ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων
Πληροφορικής

Μελέτη και αξιολόγηση του Containerlab για την εξομοίωση δικτυακών τοπολογιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Άννα Π. Κουτσώνη

Επιβλέπων : Ευστάθιος Συκάς
Ομότιμος Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων
Πληροφορικής

Μελέτη και αξιολόγηση του Containerlab για την εξομοίωση δικτυακών τοπολογιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Άννα Π. Κουτσώνη

Επιβλέπων : Ευστάθιος Συκάς
Ομότιμος Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Σεπτεμβρίου 2025.

.....	.....	.....
Ευστάθιος Συκάς	Νικόλαος Μήτρου	Ιωάννα Ρουσσάκη
Ομότιμος Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Αναπληρώτρια Καθηγήτρια Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2025

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

Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει τις δυνατότητες του Containerlab, ενός περιβάλλοντος προσομοίωσης δικτύων βασισμένο σε τεχνικές ελαφράς εικονικοποίησης, με στόχο την διεύρυνση της χρήσης containers ως εναλλακτική στη χρήση εικονικών μηχανημάτων κατά την εξομοίωση δικτυών. Αρχικά παρουσιάζεται η λειτουργία εικονικών μηχανημάτων (Virtual Machines) και containers και συντελείται σύγκριση μεταξύ των δύο σε τεχνικό επίπεδο και αξιολόγηση τους. Στη συνέχεια αναλύεται λεπτομερώς η πλατφόρμα του Docker, το οποίο αποτελεί τη βάση του Containerlab. Ακολουθεί αναλυτική παρουσίαση του Containerlab με περιγραφή της αρχιτεκτονικής του, του τρόπου λειτουργίας του, των μηχανισμών δικτυακής διασύνδεσης του αλλά και των βασικότερων εντολών. Το πειραματικό σκέλος της εργασίας περιλαμβάνει την υλοποίηση πολλαπλών δικτυακών τοπολογιών, οι οποίες αξιοποιούν διάφορα πρωτόκολλα δρομολόγησης (RIP, OSPF, BGP) και σύγχρονες τεχνολογίες δικτύου (ECMP, VXLAN, EVPN, Switches). Για τις τοπολογίες αυτές παρουσιάζονται σημαντικά βήματα της υλοποίησης τους και της τελικής συμπεριφοράς του δικτύου, καθώς και αξιοσημείωτες ιδιαιτερότητες τους στα πλαίσια του Containerlab. Γίνεται εκτενής αναφορά στο θεωρητικό υπόβαθρο των παραπάνω πρωτοκόλλων και τεχνολογιών, όπως και των διαφόρων εργαλείων που αξιοποιήθηκαν στα πλαίσια της πειραματικής μελέτης. Στο τέλος διερευνάται η κατανάλωση πόρων εκ μέρους του Containerlab με στόχο την αξιολόγηση της επίδοσης του και την εξαγωγή συμπερασμάτων για την εφικτότητα της ευρείας χρήσης του. Για το σκοπό αυτό μελετάται η συμπεριφορά του συστήματος υπό πολλαπλά σενάρια χρήσης με διαφορετικές συνθήκες όπως ο αριθμός των χρηστών και ο φόρτος των τοπολογιών. Τα τελικά αποτελέσματα υποδεικνύουν υψηλή αποδοτικότητα και ευελιξία του Containerlab αναδεικνύοντας το ως μια ιδανική πλατφόρμα για την ανάπτυξη, δοκιμή και προσομοίωση σύνθετων δικτυακών τοπολογιών, ικανή να ωφελήσει σημαντικά τον τομέα των δικτύων.

Λέξεις Κλειδιά: Containerlab, Docker, Containers, Εικονικοποίηση, Προσομοίωση Δικτύων, YAML, RIP, OSPF, BGP, ECMP, VXLAN, EVPN

Abstract

This diploma thesis examines the capabilities of Containerlab, a network simulation environment based on light virtualization techniques, with the aim of expanding the use of containers as an alternative to the use of virtual machines in network simulation. Initially, the operation of virtual machines and containers is presented and a comparison between the two at a technical level and their evaluation is conducted. Then, the Docker platform, which is the basis of Containerlab, is analyzed in detail. This is followed by a detailed presentation of Containerlab with a description of its architecture, its operation, its networking mechanisms and some of the most basic commands. The experimental part of the diploma thesis includes the implementation of multiple network topologies, which include various routing protocols (RIP, OSPF, BGP) and modern network technologies (ECMP, VXLAN, EVPN, Switches). For these topologies, important steps of their implementation and the final behavior of the network are presented, as well as any notable features within the framework of Containerlab. There has been added an extensive mention of the theoretical background of the above protocols and technologies, as well as the various tools that were used during the experimental study. Finally, the resource consumption of Containerlab is investigated in order to evaluate its performance and draw conclusions about the feasibility of its widespread use. For this purpose, the behavior of the system is studied under multiple usage scenarios with different conditions such as the number of users and the load of the topologies. The final results indicate high efficiency and flexibility of Containerlab, highlighting it as an ideal platform for the development, testing and simulation of complex network topologies, capable of significantly benefiting the network field.

KeyWords: Containerlab, Docker, Containers, Virtualization, Network Simulation, YAML, RIP, OSPF, BGP, ECMP, VXLAN, EVPN

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα καθηγητή κ. Ευστάθιο Συκά για την συνεργασία και την υποστήριξη που μου προσέφερε καθ'όλη τη διάρκεια. Επιπλέον, θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου και τους ανθρώπους μου, οι οποίοι με στήριξαν όλα αυτά τα χρόνια και συνεχίζουν να στηρίζουν όλες μου τις προσπάθειες.

