τον σύνδεσμο του back-end με το front-end. Η σύνδεση γίνεται με την μετάφραση του configuration file trellis.rb σε στιγμιότυπα κλάσεων του Network.rb. Στην συνέχεια, καλούνται οι μέθοδοι που θα παράγουν την εικονική τοπολογία με βάση αυτά τα στιγμιότυπα. Η βασικότερη από αυτές τις μεθόδους, όσον αφορά την τοπολογία, είναι η VsysGenerator που περιέχεται στο ομώνυμο αρχείο.

IIAS `VsysGenerator.rb`: Το αρχείο `VsysGenerator.rb` δημιουργεί τα scripts που παράγουν την τοπολογία. Τυπώνει, δηλαδή, τα bash scripts που θα εκτελεστούνε στα slivers. Παράγει διαφορετικά scripts για το κάθε sliver. Τα bash scipts που παράγονται χρησιμοποιούν τα Vsys scripts setup-link και setup-nat προκειμένου να εγκαταστήσουν τα ζητούμενα εικονικά links και να προσδώσουν network connectivity στα slivers. Δημιουργούνται τέσσερα bash scripts για κάθε sliver:

- `setup-links`: Όταν εκτελεστεί μέσα σε ένα sliver δημιουργεί και ρυθμίζει όλα τα απαραίτητα logical και virtual network devices που απαιτούνται για τα virtual links στα οποία συμμετέχει το sliver καθώς και για την παροχή του network connectivity στο sliver. Απαραίτητα εργαλεία αποτελούν τα Vsys scripts setup-link και setup-nat.

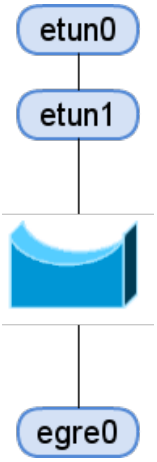
- `teardown-links`: Καταστρέφει όλα τα network devices που δημιουργήθηκαν από το script `setup-links`.

- `ping-test`: Δοκιμάζει την λειτουργία του network connectivity του sliver, καθώς και την ορθή λειτουργία των virtual links.

- `startup`: Ενεργοποιεί και ρυθμίζει υπηρεσίες που πιθανώς απαιτούνται από το sliver, οπως τον Quagga software router.

Vsys `setup-link`: Το αρχείο `setup-link` είναι υπεύθυνο για την δημιουργία της τοπολογίας στο εκάστοτε sliver. Δημιουργεί όλη την απαραίτητη ομάδα από interfaces, στο sliver και στον host, που θα αποτελέσει το ένα άκρο του virtual link. Η ομάδα αποτελείται από 4 linux network interfaces συνδεδεμένα μεταξύ τους σε σειρά, όπως φαίνεται στό σχήμα 5.2.

[1] `http://openvpn.net/`



Σχήμα 5.2: Τα interfaces που δημιουργεί το script setup-link

1. **etun0**: Το ένα interface του ενός Virtual Ethernet ζευγαριού (veth pair). Ένα veth είναι αποτελείται από δύο εικονικά Ethernet devices συνδεδεμένα μεταξύ τους έτσι ώστε όταν ένα πλαίσιο εισέρχεται στο ένα device τότε εξέρχεται από το άλλο. Η συνήθης χρήση των veth είναι για την σύνδεση ενός guest με κάποιο network interface του host. Κάθε συσκευή έχει την δικία της διεύθυνση MAC και εφόσον συνδεθεί με κάποιο ενεργό interface όπως κάποιο ethX μπορεί να δώσει στον guest πλήρη δικτυακή λειτουργικότητα. Το etun0 είναι το άκρο του veth pair που εγκαθίσταται στο container (guest).

2. **etun1**: Το άκρο του veth pair από την πλευρά του host.

3. **br0**: Το **br0** είναι ένα linux bridge που χρησιμποιείται για να συνδέσει το **etun1** με το **egre0**. Μέσω του **br0** καταφέρνει το veth pair να αποκτήσει πρόσβαση σε πακέτα που προέρχονται από το δίκτυο και καταλήγουν στο egre0.

4. **egre0**: Το **egre0** είναι το end-point ενός EoGRE tunnel. Στην άλλη άκρη του tunnel υπάρχει το node που περιέχει το δεύτερο sliver του virtual link. Υλοποιείται ως linux logical network interface που είναι συνδεδεμένο με το physical interface του node.

Το script `setup-link` υιοθετεί μία "χαζή" συμπεριφορά, δημιουργώντας μία ομάδα από interfaces για κάθε ξεχωριστό virtual link ενός sliver, αναντίστοιχα με τις προδιαγραφές του Trellis.

Vsys `setup-nat`: Το αρχείο `setup-nat` καθιερώνει το network connectivity του slice. Ουσιαστικά, δημιουργεί ένα κανούργιο network interface στο sliver και το συνδέει με τον host κρυβωντάς το από NAT. Τα interfaces που δημιουργούνται είναι ένα Virtual Ethernet pair, τα nat$gre_key και natx$gre_key, όπου $gre_key το GRE key του slice.

- **nat$gre_key**: Το interface **nat$gre_key** αποτελεί το άκρο του veth pair που εγκαθίσταται μέσα στο sliver. Μπορεί να χρησιμοποιηθεί για σύνδεση με το διαδίκτυο, ή οποιοδήποτε δίκτυο στο οποίο είναι συνδεδεμένος ο node. Ακόμα, σε περίπτωση που το slice τρέχει OpenVPN server χρησιμοποιείται το interface **nat$gre_key** ώστε να εισάγει κίνηση ο χρήστης μέσω του διαδίκτυου στο virtual network που του παρέχεται.

- **natx$gre_key**: Το interface **natx$gre_key** αποτελεί το άκρο του veth pair που εγκαθίσταται στο root context του node. Μέσω του εργαλείου Linux ipables[2] δημιουργείται ένα Network Address Translation (NAT) μεταξύ του **natx$gre_key** και του physical network device του host. Έτσι υλοποιείται η σύνδεση μεταξύ τυο sliver και του φυσικού δικτύου στο οποίο βρίσκεται το node.

Front-end:

IIAS `trellis.rb`: Το αρχείο `trellis.rb` είναι το configuration file, όπου ο Network Administrator μπορεί να δηλώσει την επιθυμητή τοπολογία. Ουσιαστικά, αρχικοποιούνται κλάσεις που περιέχονται στο `Network.rb`. Ο χρήστης ορίζει αρχικά σε ποιά slices (από τα υπάρχοντα στο site) θέλει να δημιουργήσει τις εικονικές τοπολογίες. Στην συνέχεια, ορίζονται τα χαρακτηριστικά των κόμβων που εγκαθίστανται τα slivers του slice και τα virtual links που θα δημιουργηθούν μεταξύ των slivers.

IIAS `Makefile`: Το Makefile αποτελεί ουσιαστικά το το εργαλείο του Network Administrator για την δημιουργία της τοπολογίας. Παρέχει ένα σύνολο από εντολές που αντιστοιχούν σε ενέργειες του back-end. Το Makefile είναι υπέυθυνο για την κλήση των back-end scripts, την μεταφορά των παραγόμενων scripts στα slices καθώς και για την εκτέλεση τους.

5.1.1 Δημιουργία της virtual switched τοπολογίας

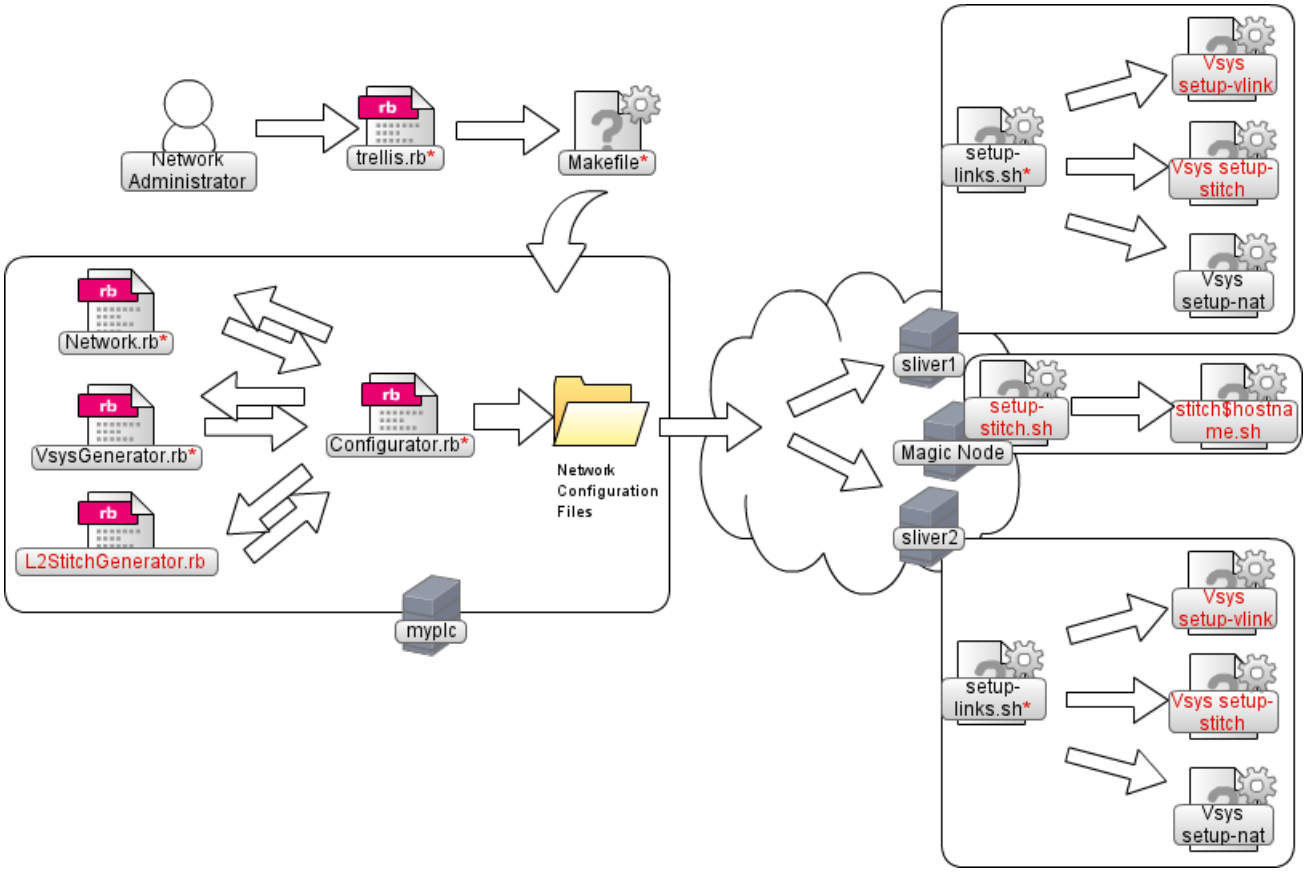
Η αρχιτεκτονική IIAS βασίζεται στην δημιουργία virtual point-to-point links μεταξύ των slivers. Εμείς στοχεύουμε στην επέκταση της λειτουργικότητας της αρχιτεκτονικής IIAS, με την προσπάθεια να δημιουργήσουμε ένα Virtual Private Lan Service βασισμένο στο IIAS. Αυτή η νέα αρχιτεκτονική, βασίζεται όπως αναφέρθηκε στην χρήση ενός sliver ως switch sliver, που θα παρέχει συνδεσιμότητα μεταξύ όλων των υπολοίπων slivers. Το switch sliver επίσης θα είναι υπέθυνο για το L2Stitching. Ουσιαστικά επεκτείνουμε τα εργαλεία Vsys και IIAS δίνοντας στο network administrator του των αφιερωμένων VINI nodes που εκτελούν το Trellis την δυνατότητα για αυτόματη εγκατάσταση και εκτέλεση του Open vSwitch καθώς και αυτόματη δημιουργία του της switched τοπολογίας, με βάση τις προδιαγραφές που έχει ζητήσει ο χρήστης.Ακόμα, προσθέτουμε την δυνατότητα αυτόματου L2 stitching με την χρήση Magic Node.