Περιεχόμενα

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ, ΠΙΝΑΚΩΝ ΚΑΙ ΣΧΗΜΑΤΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ ΚΑΙ ΑΚΡΩΝΥΜΙΑ

ABR	Border Gateway Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ASBR	Autonomous System Boundary Router
ASN	Autonomous System Number
AUFS	Advanced multi-layered Unification Filesystem
BDR	Backup Designated Router
BFD	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
BR	Backbone Router
CI/CD	Continuous Integration/Continuous Delivery
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name System
DR	Designated Router
EAD	Ethernet Auto-Discovery
eBGP	external Border Gateway Protocol
ECMP	Equal Cost Multi-Path
EIGHRP	Enhanced Interior Gateway Protocol
EVPN	Ethernet Virtual Private Network
FDB	Forwarding Database
FRR	Free Range Routing
HTML5	HyperText Markup Language version 5
iBGP	internal Border Gateway Protocol
ICMP	Internet Control Message Protocol
ID	Identifier
IGP	Interior Gateway Protocol
IP	Internet Protocol
IPC	Inter-Process Communication
IR	Internal Router
IS-IS	Intermediate System to Intermediate System
LAN	Local Area Network
LDP	Label Distribution Protocol
LSA	Link-State Advertisement
LSDB	Link-State Database
LXC	Linux Containers
MAC	Media Access Control
MTU	Maximum Transmission Unit
NHRP	Next Hop Resolution Protocol
NIC	Network Interface Card
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First

OVS	Open vSwitch
PID	Process Identifier
PIM	Protocol Independent Multicast
POSIX	Portable Operating System Interface (for UNIX)
QoS	Quality of Service
RAM	Random Access Memory
RHEL	Red Hat Enterprise Linux
RIP	Routing Information Protocol
SDN	Software-Defined Networking
SPF	Shortest Path First
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UTS	UNIX Time-Sharing
VLAN	Virtual Local Area Network
VLSM	Variable Length Subnet Masking
VM	Virtual Machine
VNI	Virtual Network Identifier
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual Extensible Local Area Network
WLAN	Wireless Local Area Network
WSL	Windows Subsystem for Linux
YAML	YAML Ain’t Markup Language

1. Εισαγωγή

1.1 Το πλαίσιο του προβλήματος

Η ανάπτυξη και η διαρκής εξάπλωση των cloud τεχνολογιών καθιστούν αναγκαία την ένταξη τους και στον τομέα της διαχείρισης δικτύων. Μέχρι πρότινος καθοριστικός περιοριστικός παράγοντας στην γρήγορη υλοποίηση δικτυακών τοπολογιών και στον έλεγχο της ορθής λειτουργίας τους, ήταν η πρόσβαση σε φυσικό εξοπλισμό, γεγονός χρονοβόρο, δαπανηρό και με υψηλές απαιτήσεις διαθέσιμου χώρου. Συνεπώς προέκυψε η ανάγκη δημιουργίας εργαλείων προσομοίωσης, τόσο για χρήση στην εκπαιδευτική διαδικασία, όσο και στην έρευνα και τη βιομηχανία. Ακολούθησε η εισαγωγή των εικονικών μηχανημάτων, τα οποία προσέφεραν δραστικές αλλαγές και διευκολύνσεις στην διαχείριση δικτύων. Ωστόσο η χρήση τους συνοδεύτηκε από νέες προκλήσεις και προβλήματα σχετικά με την κατανάλωση πόρων, την πολυπλοκότητα της διαχείρισης και την αποδοτικότητα. Σε αυτό το σημείο η εμφάνιση των containers έφερε μια νέα προσέγγιση στη διαχείριση δικτύων που προσφέρει ελαφρύτερη, ταχύτερη και αποδοτικότερη εκτέλεση εφαρμογών. [1,3]

1.2 Σκοπός της διπλωματικής εργασίας

Σκοπός της παρούσας διπλωματικής εργασίας είναι να διερευνήσει τις δυνατότητες του περιβάλλοντος εξομοίωσης Containerlab, το οποίο βασίζεται σε τεχνικές ελαφράς εικονικοποίησης (light virtualization) των containers, όσων αφορά την υλοποίηση δικτυακών τοπολογιών, οι οποίες αξιοποιούν διάφορα πρωτόκολλα δρομολόγησης (BGP,RIP,OSPF) και άλλες δικτυακές τεχνολογίες (VXLAN, EVPN, ECMP). Στόχος της μελέτης αυτής είναι η ένταξη του Containerlab στην εκπαιδευτική διαδικασία στη θέση των εικονικών μηχανημάτων.
Συγκεκριμένα επιδιώκονται:

1. Η ανάλυση του τεχνολογικού υποβάθρου της λειτουργίας των containers.
2. Η παρουσίαση του περιβάλλοντος εξομοίωσης Containerlab.
3. Η δημιουργία πολλαπλών δικτυακών τοπολογιών σε συνδυασμό με την θεωρητική ανάλυση των τεχνολογιών που αξιοποιούνται και τη μελέτη της συμπεριφοράς του εκάστοτε δικτύου.
4. Η εξέταση των απαιτούμενων πόρων για τις παραπάνω διαδικασίες με βάση την οποία πραγματοποιείται συνολική αξιολόγηση της αποδοτικότητας του Containerlab.

1.3 Μεθοδολογία και υλοποίηση

Για την επίτευξη των παραπάνω στόχων παρουσιάζεται λεπτομερώς ο τρόπος λειτουργίας αρχικά των εικονικών μηχανημάτων (virtual machines) και η μέθοδος με την οποία αυτά χρησιμοποιούνται για την προσομοίωση δικτυακών τοπολογιών και έπειτα των containers και των τεχνικών που αξιοποιεί το Containerlab για την δημιουργία και τη διαχείριση εργαστηρίων δικτύωσης.

Στη συνέχεια υλοποιούνται πολλαπλές δικτυακές τοπολογίες σε περιβάλλον Containerlab, παρατίθενται και αναλύονται τα απαραίτητα βήματα και οι κώδικες για την παραμετροποίηση και την ορθή λειτουργία τους. Παράλληλα δίνεται έμφαση στο θεωρητικό υπόβαθρο των δικτυακών τεχνολογιών που χρησιμοποιούνται και αξιοποιούνται εργαλεία ανάλυσης δικτυακής κυκλοφορίας και γραφικής απεικόνισης δικτυακών συνδέσεων, όπως το Wireshark, Edgeshark, tcpdump, bmon.

Τέλος, εξετάζεται η κατανάλωση πόρων που απαιτεί το Containerlab και η ταυτόχρονη υλοποίηση των παραπάνω τοπολογιών μέσω της δημιουργίας πολλαπλών χρηστών σε έναν server εντός του δικτύου του Ε.Μ.Π., οι οποίοι εργάζονται παράλληλα. Διερευνώνται διάφορα σενάρια αριθμού χρηστών και φόρτου εργασίας και ελέγχεται η απόδοση του συστήματος.

2. Full vs Light Virtualization

Η εικονικοποίηση αποτελεί μια θεμελιώδη τεχνική ευρείας χρήσης, η οποία καθιστώντας δυνατή τη δημιουργία πολλαπλών εικονικών περιβαλλόντων σε ένα φυσικό μηχάνημα μείωσε δραστικά τις υψηλές απαιτήσεις σε εξοπλισμό, χώρο, κόστη συντήρησης, διευκόλυνε και επιτάχυνε σημαντικά την ταυτόχρονη διαχείριση εφαρμογών με αποτέλεσμα η πρόσβαση σε διάφορες τεχνολογίες και στην έρευνα να είναι πλέον εφικτή για πολύ μεγαλύτερο μέρος του πληθυσμού.

2.1 Virtual Machines

Μια εικονική μηχανή είναι μια ολοκληρωμένη εξομοίωση ενός υπολογιστικού συστήματος μέσω ενός λογισμικού το οποίο του επιτρέπει να εκτελεί όλες τις λειτουργίες του συστήματος που εξομοιώνει σε ένα περιβάλλον απομονωμένο και από τα υπόλοιπα εικονικά μηχανήματα και από το φυσικό μηχάνημα. Η εξομοίωση υλικού (hardware virtualization) γίνεται με τη χρήση ενός λογισμικού (hypervisor), το οποίο λειτουργεί ως ενδιάμεσο στρώμα μεταξύ φυσικής και εικονικής μηχανής και χρησιμοποιείται για να διαχειρίζεται και να κατανέμει στα εικονικά μηχανήματα τους φυσικούς πόρους του συστήματος όπως CPU, μνήμη, αποθηκευτικό χώρο. Ένας hypervisor είναι επίσης υπεύθυνος για την απομόνωση (isolation) μεταξύ των εικονικών μηχανών αποτρέποντας την μεταξύ τους αλληλεπίδραση. Κάθε εικονική μηχανή έχει το δικό της ολόκληρο λειτουργικό σύστημα (OS), το οποίο μπορεί και να διαφέρει από το λειτουργικό του φυσικού μηχανήματος, αφού τα εικονικά μηχανήματα έχουν περιορισμένα δικαιώματα σχετικά με τις εντολές που μπορούν να εκτελούν και παρεμβαίνει ο hypervisor μεταφράζοντας τις εντολές του εικονικού μηχανήματος σε αντίστοιχες εντολές στο λειτουργικό του φυσικού μηχανήματος που μπορούν να εκτελεστούν από αυτό. [5, 6, 10]

2.2 Containers

Ένα container αποτελεί μια ελαφριά μονάδα εκτελέσιμου λογισμικού που βασίζεται στην εικονικοποίηση λειτουργικού συστήματος (OS virtualization), κατά την οποία το κάθε container δεν έχει το δικό του λειτουργικό σύστημα όπως στην περίπτωση των εικονικών μηχανημάτων (hardware virtualization) αλλά βασίζεται στον πυρήνα (kernel) του λειτουργικού συστήματος του φυσικού διακομιστή. Επιπλέον δεν απαιτείται η ύπαρξη hypervisor αφού τώρα ο διαμοιρασμός πόρων και η απομόνωση διεργασιών εκτελείται από τον πυρήνα του λειτουργικού συστήματος. Σε κάθε container περιέχονται οι απαραίτητες εξαρτήσεις και βιβλιοθήκες για την εκτέλεση μιας εφαρμογής διασφαλίζοντας τη δυνατότητα αυτή να εκτελείται με συνέπεια σε

διαφορετικά περιβάλλοντα, ενώ παράλληλα εξασφαλίζεται η απομόνωση όλων των διαδικασιών που περιέχονται στο container από ταυτόχρονες διαδικασίες του φυσικού συστήματος ή άλλων container. [5, 6, 10]

2.3 Namespaces & Cgroups

Η λειτουργία των containers βασίζεται σε δύο θεμελιώδη χαρακτηριστικά του πυρήνα του Linux: τα namespaces και τα cgroups (control groups).

Τα namespaces είναι ένας μηχανισμός απομόνωσης συγκεκριμένων πόρων του λειτουργικού συστήματος (global resources) με τρόπο τέτοιο ώστε να φαίνεται ότι οι διεργασίες που ανήκουν στο εκάστοτε namespace, δηλαδή το κάθε container, έχει το δικό του απομονωμένο κομμάτι των συνολικών πόρων. Οι διεργασίες εντός ενός namespace δεν έχουν πρόσβαση στους αντίστοιχους πόρους άλλων namespaces. [5, 8]