Back-end:

IIAS `Network.rb`: Το αρχείο `trellis.rb` διατηρεί τον ρόλο του αλλά με αυξημένη λειτουργικότητα. Το αρχείο `Network.rb` Τροποιούμε τις υπάρχουσες κλάσεις εκτός της κλάσης `Iface` και ορίζουμε την νέα κλάση `Magic_Node` όπως περιγράφεται παρακάτω.

- `Slice`: Η νέα κλάση `Slice` περιέχει επιπλέον στοιχεία για το αν το slice χρησιμοποιεί Open vSwitch αντί για Linux Bridge και αν έχει ενεργοποιημένο το L2 stitching.
- `Node`: Στην νέα κλάση `Node`, τα nodes που περιέχουν switch slivers δημιουργούν ένα μοναδικό veth pair για κάθε switch sliver. Επίσης δημιουργούνται virtual links μεταξύ όλων των slivers και το switch sliver, εάν αυτό υπάρχει και ανήκει στον συγκεκριμένο node.

[2] `http://www.netfilter.org/`



Σχήμα 5.3: Back-end λειτουργίες για την δημιουργία της νέας δικτυακής τοπολογίας στο Trellis

- `Link`: Τροποποείται η λογική ανάθεσης των IP διευθύνσεων καθώς πλέον δεν έχουμε μόνο virtual point-to-point links αλλά ολόκληρα L2 broadcast domains.
- `Magic_Node`: Η κλάση `Magic_Node` διατηρεί τα απαιραίτητα στοιχεία για έναν Magic Node προκείμενου να είναι δυνατή η σύνδεση με αυτόν σε φυσικό επίπεδο. Επίσης περιέχει πληροφορία για το VLAN network interface που θα χρησιμοποιηθεί για το L2 stitching.

IIAS `Configurator.rb`: Το αρχείο `Configurator.rb` διατηρεί τον ρόλο του αλλά με αυξημένη λειτουργικότητα. Η βασική τροποποίηση είναι ο έλεγχος αν έχουν γίνει σωστά οι δηλώσεις για το Open vSwitch και το L2 Stitching, και η κλήση στην συνέχεια των τροποποιημένων μεθόδων του `VsysGenerator.rb` και των νέων μεθόδων του `L2StitchGenerator.rb`.

IIAS `VsysGenerator.rb`: Το αρχείο `VsysGenerator.rb` διατηρεί τον ρόλο του αλλά με αυξημένη λειτουργικότητα. Συγκεκριμένα, τροποιούνται τα script `setup-links` και `teardown-links` που παράγονται. Οι αλλαγές παρουσιάζονται στην συνέχεια.

- `setup-links`: Δίνεται η δυνατότητα να εκτελεστεί ένα από τα `setup-link` `setup-vlink` Vsys scripts. Στην περίπτωση που ο Netowrk Administrator έχει επιλέξει το virtual network του slice να στηρίζεται σε Open vSwitch τότε καλείται το `setup-vlink` Vsys script, αλλιώς το `setup-link`. Επιπλέον, εφ'όσον έχει οριστεί το sliver

ως switch sliver καλείται στο `setup-stitch` Vsys script προκειμένου να δημιουργήσει το configuration του stitching από την πλευρά του sliver.

- `teardown-links`: Καταστρέφει όλα τα network devices που δημιουργήθηκαν από το script `setup-links`, από όποιο Vsys script και αν έχουν δημουργηθεί. Προστέθηκαν δηλαδή οι εντολές αναστροφής των λειτουργιών του `setup-stich` Vsys script.

IIAS `L2StitchGenerator.rb`: Το αρχείο `L2StitchGenerator.rb` ρυθμίζει τα Magic Nodes ώστε να συνδέονται με τα αντίστοιχα switch slivers και να υλοποιούν το L2 stitching, δηλαδή να μετατρέπουν την εισερχόμενη EoGRE κίνηση σε VLAN κίνηση. Έχει την λογική του `VsysGenerator`, δηλαδή παράγει bash scripts, που στην συνέχεια αντιγράφονται και εκτελούνται στα Magic Nodes. Πιο συγκεκριμένα παράγει τα παρακάτω scritps.

- `stitch$node`: Ρυθμίζει το L2 stitching στο Magic Node για λογαρισμό του sliver που εγκαθίσταται στον node με όνομα $node. Δημιουργεί ένα καινούργιο EoGRE interface, στο vswitch π ου εκτελείται στο Magic Node, με gre key το gre key του slice. Το EoGRE interface συνδέεται πάνω σε μιά access Vlan port του vswitch με Vlan tag το gre key του slice. Το άλλο end-point του EoGRE tunnel έχει δημιουργηθεί με την εκτέλεση του αρχείου `setup-vlink` στο switch slice. Ακόμα προσθέτει το gre key ως trunk tag στο trunk port που είναι συνδεδεμένο με το physical switch.
- `unstitch$node`: Όταν εκτελεστεί καταστρέφει τα interfaces που δημιουργήθηκαν και αναστρέφει τις ρυθμίσεις που έγινα στα υπάρχοντα interfaces από το αντίστοιχο `stitch$node` script.

Vsys `setup-vlink`: Το αρχείο `setup-vlink` αντικαθιστά το αρχείο `setup-link`. Ο σκοπός τους είναι ο ίδιος, η δημιουργία της τοπολογίας στο sliver. Τα interfaces **veth0** και **veth1** παραμένουν ίδια αλλά το **br0** και το **egre0** τροποποιούνται όπως εξηγείται στην συνέχεια.

- **br0**: Το Linux bridge instance έχει αντικατασταθεί από ένα Open vSwitch instance. Το ίδο instance χρησιμοποιείται από όλα τα slivers του node. Το isolation της κίνησης μεταξύ των slivers επιτυγχάνεται ορίζοντας τα ports του br0 που αφορούν το ίδιο slice ως access VLAN ports με vlan tag το gre key.
- **egre0**: Αντί για χρήση του Linux virtual interface για EoGRE χρησιμοποιούμε τα native Open vSwitch EoGRE interfaces και τα συνδέουμε, όπως και στο `setup-link`, με το physical interface του node.

Vsys `setup-stitch`: Το αρχείο `setup-stitch` χρησιμοποιείται με τον ίδιο τρόπο όπως και το αρχείο `setup-vlink` για διαφορετική, όμως, λειτουργία. Ρυθμίζει το L2 stitching από την πλευρά του sliver. Το επιτυγχάνει δημιουργώντας ένα κανούργιο EoGRE Open vSwitch interface προορισμένο να επικοινωνεί με τον αντίστοιχο Magic Node. Επιπλέον συνδέει το EoGRE interface με ένα Vlan acces port του vswitch, που βρίσκεται στο root context, με VLAN tag το gre key του slice ώστε να επικοινωνεί με τα υπόλοιπα ports του slice που υπάρχουν στο συγκεκριμένο0 vswitch instance.

Vsys `ovs-setup`: Το αρχείο `ovs-setup` εκτελείται σε root-context και δεν είναι διαθέσιμο στους users των slices. Χρησιμοποιείται για την αυτόματη εγκατάσταση του Open vSwitch. Ελέγχει αν υπάρχει ήδη εγκατεστημένο το Open vSwitch και σε αυτή την περίπτωση τερματίζει

χωρίς αλλαγές. Αν δεν υπάρχει εγκατάσταση, κατεβάζει το source του Open vSwitch-1.1.1 και στην συνέχεια το κάνει build.

Vsys `ovs-start`: Το αρχείο `ovs-start` εκτελείται σε root-context και δεν είναι διαθέσιμο στους users των slices. Χρησιμοποιείται για να εκκινήσει την βάση δεδομένων και τον daemon του Open vSwitch. Για να δημιουργηθούν vswitch instances θα πρέπει να έχει εγκατασταθεί (`ovs-setup`) και εκκινηθεί (`ovs-start`) το Open vSwitch. Κατα την εκτέλεσή του `ovs-start` αφαιρούνται δύο modules από το λειτουργικό.

- **bridge module**: Είναι υπεύθυνο για τις λειτουργίες του Linux bridge. Αφαιρείται για να εγκατασταθεί σε kernel space το Open vSwitch ώστε να επιτυγχάνονται οι μέγιστες δυνατές επιδόσεις.
- **ip_gre module**: Είναι υπεύθυνο για την υλοποίηση των virtual network devices που υλοποιούν τα IPoGRE και EoGRE. Αφαιρείται για να χρησιμοποιηθεί η native EoGRE υλοποίηση του Open vSwitch για βελτίωση της απόδοσης αλλά και του ελέγχου πάνω στο EoGRE virtual network device.

Η λειτουργίες τους αντικαθίστανται από το openvswitch_mod.

Vsys `ovs-stop`: Το αρχείο `ovs-stop` εκτελείται σε root-context και δεν είναι διαθέσιμο στους users των slices. Σταματάει την λειτουργιία της βάσης και του daemon του Open vSwitch και επαναφέρει τα Linux module που αφαιρέθηκαν από το `ovs-start`.

Front-end:

IIAS `trellis.rb`: Το αρχείο `trellis.rb` διατηρεί τον ρόλο του αλλά με αυξημένη λειτουργικότητα. Ο Network Administrator μπορεί να ορίσει ορίζει αρχικά σε ποιά slices θα εκτελούν θα χρησιμοποιούν Open vSwitch και χρειάζονται υπηρεσίες L2Stitching. Στην συνέχεια, αν έχει επιλεγεί η χρήση Open vSwitch πρέπει να οριστεί το sliver ποιού κόμβου θα αποτελεί το switch sliver. Ο ορισμός ενός switch slice προκαλεί την αυτόματη δημιουργία virtual links του switch sliver με όλα τα υπόλοιπα slivers που ανήκουν στο ίδιο slice. Στο παράρτημα Α΄.1 επισυνάπτεται ένα ολοκληρωμένο παράδειγμα της μορφής του `trellis.rb`.

IIAS `Makefile`: Το αρχείο `Makefile` επίσης διατηρεί τον ρόλο του αλλά με αυξημένη λειτουργικότητα. Έχουν προστεθεί εντολές για την εγκατάσταση, εκκίνηση και τερματισμό του Open vSwitch. Ακόμα, έχουν προστεθεί εντολές για τον συγχρονισμό των αρχείων με το Magic Node και την εκκίνηση και τον τερματισμό του L2Stitching. Στο παράρτημα Α΄ επισυνάπτεται μία περιγραφή των εντολών που δέχεται το Makefile για την δημιουργία της εικονικής τοπολογίας.

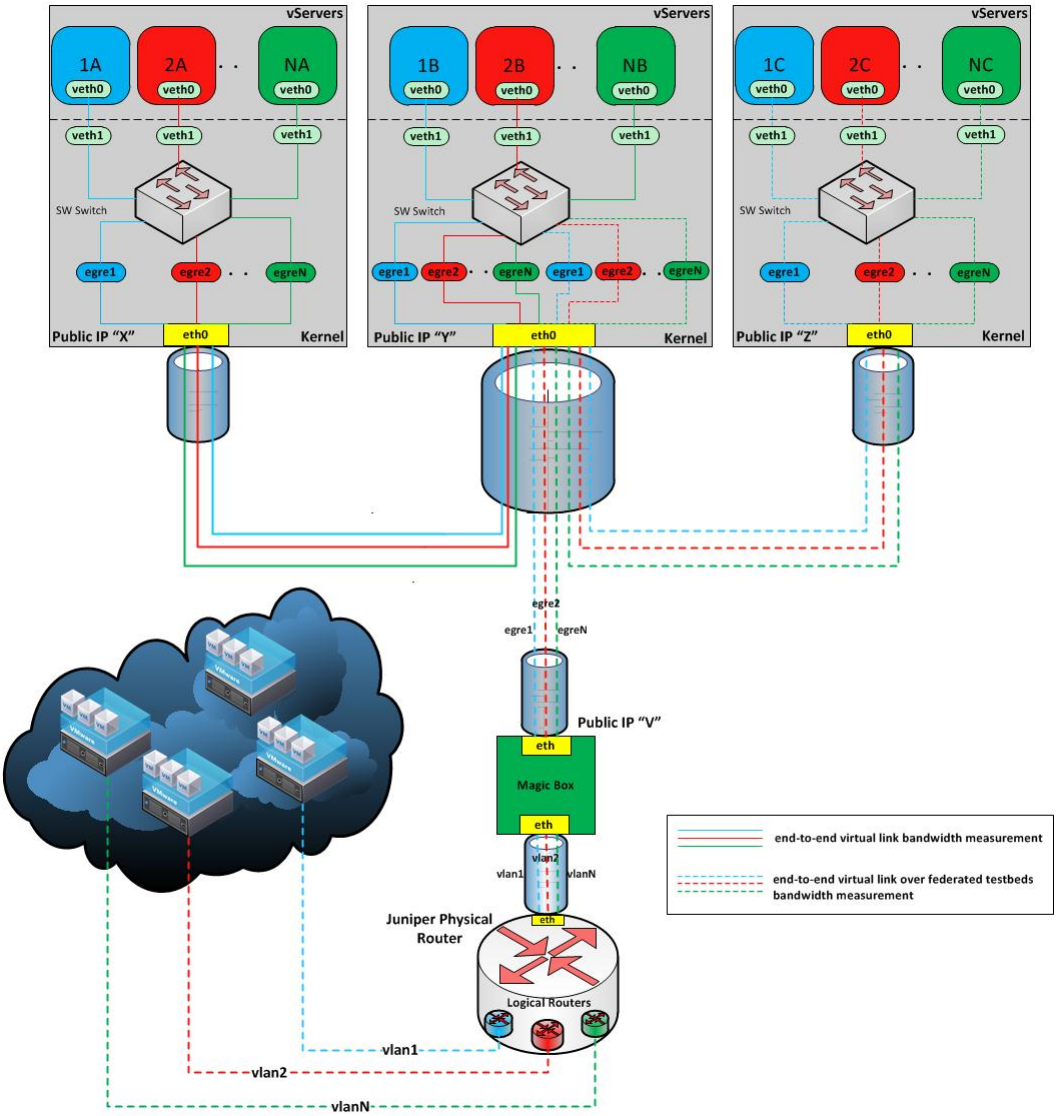
5.2 Εκτίμηση της Απόδοσης του Συστήματος

Με σκοπό να αποδείξουμε την λειτουργία των εργαλείων που δημιουργήσαμε μια σειρά από proof-of-concept L2 τοπολογίες στις οποίες ελέγξαμε την συνδεσιμότητα μεταξύ των slivers.

Η τοπολογία όπου κάναμε τον έλεγχο των εργαλείων είναι αυτή που παρουσιάζεται στο σχήμα 5.4. Έχουμε N, μεταβλητό αριθμό slices και ελέγχουμε δύο διαφορετικά σενάρια για τα διάφορα N.

1. Συνδεσιμότητα των slivers με χρήση point-to-point virtual links που χρησιμοποιούνε Open vSwitch.

2. Χρήση του Magic Node για συνδεσιμότητα μεταξύ slivers σε διαφορετικά testbeds που συνδέονται μέσα από logical routers του Juniper MX-80 router.



Σχήμα 5.4: Τα 2 σενάρια των πειραμάτων

Στην συνέχεια παρουσιάζουμε κάποια συμπεράσματα από τον πειραματισμό μας με την πλατφόρμα:

- Η εικονική τοπολογία που δημιουργήσαμε λειτουργεί ορθώς χωρίς απώλειες πακέτων ακόμα και σε ρυθμούς μετάδοσης που ξεπερνάνε τα 200 Mbps. Η ορθή λειτουργία αφορά τόσο τις intra-testbed εικονικές ζεύξεις όσο και το L2 stitching.

- Ενώ επικυρώθηκε η ορθή λειτουργία της πλατφόρμας, δεν μπορούμε να προσδιορίσουμε με απόλυτη ακρίβεια τη διαφορά των επιδόσεων μεταξύ της τυπικής Trellis αρχιτεκτονικής σε σύγκριση με την αρχιτεκτονική που περιγράψαμε στο Κεφάλαιο 4. Αυτό οφείλεται

κυρίως στις διαθέσιμες υποδομές των μετρήσεων. Τα phsysical nodes των proof-of-concept τοπολογιών υλοποιήθηκαν από εικονικά μηχανήματα με σκοπό την εξοικονόμηση φυσικών πόρων. Η επιλογή αυτή περιέπλεξε την διεξαγωγή μετρήσεων καθώς προσέθεσε ένα επιπλέον επίπεδο εικονικοποίησης. Το overhead του επιπλέον επιπέδου εικονικοποίησης φαίνεται να μεταβάλλεται δυναμικά και δεν μπορεί να μοντελοποιηθεί με ικανοποιητικό τρόπο. Γι'αυτό προτείνουμε κάθε μελλοντική πειραματική τοπολογία να δημιουργείται σε φυσικά μηχανήματα.

- Παρ΄ολα αυτά η τοπολογία που έκανε χρήση του Open vSwitch φάνηκε να έχει καλύτερες επιδόσεις από την αντίστοιχη τοπολογία με Linux Bridge. Μάλιστα το bandwidth των virtual links με χρήση Open vSwitch είχε τιμές αρκετά κοντά σε εκείνες τον αντίστοιχων physical links. Αυτός είναι ένας από τους βασικούς σκοπούς της πλατφόρμας του Trellis και με τις προσθήκες μας την φέραμε ένα βήμα πιο κοντά.

Κεφάλαιο 6

Επίλογος

6.1 Συμπεράσματα

Παρουσιάσαμε μία εικονική πλατφόρμα διεξαγωγής δικτυακών πειραμάτων βασισμένη στο Trellis και το Open vSwitch που παρέχει δύο βασικές λειτουργίες. Πρώτον, παρέχει την δυνατότητα στο χρήστη να πειραματιστεί χρησιμοποιώντας σαν βάση περίπλοκες εικονικές L2 τοπολογίες και όχι απλώς εικονικά point-to-point links. Δεύτερον, η πλατφόρμα αυτή έχει την δυνατότητα δημιουργίας L2 τοπολογιών που συνδέουν τους πόρους ενός χρήστη που ανήκουν σε διαφορετικά federated testbeds. Στην συνέχεια υλοποιήσαμε ένα proof-of-concept εργαλείο που δημιουργεί L2 broadcast domains στο Trellis και παρέχει L2 connectivity μεταξύ κόμβων από federated testbeds. Ελέγξαμε την συνδεσιμότητα για να αποδείξουμε ότι η αρχικτεκτονική λειτουργεί ορθά και συμπεράναμε, πειρματικά, ότι αυξάνεται το bandwidth των virtual links σε σχέση με το Trellis. Παρ'όλα αυτά προτείνουμε να ανατπυχθεί ένα εργαλείο που θα επιτρέπει το αυτόματο verification τοπολογιών με σκοπό να μελετηθεί σε περισσότερο βάθος η σταθερότητα και η απόδοση του συστήματος.

6.2 Προτάσεις για μελλοντική εργασία

Εκτός από την βαθύτερη μελέτη επίδοσης του συστήματος που προτείναμε, υπάρχουν μια σειρά από δυνατές επεκτάσεις στην πλατφόρμα που παρουσιάσαμε που θα την καταστήσουν πιο ολοκληρωμένη και θα αυξήσουν την λειτουργικότητα της. Η πιο ενδιαφέρουσα επέκταση είναι η σύνδεση των Open vSwitch instances με OpenFlow controllers, κάνοντας πιο δυναμικό τον τρόπο ελέγχου των εικονικών τοπολογιών και παρέχοντας περισσότερες δυνατότητες πειραματισμού στον χρήστη. Στην συνέχεια υπάρχουν και πιο τεχνικές επεκτάσεις. Για παράδειγμα μπορούμε να επεκτείνουμε την πλατφόρμα για να δώσουμε στον χρήστη δυνατότητα πειραματισμού με την τεχνολογία VLAN. Αυτό μπορεί να γίνει είτε με η επίτευξη του islotion της κίνησης του κάθε slice μέσα σε ένα Open vSwitch instance χωρίς την χρήση VLAN, είτε με υλοποίηση του 802.1QinQ για το Open vSwitch. Τέλος πρέπει να υλοποιηθούν μιά σειρά από εργαλεία διαχείρισης και ελέγχου των vswitches για του χρήστες ώστε να μπορεί ο χρήστης να εκμεταλλέυεται στο μέγιστο την λειτουργικότητα του εικονικού μεταγωγέα.