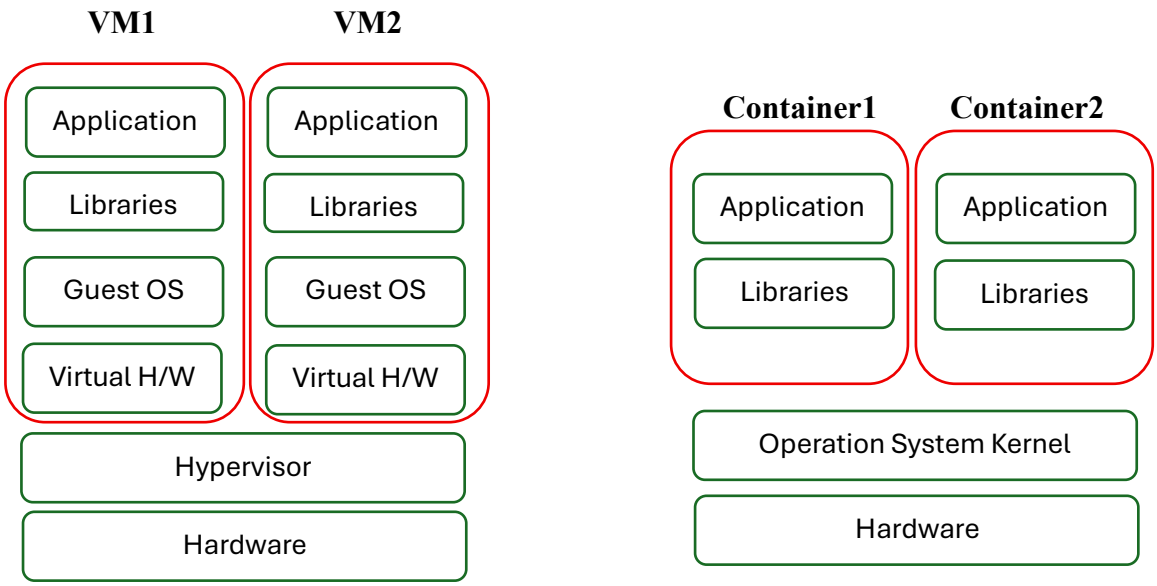
Τα διαθέσιμα είδη namespace στο Linux είναι τα εξής:

- PID → Απομονώνει τα αναγνωριστικά των διεργασιών (process IDs).
- Time → Απομονώνει τον χρόνο που μετράει από την εκκίνηση του συστήματος (monotonic clock) και τον χρόνο από την εκκίνηση του συστήματος συμπεριλαμβανομένου του χρόνου που το σύστημα ήταν σε αδράνεια (sleep/suspend), (boot clock).
- User → Απομονώνει τα αναγνωριστικά χρηστών και ομάδων (user, group IDs).
- UTS → Απομονώνει τα hostname, domain name.
- Mount → Απομονώνει το σύστημα αρχείων (filesystem).
- Network → Απομονώνει το σύνολο των δικτυακών πόρων.
- IPC → Απομονώνει τα αντικείμενα IPC όπως κοινή μνήμη, ουρές μηνυμάτων (System V IPC, POSIX message queues).
- Cgroup → Απομονώνει τους πόρους για τα cgroups και αποτελεί το root directory τους.

Τα cgroups (control groups) παρέχουν έναν μηχανισμό συγκέντρωσης και διαμέρισης συνόλων εργασιών και όλων των μελλοντικών υποεργασιών που θα προκύψουν από αυτά τα σύνολα, σε ιεραρχικές ομάδες με εξειδικευμένη συμπεριφορά. Επιπλέον διανέμουν δυναμικά τους πόρους στα containers και καθορίζουν όρια για την κατανάλωση τους. Οι διαθέσιμοι τύποι πόρων που μπορούν τα cgroups να διαχειριστούν είναι οι εξής: CPU, memory, network, block-IO, devices. [5, 12, 13]

2.4 Σύγκριση VMs – Containers

Στο παρακάτω διάγραμμα φαίνεται μια σύγκριση της αρχιτεκτονικής των VMs και των container.



Εικόνα 1: *Σύγκριση της αρχιτεκτονικής VM και Container*

Όπως αναφέρθηκε παραπάνω και φαίνεται και στο προηγούμενο σχήμα, η βασική διαφορά των containers από τα VMs είναι η έλλειψη hypervisor και το γεγονός ότι μοιράζονται με φυσικό μηχάνημα τον πυρήνα του λειτουργικού του συστήματος. Κατά συνέπεια τα containers συνιστούν πολύ ελαφρύτερες δομές από τα VMs, με μέγεθος συνήθως μερικές δεκάδες MB ενώ μια εικονική μηχανή μπορεί να φτάσει μέχρι και κάποια GB. Επιπλέον έχουν πολύ μικρότερες απαιτήσεις σε μνήμη RAM και χρήση CPU και πολύ ταχύτερους χρόνους εκκίνησης και δημιουργίας, από μερικά χιλιοστά του δευτερολέπτου έως κάποια δευτερόλεπτα, ενώ τα VMs χρειάζονται από κάποια δευτερόλεπτα έως και κάποια λεπτά. Απόρροια όλων των παραπάνω είναι η δυνατότητα υποστήριξης πολλών container ταυτόχρονα από ένα σύστημα χωρίς αυτό να επιβαρύνεται σημαντικά και να υφίσταται περιορισμούς στη λειτουργία του. Ένα ακόμη βασικό πλεονέκτημα των containers είναι η ικανότητα τους να αποθηκεύονται, να μεταφέρονται και να αναπαράγονται σε διαφορετικά συστήματα με απλό και γρήγορο τρόπο χωρίς να υφίστανται αλλοιώσεις διευκολύνοντας έτσι την κοινή χρήση εργασιών μεταξύ συνεργατών, την δοκιμή και την επαναφορά παραμετροποιήσεων με πολύ μικρούς χρόνους αναμονής. [3, 4, 5, 7, 9, 11]

Τα παραπάνω χαρακτηριστικά των containers μπορούν να αξιοποιηθούν στον τομέα των δικτύων προσφέροντας πρακτικά οφέλη. Παρέχεται ένα περιβάλλον το οποίο επιτρέπει την άμεση δημιουργία και τροποποίηση διαφόρων σεναρίων και την εξομοίωση δικτυακών τοπολογιών χωρίς την ανάγκη φυσικού εξοπλισμού ή άλλες πολύπλοκες διαδικασίες που θα απαιτούσε η εγκατάσταση εικονικών μηχανών.