Παράρτημα Α΄

Οδηγός εγκατάστασης Τοπολογίας Υποδομής

Α΄.1 Ρύθμιση του trellis.rb

Παρουσιάζουμε ένα παράδειγμα του configuration file trellis.rb για την δημιουργία δικτυακής υποδομής σε δύο VINI Trellis Slices.

Listing Α΄.1: src/trellis.rb

```
1  ################## Slices  ##################
2
3  # Slice to be applied tha virtual topology
4  # must be an existant PLC slice
5  ### Slice 1
6  $iias_1 = Slice.new('pl_vini_slice1')
7  $iias_1.set_environment('Trellis')
8  $iias_1.set_key(101);
9  # Enable usage of OVS for this slice instead of Linux Bridge
10 $iias_1.set_ovs(true)
11 # Enable l2 stitching for this slice by the Trellis side
12 $iias_1.set_l2stitch(true)
13
14 # Slice to be applied tha virtual topology
15 # must be an existant PLC slice
16 ### Slice 2
17 $iias_2 = Slice.new('pl_vini_slice2')
18 $iias_2.set_environment('Trellis')
19 $iias_2.set_key(102);
20 # Enable usage of OVS for this slice instead of Linux Bridge
21 $iias_2.set_ovs(true)
22 # Enable l2 stitching for this slice by the Trellis side
23 $iias_2.set_l2stitch(true)
24
25 ##########################   Nodes   ######################
26
27 # Nodes belonging to the first slice and are going to be used
28 # must be an existant PLC nodes
29 $vini1_1 = Node.new('vini1.netmode.ntua.gr', $iias_1, 'nmode1_1')
```

```
30 $vini2_1 = Node.new('vini2.netmode.ntua.gr', $iias_1, 'nmode2_1')
31 $vini3_1 = Node.new('vini3.netmode.ntua.gr', $iias_1, 'nmode3_1')
32 # Enable switch slice in node vini2_1
33 # Automatically creates virtual links between
34 # this slice and every other slice
35 $vini2_1.switch(true)
36
37 # Give Magic Node info
38 # usage MagicNode.new(slice,hostname,IP,VLANiface)
39 $vinim_1 = MagicNode.new($iias1,"yoda.netmode.ntua.gr","147.102.13.57","eth2")
40
41 # Nodes belonging to the second slice and are going to be used
42 # must be an existant PLC nodes and belong to this slice
43 $vini1_2 = Node.new('vini1.netmode.ntua.gr', $iias_2, 'nmode1_2')
44 $vini2_2 = Node.new('vini2.netmode.ntua.gr', $iias_2, 'nmode2_2')
45 $vini3_2 = Node.new('vini3.netmode.ntua.gr', $iias_2, 'nmode3_2')
46 # Enable switch slice in node vini2_2
47 # Automatically creates virtual links between
48 # this slice and every other slice
49 $vini2_2.switch(true)
50
51 # Give Magic Node info
52 # usage: MagicNode.new(slice,hostname,IP,VLANiface)
53 $vinim_2 = MagicNode.new($iias2,"yoda.netmode.ntua.gr","147.102.13.57","eth2")
54
55 ########################   Links   ######################
56
57 ### Link 1
58 # Create a virtual link between vini1_1 and vini3_1 with loss rate 10
59 $link1_1 = Link.new($vini1_1, $vini3_1, 10)
60
61 ### Link 2
62 # Create a virtual link between vini1_2 and vini3_2 with loss rate 10
63 $link1_2 = Link.new($vini1_2, $vini3_2, 10)
64
65 ########################   hostinfo   ######################
66
67 $vini1_1.hostinfo("147.102.13.156")
68 $vini2_1.hostinfo("147.102.13.157")
69 $vini3_1.hostinfo("147.102.13.168")
70
71 $vini1_2.hostinfo("147.102.13.156")
72 $vini2_2.hostinfo("147.102.13.157")
73 $vini3_2.hostinfo("147.102.13.168")
```

Α΄.2 Εγκατάσταση Τοπολογίας

Όλα τα αρχεία που περιγράψαμε παραπάνω αποτελούν το backend του εργαλείου το PI για την δημιουργία της ζητούμενης από τους users εικονικής τοπολογίας. Το frontend, που χρειάζεται να γνωρίζει ο PI αποτελείται από ένα Makefile με πολλές επιλογές που εξηγούνται στην συνέχεια.

`make configs`: Δημιουργεί τα network configuration scripts στον κόμβο που βρίσκονται τα IIAS scripts, σύμφωνα με το configuration που έχει περιγραφεί στο trellis.rb.

`make install_ovs`: Εγκαθιστά το Open vSwitch στο root-context όλων των κόμβων που ανφέρονται στο trellis.rb. Περισσότερες πληροφορίες παρέχονται στο αντίστοιχο backend file ovs-setup του Vsys (§5.1.1).

`make start_ovs`: Εκκινεί το Open nSwitch στο root-context όλων των κόμβων που ανφέρονται στο trellis.rb. Περισσότερες πληροφορίες παρέχονται στο αντίστοιχο backend file ovs-start του Vsys (§5.1.1).

`make stop_ovs`: Τερματίζει την λειτουργία του Open nSwitch στο root-context όλων των κόμβων που ανφέρονται στο trellis.rb. Περισσότερες πληροφορίες παρέχονται στο αντίστοιχο backend file ovs-stop του Vsys (§5.1.1).

`make sync`: Αντιγράφει τα network configuration scripts στα προσδιορισμένα slices.

`make sync_magic`: Αντιγράφει τα stitching configuration scripts στους magic nodes.

`make topo`: Συνδέεται στα slices και εκτελεί τα netwrok configuration scripts που δημιουργούν τα virtual networks.

`make test`: Δοκιμάζει την συνδεσιμότητα των virtual networks που έχουν δημιουργηθεί στα προγούμενα βήματα.

`make stitch`: Συνδέεται στους magic nodes και εκτελεί τα stitching configuration scripts ολοκληρώνοντας το L2 stitching.

`make teardown`: Συνδέεται στα slices και καταστρέφει τα virtual networks που έχουν δημιουργηθεί στα προηγούμενα βήματα.

`make unstitch`: Συνδέεται στα magic nodes και αναστρέφει τις ρυθμίσεις που έχουν γίνει για το L2 stitching.

`make clean_nodes`: Διαγράφει από τα slices τα υπάρχοντα network configuration scripts.

`make clean`: Διαγράφει από τον κόμβο που βρίσκονται τα IIAS scripts τα υπάρχοντα network configuration files.

Παράρτημα Β΄

Open vSwitch

Β΄.1 Open vSwitch Tables

Bridge Table Columns

Column	Type
Core Features	
name	string: Ένα μοναδικό όνομα που αντιστοιχεί στο bridge αυτό
ports	set of Ports: ports included on the bridge
flood_vlans	int[0-4095]*: VLAN IDs των VLAN που θέλουμε να αντικαταστήσουμε το MAC learning με flooding
Other Features	
datapath_type	string: Το όνομα του παροχέα του datapath. Το kernel datapath έχει τύπο system, ενώ το userspace datapath έχει τύπο netdev

Port Table Columns

Column	Type
name	string: Ένα μοναδικό όνομα που αντιστοιχεί στο port αυτό
interfaces	Ένα ή περισσότερα(σε περίπτωση Bonding) interfaces
Vlan Configuration	
tag	int[0-4095]: Προαιρετικός ακέραιος που μετατρέπει το port σε acces port για το VLAN με το δεδομένο VLAN ID.
trunks	int[0-4095]*: Προεραιτικοί ακέραιοι που μετατρέπουν το port σε trunk port για τα VLANs που αντιστοιχούν στα δεδομένα VLAN IDs
Other Features	
qos	Προαιρετικός δείκτης σε πίνακα QoS που περιέχει τις ρυθμίσεις
fake-bridge	boolean: εκονικό sub-bridge για συγκεκριμένο VLAN

Interface Table Columns

Column	Type
Core Features	
name	string: Ένα μοναδικό όνομα που αντιστοιχεί στο interface αυτό
mac	optional string: διεύθυνση Ethernet για το interface
System-Specific Details	
type	string: Δυνατές τιμές: system(Linux network device),internal, tap(TUN/TAP), gre(EoGREoIPv4: RFC 2890), ipsec_gre(EoGREoIPv4 IPSec: RFCs 2401,2890), capwap(RFC 5415), patch(ζευγάρι από εικονικά devices), null
options	map of string-string pairs: Ρυθμίσεις που αφορούν την εγγραφή type
Interface status	
admin_state	optional sting : up or down
link_state	optional string : either down or up
link_speed	optional integer
duplex	optional string: full or half
mtu	optional integer
status	map of string-string pairs
Ingress Policing	
ingress_policing_rate	integer, at least 0: Μεγιστός ρυθμός για εισερχόμενα δεδομένα σε αυτό το interface, σε kbps.
ingress_policing_burst	integer, at least 0: Μέγιστο burst size για εισεχρόμενα δεδομένα σε αυτό το interface, σε kb. Έχει νόημα για rate > 0. Πρέπεί να είναι τουλάχιστον ίσο το μέγεθος του MTU. Κάνει πιο ελαστικό τον αλγόριθμο με σκοπό την αύξηση του rate.

B′.2 Basic Open Vswitch API

Στο σενάριο χρήσης του Open vSwitch ως εικονικού μεταγωγέα παρέχονται οι εξής δυνατότητες:

- L2 switching και MAC learning
- υποστήριξη του πρωτοκόλλου 802.1Q για VLAN
- Port mirroring, με δυνατότητα VLAN tagging
- υποστήριξη για NetFlow v5[1]
- υποστήριξη για sFlow®[2] monitoring
- Σύνδεση σε εξωτερικό OpenFlow controller, όπως ο NOX[3]
- NIC bonding με αυτόματο fail-over και source MAC-based TX εξισορρόπηση φορτίου

Παρουσίαση του βασικού API του Open vSwitch. Δεν γίνονται αναφορές στην σύνδεση με το OpenFlow.