Επίσης τα υψηλά επίπεδα απομόνωσης και ελέγχου των containers που έχει στη διάθεση του ο χρήστης διευκολύνουν σε μεγάλο βαθμό την αξιοποίηση τους για επίλυση προβλημάτων (troubleshooting) σε σύγχρονες δικτυακές υποδομές. [3, 4, 5, 7, 9]

3. Docker

Το 2013 κυκλοφόρησε το Docker, μια πλατφόρμα ανοικτού κώδικα υλοποιημένη στη γλώσσα προγραμματισμού Go, η οποία αξιοποίησε κυρίως ήδη υπάρχουσες τεχνικές των Linux όπως τα Namespaces, Cgroups, Linux Containers (LXC), Union Filesystems (AUFS, OverlayFS). Ενώνοντας αυτές τις τεχνικές και εισάγοντας νέες λειτουργίες έκανε τα containers πολύ πιο εύκολα στη χρήση και προσβάσιμα προς το χρήστη αυξάνοντας τις δυνατότητες τους, με αποτέλεσμα να διευρύνονται διαρκώς τα πεδία στα οποία μπορούσαν να αξιοποιηθούν. Βελτιώνοντας την εμπειρία του χρήστη και προσφέροντας νέες αποδοτικότερες λύσεις για υπάρχοντα προβλήματα, το Docker κατάφερε σε μικρό χρονικό διάστημα να αντικαταστήσει ένα πλήθος παλιών τεχνικών και να εξελιχθεί σε ένα πολύ αξιόλογο και απαραίτητο εργαλείο, το οποίο σήμερα μετρά πάνω από 20 εκατομμύρια χρήστες. [14, 16, 17, 18]

3.1 Εφαρμογές και χρήσεις

Το Docker δίνει στους χρήστες τη δυνατότητα να αναπτύσσουν τον κώδικά τους τοπικά κάνοντας δοκιμές και διορθώσεις και χάρη στις ιδιότητες απομόνωσης και φορητότητας των containers να προωθούν εύκολα τις αλλαγές τους με λιγότερα σφάλματα, διευκολύνοντας σημαντικά τις μεθόδους συνεχούς ολοκλήρωσης και παράδοσης (continuous integration, continuous delivery – CI/CD).

Ιδιαίτερα εύχρηστη είναι επίσης η ικανότητα του Docker να λειτουργεί άρτια σε ποικίλα περιβάλλοντα, όπως προσωπικούς υπολογιστές, εικονικές μηχανές, παρόχους cloud υπηρεσιών ή και συνδυασμούς αυτών χωρίς υψηλές απαιτήσεις σε κατανάλωση πόρων.

Επιπλέον μέσω του Docker είναι εφικτή η αυτοματοποιημένη δημιουργία containers βασισμένων στον πηγαίο κώδικα μιας εφαρμογής. Η διαδικασία αυτή οδηγεί στην δημιουργία ενός container image για το οποίο δύνανται να αποθηκεύονται προς χρήση όλες οι εκδοχές του (versions) αλλά και αρχεία που περιέχουν μόνο τις αλλαγές μεταξύ εκδόσεων (delta files).

Η αρχιτεκτονική των microservices είναι μια προσέγγιση σύμφωνα με την οποία μια εφαρμογή χωρίζεται σε μικρές, ανεξάρτητες υπηρεσίες (microservices) που επικοινωνούν μεταξύ τους. Η υλοποίηση αυτή απλοποιείται δραστικά με τη χρήση του Docker, καθώς κάθε microservice μπορεί να διαχειρίζεται ανεξάρτητα ως ένα αυτόνομο container. Ο συνδυασμός αυτών των δύο δημιουργεί ένα πολύ ευνοϊκό περιβάλλον για πειραματισμό και ευελιξία όσων αφορά την ανάπτυξη λογισμικού και υπηρεσιών (DevOps) συμβάλλοντας ουσιαστικά στην άμεση ανταπόκριση των μηχανικών στις απαιτήσεις της αγοράς. Η τεχνολογία του Docker αξιοποιείται και

από πολλούς cloud παρόχους, οι οποίοι προσφέρουν CaaS (Container-as-a-Service) πλατφόρμες. [16, 17]

3.2 Αρχιτεκτονική και Εσωτερική Λειτουργία

Το Docker χρησιμοποιεί το μοντέλο πελάτη-διακομιστή (client-server). Τα δύο βασικότερα στοιχεία της αρχιτεκτονικής του είναι ο Docker client και ο Docker daemon. Η client-server εφαρμογή που περιλαμβάνει τον Docker daemon, ένα Docker API και μια γραμμή εντολών (CLI) για αλληλεπίδραση με αυτόν, ονομάζεται Docker Engine.

Ο Docker daemon (dockerd) αποτελεί τον βασικό κεντρικό μηχανισμό, ο οποίος εκτελεί τις εντολές του Docker client, δημιουργεί, διαχειρίζεται και καταστρέφει τα Docker objects, δηλαδή τα στοιχεία μιας υλοποίησης (containers, images, volumes, networks). Επικοινωνεί επίσης με άλλους δαίμονες με σκοπό την διαχείριση Docker services.

Ο Docker client (docker) είναι ο πρωτεύον τρόπος αλληλεπίδρασης του χρήστη με την πλατφόρμα καθώς παρέχει τη γραμμή εντολών (CLI) και μέσω του Docker API στέλνει τις εντολές στον Docker daemon για εκτέλεση. Μπορεί να επικοινωνεί ταυτόχρονα με περισσότερους από έναν δαίμονες. Ο χρήστης μπορεί είτε να συνδέσει τον Docker client σε έναν απομακρυσμένο Docker daemon είτε σε κάποιον που βρίσκεται στο ίδιο σύστημα. [16, 17]