- **ovsdb-server**: Ο daemon που εκκινεί την βάση δεδομένων και απαντάει στις κλήσεις του ovs-vsctl. Διατηρεί όλες τις δομές που περιγράφονται ή παρουσιάζονται στο παράρτημα B′.1.
- **ovs-vswitchd**: Ο βασικός deamon του Open vSwitch, σε περίπτωση που χρησιμοποιείται σαν εικονικός μεταγωγέας και όχι OpenFlow switch. Διατηρεί την βασική παραμετροποίηση όλων των δημιουργημένων εικονικών μεταγωγέων. Κατά την εκκίνηση του λαμβάνει της πληροφορίες παραμετροποίησης από την βάση δεδομένων. Δημιουργεί τα απαραίτητα datapaths και στην συνέχεια εκτελεί το switching για κάθε μεταγωγέα όπως περιγράφει η ρύθμιση του. Διατηρείται ενημερωμένος σχετικά με τις αλλαγές στην βάση δεδομένων.
- **ovs-appctl**: Εργαλείο διαχείρησης των εκτελούμενων Open vSwitch daemons. Οι Open vSwitch daemons δέχονται ορισμένες εντολές κατά την διάρκεια εκτέλεσης που ελέγχουν την συμπεριφορά τους, είτε ζητούν τις ρυθμίσεις τους. Η εντολή ovs-appctl παρέχει έναν απλό τρόπο χρήσης αυτών των εντολών.
- **ovs-vsctl**: Πρόγραμμα που χρησιμοποιείται για την ρύθμιση του ovs-vswitchd παρέχωντας μία υψηλού επιπέδου διαπροσωπεία για την επικοινωνία με την βάση δεδομένων. Από προεπιλογή, συνδέεται σε έναν ovsdb-server που διαχειρίζεται την βάση παραμετροποίησης του Open vSwitch. Μέσω αυτής της σύνδεσης θέτει ερωτηματα είτε εφαρμόζει αλλαγές στην βάση δεδομένων, που καθορίζονται από τις εντολές που δέχεται.Στην συνέχεια, αν εφάρμοσε αλλαγές μέχρι να ολοκληρωθεί η επαναπαραμετροποίηση του ovs-vswitchd και ύστερα τερματίζει.
- **ovs-brcompatd**: Daemon που παρέχει συμβατότητα με το front-end των Linux Bridge εργαλείων στον ovs-vswitchd. Λειτουργεί αναμένοντας για εντολές τύπου bridge ioctl (π.χ. αυτές που παράγονται από το πρόγραμμα brctl) που αφορούν προσθήκη ή αφαίρεση datapaths και interfaces που είναι προσαρτημένα σε αυτά.

[1] http://www.cisco.com/go/netflow
[2] http://www.sflow.org/
[3] http://noxrepo.org/

- **ovs-dpctl**: Πρόγραμμα διαχείρησης των Open vSwitch datapaths. Μπορεί να δημιουργήσει, τροποποιήσει και διαγράψει Open vSwitch datapaths. Κάθε μηχάνημα μπορεί να φιλοξενήσει μέχρι 250 datapaths. Δηλαδή, στην περίπτωση χρήσης του εικονικού μεταγωγέα μπορούν να δημιουργηθούν το μέγιστο 250 εικονικοί μεταγωγείς. Η βασική του χρηστικότητα είναι εμφανής αν χρησιμοποιηθεί το Open vSwitch σαν OpenFlow switch. Παρ'όλα αυτά, παρέχει χρήσιμα στατιστικά για την λειτουργία του εικονικού μεταγωγέα.
- **ovs-ofctl**: Πρόγραμμα για διαχείρηση του flow-based forwarding καθώς και τον έλεγχο και την διαχείριση των διαθέσιμων OpenFlow switches. Μέσω του εργαλείου **ovs-ofctl** είναι δυνατή η πρόσβαση στην κατάσταση ενός OpenFlow switch καθώς και η παραμετροποίηση του. Επιπλέον, μπορούμε να χειριστούμε ένα vswitch intsance σαν OpenFlow switch και να θέσουμε flow-based κανόνες.

Παράρτημα Γ'

Source code

Listing Γ'.1: src/iias/Network.rb

```
1  # This file implements in a way the backend of the IIAS tools.
2  # It is consisted by the definition of the main classes that
3  # are used. From more general to more  specialized:
4  # Slice, Iface, Node, Link
5
6  $Slices = Array.new
7  $Nodes = Array.new
8  $Links = Array.new
9  $Network = "192.168.100"
10 # Start manos
11 $IP = "2"
12 $MagicNode = nil
13 # End manos
14
15 # Slice: set attributes
16 # Start manos: Added ovs attribute
17 class Slice
18
19   attr_reader :environment, :slice, :router, :click_port, :udp_port, :
         click_forwarding, :gre_key, :ovs, :l2stitch
20 # Stop manos
21
22   def initilaize(cp,up,slice,cf,router)
23     @click_port = cp
24     @udp_port = up
25     @slice = slice
26     @click_forwarding = cf
27     @router = router
28     @environment = 'PL-VINI'
29 # Start manos
30     @ovs = false
31 # Stop manos
32
33     $Slices << self
34   end
35
36   def initialize(slice)
```

```
37      @click_port = nil
38      @udp_port = nil
39      @slice = slice
40      @click_forwarding = nil
41      @environment = 'PL-VINI'
42      @router = 'none'
43      @gre_key = 0;
44  # Start manos
45      @ovs = false
46      @l2stitch = false
47  # Stop manos
48
49      $Slices << self
50    end
51
52    def to_s
53      return @slice
54    end
55
56    def set_environment(env)
57      @environment = env
58    end
59
60    def set_router(rtr, cf = false)
61      @router = rtr
62      @click_forwarding = cf
63    end
64
65    def set_ports(cp, up)
66      @click_port = cp
67      @udp_port = up
68    end
69
70    def set_key(key)
71      @gre_key = key
72    end
73
74  # Start manos
75  # config setter for OVS
76    def set_ovs(val)
77      @ovs = val
78      print "Slice #{@slice} has enabled OpenSwitch! \n"
79    end
80
81  # config setter for L2stich
82    def set_l2stitch(val)
83      @l2stitch = val
84      print "Slice #{@slice} has enable L2stitching \n"
85    end
86  # Stop manos
87
88  end
89
90  # Iface : set attributes
91  class Iface
92    attr_reader :link, :node, :name, :label, :ipaddr, :macaddr, :neighbor
93    def initialize(link, node, name, ipaddr, macaddr = nil)
```

```
94      @link = link
95      print "Network.rb : Iface : #{node.dnsname} #{name} -> #{ipaddr} \n"
96      @ipaddr = ipaddr
97
98      case node.slice.environment
99      when 'Trellis'
100 # THe name is natD* or ethD* so we keep just the number
101       index = name.gsub(/\D/,"")
102
103       @name = "a#{node.slice.gre_key}if#{index}"
104       @label = index
105     else
106       @name = name
107       @label = name
108     end
109
110     @macaddr = macaddr ? macaddr : ip_to_uml_mac(@ipaddr)
111     @node = node
112     @neighbor = nil
113   end
114
115   def ip_to_uml_mac(ip)
116     mac = "fe:fd"
117     @addr = ip.split('.')
118     @addr.each_index { |i|
119       mac += sprintf ":%02x", @addr[i]
120     }
121     return mac
122   end
123
124   def add_neighbor(nbr)
125     @neighbor = nbr
126   end
127
128   def get_ipaddr
129     return @ipaddr
130   end
131
132   def to_s
133     "#{@name}: #{@ipaddr} / #{@macaddr}"
134   end
135 end
136
137
138 # Node : set attributes, set NAT, set OVS
139 # Start manos : added run_switch
140 # TODO make switch like router : slice.switch
141 class Node
142   attr_reader :dnsname, :label, :router, :run_ospf, :run_ibgp,
143       :ibgp_route_reflector, :tap0, :fea, :iface, :click_forwarding,
144       :nat_dests, :nat_gateway, :realip, :run_openvpn, :slice,
145       :clientnet, :run_switch
146 # End manos
147
148   @@labels = Hash.new
149
150   def initialize(dnsname, slice, label=nil)
```

```
151|     @dnsname = dnsname
152|
153|     if label
154|       @label = label
155|     else
156|       # This works for the PlanetLab I2 nodes at least
157|       @label = @dnsname.split('.')[1]
158|     end
159|
160|     # Check for duplicate labels, which cause problems in Click config
161|     if @@labels[@label]
162|       print "ERROR: Nodes have same label '#{@label}'\n"
163|       print "       #{@@labels[@label]}\n"
164|       print "       #{dnsname}\n"
165|       print "Please see PL-VINI User Guide for help in specifying unique Node
           labels\n"
166|       print "Aborting....\n\n"
167|       exit 1
168|     else
169|       @@labels[@label] = dnsname
170|     end
171|
172|     @slice = slice
173|     @router = slice.router
174|     @click_forwarding = slice.click_forwarding
175|
176|     @run_ospf = $run_ospf
177|     @run_ibgp = $run_ibgp
178|     @run_openvpn = false
179|     @ibgp_route_reflector = nil
180| # Start manos
181|     @run_switch = false
182| # End manos
183|     @tap0 = nil
184|     @fea = nil
185|     @realip = nil
186|
187|     @iface = Array.new
188|     @nextiface = 0
189|
190|     @nat_dests = Array.new
191|     @nat_gateway = nil
192|
193|     @clientnet = nil
194|
195|     $Nodes << self
196|   end
197|
198|   def to_s
199|     ospf = @run_ospf ? "OSPF" : "no OSPF"
200|     ibgp = @run_ibpg ? "iBGP" : "no iBGP"
201|     string = "#{@dnsname}: #{@router}, #{ospf}, #{ibgp}"
202|     if (@tap0)
203|       string += "\n  #{@tap0.to_s}"
204|     end
205|     if (@fea)
206|       string += "\n  #{@fea.to_s}"
```

```
207     end
208     @iface.each_index { |i|
209       string += "\n  #{@iface[i].to_s}"
210     }
211     string
212   end
213
214 # hostinfo: find nodes' hosts IPs and set nat ifaces
215   def hostinfo(realip, tapip=nil, tapmac=nil)
216     @realip = realip
217
218     if (tapip && tapmac)
219       # Eth0 and fea have .2 and .3 in tap0's subnet
220       @tap0 = Iface.new(nil, self, "tap0", tapip, tapmac)
221       @iface[0] = Iface.new(nil, self, "eth0", tapip.succ)
222       @fea = Iface.new(nil, self, "fea", tapip.succ.succ)
223       #print "Network.rb : Node : #{@dnsname} tap ip #{@tap0.ipaddr} and iface
              ip #{@iface[0].ipaddr} \n"
224 # Start manos
225       case @slice.ovs
226       when true
227         $Network = "192.168.#{slice.gre_key}"
228         natip = $Network + ".99"
229         nexthopip = $Network + ".1"
230       print "Network.rb : Node : #{@dnsname} -> NAT: #{natip} NextHop: #{
            nexthopip} \n"
231       else
232         natip = $Network + ".99"
233         nexthopip = $Network + ".1"
234         $Network = $Network.succ
235       print "Network.rb : Node : #{@dnsname} -> NAT: #{natip} NextHop: #{
            nexthopip} \n"
236       end
237  # End manos
238       @iface[1] =  Iface.new(nil, self, "eth1", natip)
239       @nat_gateway = Iface.new(nil, self, nil, nexthopip)
240
241       # OpenVPN clients run on tap0's /24.
242       @clientnet = Iface.new(nil, self, "OpenVPN", tapip.sub(/.1$/,'.0'))
243
244       @nextiface = 2
245     end
246   end
247
248   def create_iface(link, ipaddr)
249 # Start manos
250 # Create more than one ifaces in the switch node with same IP
251 # We suppose we use it only for switch purposes
252     if (@run_switch && @nextiface != 0)
253   ipaddr = @iface[0].get_ipaddr
254     end
255 # Stop manos
256     iface = Iface.new(link, self, "eth#{@nextiface}", ipaddr)
257     @iface[@nextiface] = iface
258     @nextiface += 1
259     @iface.each { |i| print i, "\n"}
260     return iface
```

```
261|   end
262|
263|   def add_nat_dests(ipnets)
264|     @nat_dests = @nat_dests | ipnets
265|   end
266|
267|   def openvpn(val)
268|     @run_openvpn = val
269|   end
270|
271|   def ibgp(val)
272|     @run_ibgp = val
273|   end
274|
275|   def route_reflector(node)
276|     @ibgp_route_reflector = node
277|   end
278|
279| # Start manos
280| # Setter for ovs
281|   def switch(val)
282|     if (@slice.ovs)
283|       @run_switch = val
284|       print " #{@dnsname} is going to act as a switch for #{@slice}! \n"
285|     else
286|       print "WARNING: Slice #{@slice} is not configured for OpenVSwitch\n"
287|       print "     Node #{dnsname} will not run OVS\n"
288|     end
289|   end
290| # End manos
291|
292| end
293|
294| # Link: define link ip's
295| class Link
296|   attr_reader :node0, :node1, :iface0, :iface1, :ospf_cost, :loss, :rate
297|
298|   OspfCost = nil
299|   Loss = 0
300|
301|   def initialize(node0, node1, ospf_cost=nil, loss=nil, rate=nil)
302|     @node0 = node0
303|     @node1 = node1
304|     @ospf_cost = ospf_cost ? ospf_cost : OspfCost
305|     @loss = loss ? loss : Loss
306|     @rate = rate
307|
308| # Start manos
309| #     $Network = "192.168.#{node0.slice.gre_key}."
310| #     ipaddr = $Network + $IP
311| #     ipaddr = "192.168.#{node0.slice.gre_key}.#{IP}"
312| #     $IP = ($IP.succ).succ
313|     case (node0.run_switch || node1.run_switch)
314|     when true
315|       $Network1 = "192.168.#{node0.slice.gre_key}"
316|       ipaddr = $Network1 + "." + $IP
317|       $IP = ($IP.succ).succ
```

```
318         print "Switch Link : #{node0.dnsname} + #{node1.dnsname} : #{$Network1} \n
                "
319       else
320         $Network = $Network.succ
321     # reject the GRE-key subnet
322     if $Network == "192.168.#{node0.slice.gre_key}"
323       $Network = $Network.succ
324           print "Normal Link (GRE conflict): #{node0.dnsname} +#{node1.dnsname} :
                #{$Network} \n"
325     end
326         print "Normal Link : #{node0.dnsname} +#{node1.dnsname} : #{$Network} \n"
327         ipaddr = $Network + ".2"
328       end
329 # End manos
330       @iface0 = @node0.create_iface(self, ipaddr)
331       @iface1 = @node1.create_iface(self, ipaddr.succ)
332       @iface0.add_neighbor(@iface1)
333       @iface1.add_neighbor(@iface0)
334
335       $Links << self
336     end
337
338     def to_s
339       cost = @ospf_cost ? @ospf_cost.to_s : "no"
340       loss = @loss ? (@loss*100).to_s : "no"
341       "#{@iface0.ipaddr} - #{@iface1.ipaddr}: #{cost} cost, #{loss}% loss"
342     end
343 end
344
345 class MagicNode
346     attr_reader :slice, :label, :realip, :vlan_iface
347
348     def initialize(slice, label, realip, vlan_iface)
349     @slice = slice
350     @label = label
351     @vlan_iface = vlan_iface
352     @realip = realip
353     print "Slice #{@slice.slice}: Magic Node #{@label}@#{@realip}, Using iface #{
          @vlan_iface} \n"
354     $MagicNode = self
355     end
356 end
```

Listing Γ'.2: src/iias/Configurator.rb

```
#!/usr/bin/ruby

$root = File.dirname(__FILE__)

$LOAD_PATH << ($root)

require 'socket'
require 'optparse'
require 'Network'
require 'XORPGenerator'
require 'ClickGenerator'
require 'UmlNetGenerator'
require 'StartupGenerator'
require 'QuaggaGenerator'
require 'OpenVPNGenerator'
require 'VsysGenerator'
# Start manos
require 'L2StitchGenerator'
# End manos


# Set up some reasonable defaults
$config = nil
$outdir = "#{$root}/../../vini"
$run_ospf = true
$run_ibgp = false
$prio_boost = false

$config = ARGV[0]
if (! $config)
  printf "ERROR: Must specify name of config file as argument\n"
  exit 1
end

require "#{$config}"

if (! $outdir)
  printf "ERROR: Must specify an output directory ($outdir)\n"
  exit 1
end

# Get tap0 information for the node, if not provided in config file
$Nodes.each { |n|
  if (! n.realip)
    realip = TCPSocket.gethostbyname(n.dnsname)[3]

    case n.slice.environment
    when 'PL-VINI'
      tapip = tapmac = nil
      IO.popen("ssh -o StrictHostKeyChecking=no -l #{n.slice} #{n.dnsname} '/
          sbin/ifconfig|grep tap -A 2'") { |f|
        if f.gets =~ /HWaddr\s+([0-9A-F:]+)/
          tapmac = $1
        end

```

```
55          if f.gets =~ /(10\..*?)\s/
56            tapip = $1
57          end
58        }
59        n.hostinfo(realip, tapip, tapmac)
60        print "#{n.dnsname}: hostinfo(\"#{realip}\", \"#{tapip}\", \"#{tapmac}\")\
              n"
61      when 'Trellis'
62        n.hostinfo(realip)
63        print "#{n.dnsname}: hostinfo(\"#{realip}\")\n"
64
65      end
66    end
67  }
68
69  # Debugging stuff
70  if ($debug)
71    print "*** Nodes ***\n"
72    $Nodes.each { |n|
73      print "#{n.to_s}\n"
74    }
75
76    print "\n*** Links ***\n"
77    $Links.each { |l|
78      print "#{l.to_s}\n"
79    }
80  end
81
82  system("mkdir -p #{$outdir}")
83  # For use from makefile to connect to the nodes
84  f = File.new("#{$outdir}/node_list", "w+")
85  # Start manos
86  # For use from my vsys to update the scripts
87  f1 = File.new("#{$outdir}/../openvswitch/root_node_list", "w+")
88  # End manos
89
90  xorpgen = XORPGenerator.new
91  quaggagen = QuaggaGenerator.new
92  clickgen = ClickGenerator.new
93  umlgen = UmlNetGenerator.new
94  startupgen = StartupGenerator.new
95  openvpngen = OpenVPNGenerator.new
96  vsysgen = VsysGenerator.new
97  l2gen = L2StitchGenerator.new
98
99  # For each node, call appropriate Generators
100 print "Generating configuration files and scripts:\n"
101 $Nodes.each { |n|
102
103   print "  #{n.slice}@#{n.dnsname}: "
104
105   # Generate router configs
106   case n.router
107   when 'XORP'
108     xorpgen.printConfig(n)
109     print "XORP "
110   when 'Quagga'
```

```
111     quaggagen.printConfig(n)
112     print "Quagga "
113   end
114
115   case n.slice.environment
116   when 'PL-VINI'
117     # Generate topology in Click
118     clickgen.printConfig(n)
119     print "Click "
120
121     # Generate UML network interfaces
122     umlgen.printConfig(n)
123     print "UML "
124
125     # Generate UML network interfaces
126     startupgen.printConfig(n)
127     print "start\n"
128
129     umlgen.printDNSNames
130     openvpngen.printConfig
131
132   when 'Trellis'
133     vsysgen.printConfig(n)
134     vsysgen.printDNSNames
135 #     print "vsys\n"
136     if (n.slice.ovs && n.run_switch && n.slice.l2stitch)
137       if !(n.slice.ovs)
138     print "No openvswitch => no L2Stitch"
139   elsif !(n.run_switch)
140     print "If the node is a switch has a meaning to L2Stitch"
141   elsif !($MagicNode)
142         print "You haven't defined a Magic node\n"
143       else
144         m = $MagicNode
145     vsysgen.addStitch(n,m)
146         l2gen.printScript(n,m)
147
148       end
149     end
150   end
151
152   f.print "#{n.slice}@#{n.dnsname}\n"
153 # Start manos
154   f1.print "root@#{n.dnsname}\n"
155 # End manos
156 }
157
158 f.close
159 # Start manos
160 f1.close
161 # End manos
```