3.3 Docker Images

Η δημιουργία των containers στο περιβάλλον του Docker βασίζεται στα Docker Images. Ένα image είναι ένα read-only αρχείο το οποίο περιέχει εκτελέσιμο κώδικα με όλες τις απαραίτητες εντολές και παραμετροποιήσεις, τα αρχεία, τις βιβλιοθήκες και τις εξαρτήσεις που απαιτούνται για να δημιουργηθεί ένα container με τα επιθυμητά χαρακτηριστικά και εφαρμογές. Η βάση ενός image είναι το Dockerfile, ένα απλό αρχείο κειμένου που αυτοματοποιεί τη διαδικασία κατασκευής του image περιέχοντας τις απαιτούμενες για αυτήν εντολές. Κάθε εντολή του Dockerfile αποτελεί και ένα διαφορετικό στρώμα (layer) του image, το οποίο αντιστοιχεί και σε μια διαφορετική έκδοση (version) του image. Αυτό έχει ως αποτέλεσμα κάθε φορά που το image υφίσταται μια μεταβολή να προστίθεται και ένα νέο στρώμα πάνω από το προηγούμενο και άρα να δημιουργείται και μια νέα έκδοση του image, η οποία χαρακτηρίζεται ως η πιο πρόσφατη “latest”. Οι προηγούμενες εκδόσεις αποθηκεύονται παρέχοντας στο χρήστη τη δυνατότητα να ανατρέξει σε αυτές και να τις επαναχρησιμοποιήσει. Σημειώνεται ότι μετά τη δημιουργία του ένα image δεν

επιδέχεται απευθείας αλλαγές και σε αντίθεση με τα containers παραμένει αμετάβλητο διατηρώντας την αρχική του μορφή. Συνεπώς οποιαδήποτε αλλαγή συνεπάγεται την κατασκευή καινούριου image με διαφορετικό ID. Κατά την αναπροσαρμογή του image και την δημιουργία του νέου μόνο οι εντολές των στρωμάτων που υπέστησαν αλλαγές επανεκτελούνται, γεγονός πολύ ωφέλιμο για την κατανάλωση πόρων.

Κάθε container δημιουργείται με βάση κάποιο image χωρίς να το επηρεάζει καθόλου. Προσθήκες ή αλλαγές εντός του container είναι εφικτές αλλά υφίστανται μόνο στα πλαίσια του container όσο αυτό υπάρχει, καθώς δεν αποθηκεύονται στο image ούτε σχετίζονται με αυτό. Η ιδιότητα αυτή προσφέρει τη δυνατότητα κατασκευής πολλών διαφορετικών container που έχουν κάποια θεμελιώδη κοινά χαρακτηριστικά από ένα μόνο image ως βάση, καθιστώντας έτσι τις απαιτήσεις της όλης διαδικασίας σε μνήμη, αποθηκευτικό χώρο, CPU και ταχύτητα αρκετά χαμηλές.
Για τη αποθήκευση των images και τον εύκολο διαμοιρασμό τους μεταξύ των χρηστών έχουν δημιουργηθεί κεντρικά αποθετήρια τα οποία ονομάζονται Docker registries. Σε αυτά είναι δυνατή η αναζήτηση και η εύρεση ενός image και διαφορετικών εκδόσεων του. Υπάρχουν ιδιωτικά και δημόσιες registries. Το πιο γνωστό δημόσιο το οποίο αποτελεί και προεπιλογή του Docker για την αναζήτηση images είναι το Docker Hub. [16, 17, 19]

3.4 Docker Volumes & Bind Mounts

Όπως προαναφέρθηκε τα containers αποτελούν εφήμερες δομές, τα δεδομένα των οποίων χάνονται με τον τερματισμό ή την διαγραφή τους. Σε περίπτωση που αυτό δεν είναι επιθυμητό το Docker περιλαμβάνει δύο μηχανισμούς για μόνιμη αποθήκευση δεδομένων, τα volumes και τα bind mounts.
Τα Docker volumes διαχειρίζονται πλήρως από το Docker και δημιουργούν έναν νέο συγκεκριμένο φάκελο στο σύστημα του Docker host, του συστήματος δηλαδή στο οποίο λειτουργεί το Docker, για να αποθηκεύουν εκεί τα αρχεία των container που ορίζονται. Η ανεξαρτησία τους από τη δομή του συστήματος αρχείων του του Docker host τα καθιστά ιδανικά για δημιουργία αντιγράφων ασφαλείας και μεταφορά δεδομένων.

Αντίθετα τα bind mounts ελέγχονται από τον χρήστη και τον docker host. Δεν δημιουργούν καινούριους φακέλους μέσα στο σύστημα για να αποθηκεύσουν τα δεδομένα των containers αλλά συνδέουν αρχεία και φακέλους των containers με άλλα ήδη υπάρχοντα του host. Οι τροποποιήσεις στα αρχείο του host εμφανίζονται και μεταφέρονται αυτόματα και στα αντίστοιχα συνδεδεμένα με αυτά αρχεία του container και αντίστροφα. Το χαρακτηριστικό αυτό παρά την πρακτικότητα του μπορεί αν δεν χρησιμοποιηθεί με προσοχή να επιφέρει κινδύνους ασφαλείας αφού δίνει στο container εν μέρει πρόσβαση στα αρχεία του host. [20, 21]

3.5 Docker Compose

Ένα χρήσιμο εργαλείο του Docker είναι το Docker Compose, το οποίο χρησιμοποιείται για τον ορισμό και την διαχείριση εφαρμογών που αποτελούνται από πολλά containers. Απλοποιεί σημαντικά την διαδικασία αντιμετωπίζοντας τα container σαν ένα ενιαίο σύστημα και ενσωματώνοντας όλες τις εντολές για την δημιουργία και την παραμετροποίηση του συνόλου των containers της εκάστοτε εφαρμογής, σε ένα μόνο αρχείο YAML. Το προκαθορισμένο όνομα αρχείου εντός του τρέχοντος φακέλου για αυτή τη διαδικασία είναι compose.yaml. Κάθε εφαρμογή μπορεί να έχει το δικό της compose.yaml αρχείο εντός του φακέλου της. [17, 22]

3.6 Χρήσιμες Εντολές του Docker

Παρατίθενται κάποιες βασικές εντολές του Docker και η χρήση τους. [23]