Listing Γ′.3: src/iias/VsysGenerator.rb

```
#!/usr/bin/ruby

require 'time'

class VsysGenerator

  def printConfig(node)
    printSetupLinks(node)
    printTeardownLinks(node)
    printPingTest(node)
    printStartup(node)
    addStitch(snode,mnode)
  end

  def printSetupLinks(node)
# Start manos
    if (node.slice).ovs == true
        print "Using setup-vlink! \n"
  script = "setup-vlink"
    else
        script = "setup-link"
    end
# End manos
    dir = "#{$outdir}/#{node.slice}@#{node.dnsname}"
    system("mkdir -p #{dir}")
    outfile = "#{dir}/setup-links"
    f = open(outfile.to_s, "w+")
    key = "#{node.slice.gre_key}"

    ctime = Time.new

    f.print <<EOF
# Trellis script for #{node.slice}@#{node.dnsname}
# automatically generated at #{ctime}
#!/bin/sh
set -e

# Bring up NAT interface,add default route
cat /vsys/setup-nat.out &
sleep 1
echo #{key} > /vsys/setup-nat.in
sleep 1
cat /vsys/local_grab-nat#{key}.out &
sleep 1
echo $$ > /vsys/local_grab-nat#{key}.in
sleep 1

/sbin/ifconfig nat#{key} 10.0.#{key}.2/24 up
/sbin/route add -net 10.0.#{key}.0/24 nat#{key}
/sbin/route add default gw 10.0.#{key}.1

EOF
    rmac = nil
    node.iface.each { |i|
      if (i.neighbor)
```

```
56          nbrnode = i.neighbor.node
57          nbr = nbrnode.label
58 # nbrnode.iface.each { |j|
59 #    if (j.neighbor.node == node)
60 #      rmac = j.macaddr
61 #    end
62 # }
63    lmac = i.macaddr
64    print "Configuring neighbor #{i.neighbor}\n"
65          f.print <<EOF
66 echo "Starting #{script}..."
67 cat /vsys/#{script}.out &
68 sleep 1
69 cat > /vsys/#{script}.in <<END
70 #{i.label}
71 #{nbrnode.realip}
72 #{key}
73 #{node.run_switch}
74 #{lmac}
75 END
76 # may rmac needed
77
78
79 sleep 1
80 echo "Ended #{script}!"
81
82 EOF
83        end
84      }
85
86      f.print <<EOF
87 # Bring up loopback
88 /sbin/ifconfig lo 127.0.0.1 up
89
90 EOF
91
92      node.iface.each { |i|
93        if (i.neighbor)
94      if !(node.run_switch && i.label != "0" )
95          # nbrnode = i.neighbor..node
96          # nbr = nbrnode.label
97 # Start manos changed [23] to [2-9]
98              net = i.ipaddr.sub(/.[2-9]+[0-9]*[0-9]*$/,'.0')
99              print "net: #{net} \n"
100 # End manos
101              f.print <<EOF
102 cat /vsys/local_grab-#{i.name}.out &
103 sleep 1
104 echo $$ > /vsys/local_grab-#{i.name}.in
105
106 sleep 1
107
108 /sbin/ifconfig #{i.name} #{i.ipaddr}/24 up
109 /sbin/route -v add -net #{net}/24 #{i.name}
110
111 EOF
112
```

```
113             if (i.link.rate)
114               f.print <<EOF
115 cat /vsys/local_rate-#{i.name}.out &
116 sleep 1
117 echo #{i.link.rate} >> /vsys/local_rate-#{i.name}.in
118
119 sleep 1
120
121 EOF
122             end
123   end
124       end
125     }
126 # Start manos
127 #    f.print "echo 2 > /proc/sys/net/ipv4/conf/all/arp_ignore"
128 # End manos
129
130     f.close
131   end
132
133   def printTeardownLinks(node)
134     dir = "#{$outdir}/#{node.slice}@#{node.dnsname}"
135     system("mkdir -p #{dir}")
136     outfile = "#{dir}/teardown-links"
137     f = open(outfile.to_s, "w+")
138
139     ctime = Time.new
140
141     f.print <<EOF
142 # Trellis script for #{node.slice}@#{node.dnsname}
143 # automatically generated at #{ctime}
144 #!/bin/sh
145 set -e
146
147 cat /vsys/local_delete-nat#{node.slice.gre_key}.out &
148 sleep 1
149 echo 1 > /vsys/local_delete-nat#{node.slice.gre_key}.in
150
151 EOF
152
153     node.iface.each { |i|
154       if (i.neighbor)
155 # if ! (node.run_switch and i.label != "0" )
156           f.print <<EOF
157 cat /vsys/local_delete-#{i.name}.out &
158 sleep 1
159 echo 1 > /vsys/local_delete-#{i.name}.in
160
161 EOF
162 # end
163       end
164     }
165
166     f.close
167   end
168
169   def printPingTest (node)
```

```
170|     dir = "#{$outdir}/#{node.slice}@#{node.dnsname}"
171|     system("mkdir -p #{dir}")
172|     outfile = "#{dir}/ping-test"
173|     f = open(outfile.to_s, "w+")
174|
175|     ctime = Time.new
176|
177|     f.print <<EOF
178| # Trellis script for #{node.slice}@#{node.dnsname}
179| # automatically generated at #{ctime}
180| #!/bin/sh
181|
182| check_errs()
183| {
184|   if [ "${1}" -ne "0" ]; then
185|     echo "TEST FAILED: ${2}" 1>&2
186|     exit ${1}
187|   fi
188| }
189|
190| # Test external connectivity
191| ping -c 3 vini-veritas.net
192| check_errs $? "Cannot ping via NAT"
193| echo ""
194|
195| EOF
196|
197|     node.iface.each { |i|
198|       if (i.neighbor)
199|   if ! (node.run_switch and i.label != "0" )
200|       f.print <<EOF
201| # Test link to #{i.neighbor.node.dnsname}
202| ping -c 3 #{i.neighbor.ipaddr}
203| check_errs $? "Cannot ping #{i.neighbor.ipaddr}"
204|
205|
206| EOF
207| #Stop manos
208|   end
209|       end
210|     }
211|
212|     f.close
213|   end
214|
215|   def printStartup(node)
216|     dir = "#{$outdir}/#{node.slice}@#{node.dnsname}"
217|     system("mkdir -p #{dir}")
218|     outfile = "#{dir}/startup"
219|     f = open(outfile.to_s, "w+")
220|
221|     ctime = Time.new
222|
223|     f.print <<EOF
224| # Trellis script for #{node.slice}@#{node.dnsname}
225| # automatically generated at #{ctime}
226| #!/bin/sh
```

```
227
228 EOF
229
230     case node.router
231     when "Quagga"
232       f.print <<EOF
233 cp ~/vini/#{node.slice}@#{node.dnsname}/quagga* /etc/quagga/
234 /usr/sbin/zebra -d -f /etc/quagga/quagga_zebra.cfg -i /var/run/quagga/zebra.pid
235 EOF
236
237       if node.run_ospf
238         f.print <<EOF
239 /usr/sbin/ospfd -d -f /etc/quagga/quagga_ospf.cfg -i /var/run/quagga/ospfd.pid
240 EOF
241       end
242
243       if node.run_ibgp
244         f.print <<EOF
245 /usr/sbin/bgpd -d -f /etc/quagga/quagga_bgp.cfg -i /var/run/quagga/bgpd.pid &
246 EOF
247       end
248
249     when "XORP"
250       f.print <<EOF
251 /usr/local/xorp/bin/xorp_rtrmgr -b /home/#{node.slice}/vini/#{node.slice}@#{node
      .dnsname}/xorp.cfg &
252 EOF
253     end
254
255       f.close
256     end
257
258     def printDNSNames
259
260       outfile = "#{$outdir}/hosts"
261       f = open(outfile.to_s, "w+")
262
263       ctime = Time.new
264       f.print <<EOF
265 # DNS mapping automatically generated at #{ctime}
266 127.0.0.1 localhost.localdomain localhost
267 EOF
268
269 # TODO kill duplicate lines
270       $Nodes.each { |n|
271
272         n.iface.each { |i|
273           if (i == n.iface[0])
274             f.print "#{i.ipaddr}     #{n.label}\n"
275           end
276           f.print "#{i.ipaddr}     if#{i.label}.#{n.label}\n"
277         }
278       }
279       f.close
280     end
281
282 # Start Manos
```

```
283|   def addStitch(snode,mnode)
284|     dir = "#{$outdir}/#{snode.slice}@#{snode.dnsname}"
285|     outfile = "#{dir}/setup-links"
286|     f = open(outfile.to_s, "a+")
287|     key = "#{snode.slice.gre_key}"
288|     f.print <<EOF
289| # Stitch
290| echo "Starting setup-stich"
291| cat /vsys/setup-stitch.out &
292| sleep 1
293| cat > /vsys/setup-stitch.in <<END
294| #{mnode.realip}
295| #{key}
296| END
297|
298| sleep 1
299| echo  "Finished setup-stitch"
300|
301| EOF
302|    f.close
303|
304|
305|     dir = "#{$outdir}/#{snode.slice}@#{snode.dnsname}"
306|     system("mkdir -p #{dir}")
307|     outfile = "#{dir}/teardown-links"
308|     f = open(outfile.to_s, "a+")
309|
310|     f.print <<EOF
311| #Unstitch
312| cat /vsys/local_delete-stitch#{key}.out &
313| sleep 1
314| echo 1  > /vsys/local_delete-stitch#{key}.in
315| EOF
316|
317|   f.close
318|   end
319|  # Stop Manos
320| end
```

Listing Γ′.4: src/iias/L2StitchGenerator.rb

```
#!/usr/bin/ruby


require 'time'

class L2StitchGenerator

 def printScript(snode,mnode)

    key = snode.slice.gre_key
  # Create the l2stitch script file if it doesn't exist
      dir = "#{$outdir}/../l2stitch/l2scripts"
      system("mkdir -p #{dir}")
    outfile = "#{dir}/stitch#{mnode.label}.sh"
  # The condition is used in order to create only
  # one script for every node
      if File.file?(outfile.to_s)
    f = open(outfile.to_s, "a+")
  else
    f = open(outfile.to_s, "w+")
      end
      ctime = Time.new

      f.print <<EOF
#!/bin/sh +x
# Magic Node stitch script for #{snode.dnsname}
# automatically generated at #{ctime}
OVS_CMD=`which ovs-vsctl`
KEY=#{snode.slice.gre_key}
BRIDGE=br0
EGRE=d${KEY}if0
VIFACE="#{mnode.vlan_iface}"
REMOTE="#{snode.realip}"


# Setup the bridge if it doesn't exist
$OVS_CMD br-exists $BRIDGE || $OVS_CMD add-br $BRIDGE
# Create the OVS EoGRE iface
$OVS_CMD -- add-port $BRIDGE $EGRE tag=$KEY -- set interface $EGRE type=gre
    options="remote_ip=$REMOTE,key=$KEY"
# Add the VLAN iface to OVS bridge
$OVS_CMD -- --may-exist add-port $BRIDGE $VIFACE -- add Port $VIFACE trunks $KEY
ifconfig $BRIDGE up

EOF

  f.close
  # Create the l2unstitch script file if it doesn't exist
      outfile = "#{dir}/unstitch#{mnode.label}.sh"
      if File.file?(outfile.to_s)
    f = open(outfile.to_s, "a+")
  else
    f = open(outfile.to_s, "w+")
      end
      f.print <<EOF
```

```
55|
56| #!/bin/sh +x
57| # Magic Node unstitch  script for #{snode.dnsname}
58| # automatically generated at #{ctime}
59| OVS_CMD=`which ovs-vsctl`
60| KEY=#{snode.slice.gre_key}
61| BRIDGE=br0
62| EGRE=d${KEY}if0
63| VIFACE="#{mnode.vlan_iface}"
64|
65| # Delete the EoGRE port
66| $OVS_CMD del-port  $BRIDGE $EGRE
67| $OVS_CMD remove PORT $VIFACE trunk $KEY
68|
69| EOF
70|   f.close
71|
72|   outfile = "#{dir}/../magic_nodes"
73|       if File.file?(outfile.to_s)
74|     f = open(outfile.to_s, "a")
75|   else
76|     f = open(outfile.to_s, "w")
77|       end
78|   f.print "root@#{mnode.label}\n"
79|   f.close
80|
81| end
82| end
```

Listing Γ'.5: src/vsys/ovs–setup

```
#!/bin/sh
# File: ovs-setup
# Author: Emmanouil Dimogerontakis
# Date: 18/07/2011
#
# Vsys script ran by Network Administrator to setup
# Open vSwitch in the root-context


# Handle OVS environment variable for use from ovs-start script
test -z `echo $OVS`
if [ `echo $?` != "0" ]
then
# If set use it
  OVS=`echo $OVS`
  echo "$OVS"
else
# If not set set it and add it to bashrc (for use after reboot)
  OVS="openvswitch-1.1.1";export OVS && echo "OVS=${OVS}" > /root/.bashrc &&
      echo "export OVS" >> /root/.bashrc && echo "OVS exported and added to
      baschrc"
fi

# Check  if the wanted installation is the current
ls /root/ | grep ${OVS}
if [ `echo $?` != "0" ]
then  # Not installed, or other vesrion
  rm -rf /root/openvswitch*
  cd /root

  yum -y install make gcc pkgconfig kernel-devel iproute tcpdump tar
  if [ `echo $?` != "0" ]
  then    # There was an error during install
    echo "There was an error during yum install" && exit 1;
  else    # The install was completed correctly
    # Download Open vSwitch src
    wget http://openvswitch.org/releases/${OVS}.tar.gz || exit 2;
    tar -zxvf ${OVS}.tar.gz
    cd ${OVS}/
    # Install ovs in kernel-space
    ./configure --with-l26=/lib/modules/`uname -r`/build && make && make install
        || exit 2
    # Insert ovs module
    insmod datapath/linux-2.6/openvswitch_mod.ko || (rmmod bridge && insmod
        datapath/linux-2.6/openvswitch_mod.ko && echo "WARN: bridge module
        unloaded")
    mkdir -p /usr/local/etc/openvswitch
    /usr/local/bin/ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd
        /vswitch.ovsschema
    echo "${OVS} succesfully installed!"
    exit 0
  fi
else  # The right version is installed
  echo "${OVS} already installed!"
  exit 0
```

```
50 fi
```