- *docker container ls* → Εμφάνιση όλων των υπαρχόντων running containers
- *docker container ls -a* → Εμφάνιση όλων των υπαρχόντων containers (running και σταματημένων)
- *docker ps* → Ισοδύναμη με την docker container ls
- *docker ps -a* → Ισοδύναμη με την docker container ls -a
- *docker start [Container_id]* → Εκκίνηση ενός container
- *docker stop [Container_id]* → Σταμάτημα ενός container
- *docker restart [Container_id]* → Επανεκκίνηση ενός container
- *docker exec -it { [Container_name] or [Container_id]} [Command]* → Εκτέλεση της εντολής *command* στο Container που προσδιορίζεται με το όνομα του
- *docker exec -it { [Container_name] or [Container_id]} bash* → Σύνδεση στο bash shell ενός container
- *docker rm [Container_id]* → Διαγραφή ενός container (Εάν το container είναι running θα πρέπει πρώτα να το σταματήσετε και μετά να το διαγράψετε)
- *docker images* → Εμφάνιση όλων τα διαθέσιμων images που έχουν ήδη φορτωθεί
- *docker build -t[Image_name:Tag] -f [Dockerfile path]* → Δημιουργία ενός image με βάση το Dockerfile
- *docker pull [Image_name:Tag]* → Download κάποιου image
- *docker rmi {[Image_name:Tag] or [Image_id]}* → Διαγραφή κάποιου image
- *docker inspect {[Image_name:Tag] or [Image_id]}* → Παροχή πληροφοριών για κάποιο image
- *docker network [create/connect/disconnect/inspect/ls/prune/rm]*→ Διαχείριση ενός δικτύου και εκτέλεση της εκάστοτε υποεντολής
- *docker volume [create/inspect/ls/prune/rm/update]*→ Διαχείριση ενός volume και εκτέλεση της εκάστοτε υποεντολής

3.7 Δικτύωση στο Docker

Η δικτύωση των container αφορά την δυνατότητα τους να συνδέονται και να επικοινωνούν μεταξύ τους ή με εξωτερικά δίκτυα. Το Docker υποστηρίζει πολλαπλές επιλογές διασύνδεσης. Ωστόσο τα containers δεν λαμβάνουν καμία πληροφορία ούτε για το είδος του δικτύου στο οποίο είναι συνδεδεμένα ούτε για το είδος των συστημάτων με τα οποία επικοινωνούν, αν δηλαδή εκείνα ανήκουν στο περιβάλλον Docker ή όχι. Ο χρήστης έχει την ευελιξία να δημιουργήσει δικά του δίκτυα και να συνδέσει σε αυτά πολλά container. Εναλλακτικά μπορεί να επιλέξει έναν από τους παρακάτω 6 drivers που προσφέρει το Docker για δικτύωση. [24]

Bridge
Πρόκειται για τον προεπιλεγμένο τρόπο δικτύωσης. Αν δεν επιλεχθεί κάποιος άλλος τότε το Docker χρησιμοποιεί αυτόματα τον bridge driver. Πρακτικά δημιουργείται ένα ιδιωτικό εσωτερικό δίκτυο στον docker host (φυσικό ή εικονικό μηχάνημα), επιτρέποντας στα containers που συνδέονται σε αυτό να επικοινωνούν μεταξύ τους και απομονώνοντας τα από όσα δεν είναι συνδεδεμένα σε αυτό το δίκτυο (bridge network).

Host
Στην περίπτωση αυτή τα containers δεν είναι απομονωμένα από τον docker host αλλά μοιράζονται με αυτόν το δικό του δίκτυο και χρησιμοποιούν απευθείας τη δική του IP διεύθυνση και τις δικές του θύρες.

Overlay
Χρησιμοποιείται για τη διασύνδεση containers που βρίσκονται σε διαφορετικούς docker hosts, υλοποιώντας ένα κατανεμημένο δίκτυο.

Ipvlan
Κάθε container έχει τη δική του IP αλλά μοιράζεται με τον docker host την MAC διεύθυνση του. Παρέχεται επίσης η δυνατότητα χρήσης του docker host ως δρομολογητή.

Macvlan
Σε κάθε container αποδίδεται μια δική του διεύθυνση MAC, κάνοντας το έτσι να φαίνεται σαν μια φυσική συσκευή στο εξωτερικό δίκτυο.

None
Απομονώνει πλήρως τα containers και από μεταξύ τους επικοινωνία και από τον docker host. Δημιουργείται μόνο μια loopback διεπαφή.

4. Containerlab

4.1 Εισαγωγή

Όπως προαναφέρθηκε οι δυνατότητες συλλογικής διαχείρισης container σε ένα μόνο αρχείο που εισήγαγε το Docker Compose προσέφεραν ευελιξία και απλότητα. Ωστόσο ο σχεδιασμός του δεν αφορά και δεν επαρκεί για προσομοίωση δικτυακών τοπολογιών. Δεν είναι εφικτός ο ορισμός συνδέσεων μεταξύ συγκεκριμένων διεπαφών των container και η δημιουργία συνδέσεων point-to-point. Επίσης δεν υποστηρίζονται πρωτόκολλα δρομολόγησης και παραμετροποιήσεις σε επίπεδο διεπαφών. Η απουσία ενός εύχρηστου εργαλείου που να βασίζεται στην χρήση container και να πληροί τις προϋποθέσεις για εξομοίωση δικτυακών τοπολογιών και πειραματισμό οδήγησε τους μηχανικούς της Nokia στην υλοποίηση του Containerlab, ενός CLI εργαλείου για την ανάπτυξη και τη διαχείριση εργαστηρίων δικτύων. Ο σκοπός αρχικά ήταν η αξιοποίηση του εργαλείου στο εσωτερικό της Nokia για την διευκόλυνση των μηχανικών της. Ωστόσο με την πάροδο του χρόνου και καθώς το Containerlab αποδεικνύονταν ιδιαίτερα χρήσιμο και εύχρηστο οι δημιουργοί του αποφάσισαν να το παρουσιάσουν στην αγορά και να το μετατρέψουν σε λογισμικό ανοιχτού κώδικα. [1, 2]