Listing Γ′.6: src/vsys/ovs–stop

```
#!/bin/sh
# File: ovs-stop
# Author: Emmanouil Dimogerontakis
# Date: 18/07/2011
#
# Vsys scritp ran by Network Administrator to stop ovs-execution
# in the root-context

SLICE=$1

# Add OVS paths if needed
echo $PATH | grep local || PATH=$PATH:/usr/local/bin:/usr/local/sbin && export
    PATH

OVS_APP=`which ovs-appctl`

# Stop vswitch deamon & vswitch server
$OVS_APP -t ovs-vswitchd exit && $OVS_APP -t ovsdb-server exit &

# Unload OVs module
modprobe -r openvswitch_mod || rmmod openvswitch_mod
echo "OpenVswitch module unloaded"

# Load bridge module
modprobe bridge || insmod /lib/modules/`uname -r`/kernel/net/bridge/bridge.ko
echo "bridge module loaded"

# Load ip_gre module for linux EoGRE interfaces
modprobe ip_gre && echo "ip_gre module loaded!"

echo "OpenVswitch succesfully stopped!"

exit
```

Listing Γ'.7: src/vsys/ovs–start

```
#!/bin/sh
# File: ovs-start
# Author: Emmanouil Dimogerontakis
# Date: 18/07/2011
#
# Vsys script ran by Network Administrator to start ovs
# execution in the root-context

SLICE=$1

OVS=`echo $OVS`

# Add ovs paths if needed
echo $PATH | grep local || PATH=$PATH:/usr/local/bin:/usr/local/sbin && export
    PATH

# Remove the ip_gre module to use the OVS GRE implemantation
lsmod | grep ip_gre && modprobe -r ip_gre

# Load the ovs module if needed
lsmod | grep openvswitch || modprobe -r bridge; insmod /root/${OVS}/datapath/
    linux-2.6/openvswitch_mod.ko

# Start the db server
`which ovsdb-server` /usr/local/etc/openvswitch/conf.db \
--remote=punix:/usr/local/var/run/openvswitch/db.sock \
--remote=db:Open_vSwitch,manager_options \
--pidfile \
--detach

sleep 3
# Start the vsctl service
`which ovs-vsctl` --no-wait init
# Start the vswitch deamon
`which ovs-vswitchd` unix:/usr/local/var/run/openvswitch/db.sock \
--pidfile \
--detach

echo "OpenVswitch succesfully started!"
exit
```

Listing Γ′.8: src/vsys/setup–stitch

```
#!/bin/sh +x
# File: setup-stitch
# Author: Emmanouil Dimogerontakis
# Date: 18/07/2011
#
# Vsys script to setup stitch interfaces for a switch sliver

SLICE=$1
SLICEID=`id -u $SLICE`
read REMOTE
read KEY


echo "remote $REMOTE"
echo "Started stitching"
OVS_CMD="/usr/local/bin/ovs-vsctl"
BRIDGE=br0
EGRE=stitch$KEY

# Creating OVS EoGRE interface with tag the gre key
echo "Adding Egre iface"
$OVS_CMD -- add-port $BRIDGE $EGRE tag=$KEY -- set interface $EGRE type=gre
    options="remote_ip=$REMOTE,key=$KEY"

# Creating the delete scipt
echo "creating delete script"
### Create "delete link" script
DELETE=/vsys/local_delete-$EGRE
  echo $SLICE > $DELETE.acl
  rm -f $DELETE
  cat > $DELETE <<EOF
#!/bin/sh

read NULL

$OVS_CMD del-port $BRIDGE $EGRE

EOF

# Make delete script executable
chmod +x $DELETE
echo "Ended stitching"
```

Listing Γ'.9: src/vsys/setup–vlink

```
#!/bin/sh +x
# File: setup-vlink
# Author: Emmanouil Dimogerontakis
# Date: 18/07/2011
#
# Vsys script to setup interfaces for virtual link with OVS


SLICE=$1
SLICEID=`id -u $SLICE`
read INDEX # interface index
read REMOTE # remote end-point for GRE tunnel
read KEY # GRE key
read SWITCH # Boolean, 1 if switch slice else 0
read LMAC # MAC of slice interface

echo "remote $REMOTE"
LINK=${KEY}if${INDEX}
OVS_CMD="/usr/local/bin/ovs-vsctl"
OFS_CMD="/usr/local/bin/ovs-ofctl"


### Setup bridge
# One bridge for all the slices
BRIDGE=br0
$OVS_CMD br-exists $BRIDGE || $OVS_CMD add-br $BRIDGE
echo "Bridge ready and ..."
echo "... up!"
ifconfig $BRIDGE up
echo "... up!"


ETUN0=a$LINK

# Skip this part if switch slice and non zero interface
if ! [[ $SWITCH && $INDEX != "0" ]]; then
  ### Setup etun
  ETUN0=a$LINK
  ETUN1=b$LINK
  ip link add name $ETUN0 type veth peer name $ETUN1
  echo "Veth Interfaces created and ..."
  ip link set $ETUN0 address ${LMAC}
  ifconfig $ETUN0 mtu 1458 up
  ifconfig $ETUN1 up
  $OVS_CMD add-port $BRIDGE $ETUN1 tag=$KEY
  echo "... up!"
  #ETUN1PORT=`$OFS_CMD get Interface ${ETUN1} ofport`
fi


### Setup EGRE tunnel
# Add gre tunnel the OVS way
EGRE=d$LINK
$OVS_CMD -- add-port $BRIDGE $EGRE tag=$KEY -- set interface $EGRE type=gre
    options="remote_ip=$REMOTE,key=$KEY"
```

```
55
56
57
58 ### Setup iptables so that packets are visible in the vserver
59 iptables -t mangle -A FORWARD -o $BRIDGE -j MARK --set-mark $SLICEID
60
61
62 # Skip this part if switch slice and non zero interface
63 if ! [[ $SWITCH && $INDEX != "0" ]]; then
64   ### Create "grab link" script, sets veth0 to NetNS mode
65   GRAB=/vsys/local_grab-$ETUN0
66   echo $SLICE > $GRAB.acl
67   rm -f $GRAB
68   cat > $GRAB <<EOF
69 #!/bin/sh
70
71 read PID
72
73 ip link set $ETUN0 netns \$PID
74 EOF
75   chmod +x $GRAB
76
77   ### Create script for user to set the link rate
78   BIND=/vsys/local_rate-$ETUN0
79   echo $SLICE > $BIND.acl
80   rm -f $BIND
81   cat > $BIND <<EOF
82 #!/bin/sh
83
84 read rt
85
86 tc qdisc add dev $EGRE root handle 1: htb default 10
87 tc class add dev $EGRE parent 1: classid 1:10 htb rate \$rt ceil \$rt
88
89 rm -rf $BIND.acl
90 touch $BIND.acl
91
92 EOF
93   chmod +x $BIND
94 fi
95
96 ### Create "delete link" script
97 DELETE=/vsys/local_delete-$ETUN0
98 echo $SLICE > $DELETE.acl
99 rm -f $DELETE
100 cat > $DELETE <<EOF
101 #!/bin/sh
102
103 read NULL
104
105 # Remove iptables rule
106 iptables -t mangle -D FORWARD -o $BRIDGE -j MARK --set-mark $SLICEID
107
108
109 # Get rid of bridge
110 echo "Deleting ports"
111 $OVS_CMD del-port  $BRIDGE $ETUN1
```

```
112 $OVS_CMD del-port  $BRIDGE $EGRE
113 # Check if is time to kill slice bridge
114 test -z \` $OVS_CMD list-ports $BRIDGE \` && ifconfig $BRIDGE down && $OVS_CMD
        del-br $BRIDGE; echo "Deleted bridge"
115
116 # Check if is time to kill Host Bridge
117 #test -z \` $OVS_CMD list-ports $HBRIDGE \` && ifconfig $HBRIDGE down &&
        $OVS_CMD del-br $HBRIDGE; echo "Deleted bridge"
118
119 # Get rid of etun devices, only need name of one of them
120 ip link delete dev $ETUN1
121 echo "Deleted $ETUN1"
122
123
124 # Clean up files
125 rm -f $GRAB $GRAB.acl
126 rm -f $DELETE $DELETE.acl
127 rm -f $BIND $BIND.acl
128 EOF
129 chmod +x $DELETE
```

Βιβλιογραφία

[1] Internet in a slice. `http://svn.planet-lab.org/wiki/ViniInternetInASlice`.

[2] Network virtualization. `http://en.wikipedia.org/wiki/Network_virtualization`.

[3] Platform virtualization. `http://en.wikipedia.org/wiki/Platform_virtualization`.

[4] Virtual private network. `http://en.wikipedia.org/wiki/Virtual_private_network`.

[5] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34 – 41, April 2005.

[6] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 3–14, 2006.

[7] S. Bhatia, M. Motiwala, W. Mühlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Hosting virtual networks on commodity hardware. Technical report, Georgia Tech, Computer Science Department, 2008.

[8] J. Carapinha and J. Jiménez. Network virtualization: a view from the bottom. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, VISA '09, pages 73–80, New York, NY, USA, 2009. ACM.

[9] N. Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20 –26, july 2009.

[10] S. Nanda and T. cker Chiueh. A survey of virtualization technologies. Technical report, 2005.

[11] NSF. Report on NSF Workshop on Network Research Testbeds. Technical report, US National Science Foundation, Chicago, IL, 2002.

[12] L. Peterson, A. Bavier, M. Fiuczynski, S. Muir, and T. Roscoe. Towards a comprehensive planetlab architecture. Technical Report PDN–05–030, PlanetLab Consortium, June 2005.

[13] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. Planetlab architecture: An overview. Technical Report PDN–06–031, PlanetLab Consortium, May 2006.

[14] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. e.a.: Extending networking into the virtualization layer. In *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009*, October 2009.

[15] R. Venkateswaran. Virtual private networks. *Potentials, IEEE*, 20(1):11 –15, feb/mar 2001.