4.2 Αρχιτεκτονική και λειτουργία

Το Containerlab είναι γραμμένο στη γλώσσα προγραμματισμού Go, έχει ως βάση του την πλατφόρμα του Docker ενώ αξιοποιεί επίσης διάφορα δικτυακά εργαλεία του Linux. Διανέμεται ως πακέτο Linux deb/rpm/apk για αρχιτεκτονικές amd64 και arm64 και μπορεί να εγκατασταθεί σε οποιαδήποτε διανομή του Linux βασισμένη σε Debian ή RHEL. Μπορεί ωστόσο να εκτελεστεί και να χρησιμοποιηθεί κανονικά και σε συστήματα Windows μέσω του Windows Subsystem Linux (WSL), το οποίο επιτρέπει την εκτέλεση ενός Linux περιβάλλοντος εντός των Windows με τη μορφή μιας ελαφριάς Linux εικονικής μηχανής.
Οι προϋποθέσεις που πρέπει να τηρούνται για την επιτυχή εγκατάσταση και εκτέλεση του Containerlab είναι ο χρήστης να έχει δικαιώματα διαχειριστή, να έχει προηγηθεί εγκατάσταση του Docker.
Απαιτεί τα εξής components:

- docker (docker-ce), docker compose
- Containerlab (using the package repository)
- gh CLI tool

Τα παραπάνω μπορούν να εγκατασταθούν ταυτόχρονα χρησιμοποιώντας την ακόλουθη εντολή:

curl -sL https://containerlab.dev/setup | sudo -E bash -s "all"
Μετά την εγκατάσταση αναβάθμιση στην πιο πρόσφατη έκδοση μπορεί να γίνει με την εξής εντολή:

sudo -E containerlab version upgrade

Στη συνέχεια και προκειμένου να μπορέσει να χρησιμοποιηθεί το Containerlab απαιτείται αρχικά εγγραφή στην πλατφόρμα του Docker μέσω του εξής συνδέσμου https://app.docker.com/signup και έπειτα είσοδος από τη γραμμή εντολών με χρήση της εντολής login. Στον φάκελο /etc/containerlab/lab-examples υπάρχουν έτοιμα παραδείγματα. [27]

4.3 Αρχεία YAML

Η ανάπτυξη των δικτυακών τοπολογιών στο Containerlab γίνεται με βάση αρχεία τοπολογιών τα οποία γράφει ο χρήστης σε YAML, μια ευέλικτη, αναγνώσιμη από τον άνθρωπο γλώσσα σειριοποίησης δεδομένων που χρησιμοποιείται συνήθως για τη σύνταξη αρχείων διαμόρφωσης.

Η μορφή αυτών των αρχείων είναι αρκετά απλή και ευανάγνωστη καθώς δεν χρησιμοποιεί πολλά σύμβολα. Σημαντικό ρόλο στην ορθή σύνταξη παίζουν οι εσοχές (indentation), οι οποίες ιεραρχούν και οργανώνουν τα δεδομένα. Τα αρχεία YAML έχουν επέκταση .yml ή .yaml. Και οι δύο εκδοχές είναι εξίσου ορθές, αλλά συνιστά καλή πρακτική να προτιμάται με συνέπεια η μία από τις δύο καθολικά σε μια εργασία.

Τα δεδομένα αποθηκεύονται με τη μορφή key: value , όπου key είναι το όνομα/είδος του προσδιοριζόμενου πεδίου και value η δοθείσα τιμή (π.χ. ip: 147.102.13.200). Εάν για ένα πεδίο είναι επιθυμητό να προσδιοριστούν παραπάνω από μια παράμετροι τότε χρησιμοποιούνται ένθετα επίπεδα (nested structure) τοποθετώντας τες τη μια κάτω από την άλλη και φροντίζοντας να υφίσταται τα απαραίτητα κενά. Θα πρέπει δηλαδή να έχουν μια τέτοια δομή:

key:
parameter1: value1
parameter2: value2

Είναι δυνατή επίσης η δημιουργία λίστας όμοιων στοιχείων με την εξής μορφή:

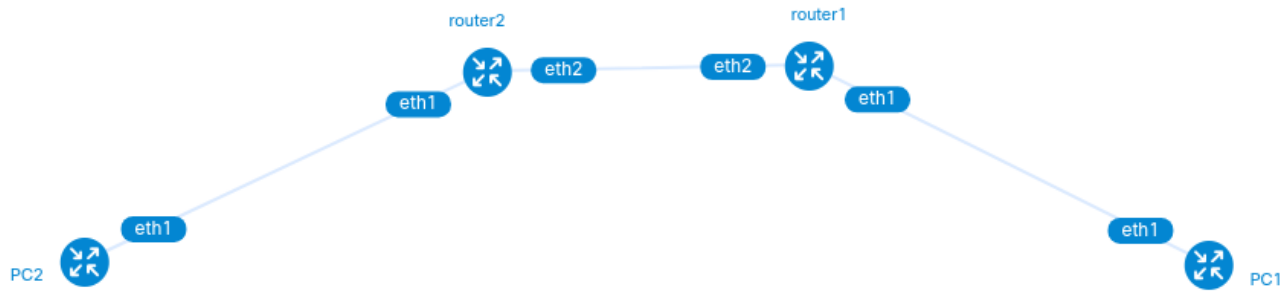
list_kind:

- ***item1***
- ***item2***

Σημειώνεται ότι ο χαρακτήρας “tab” δεν είναι αποδεκτός ενώ τυχόν σχόλια εισάγονται με χρήση του συμβόλου # στην αρχή της γραμμής.
Παρακάτω παρουσιάζεται ένα απλό αρχείο τοπολογίας και μια σχηματική αναπαράσταση της τοπολογίας που περιγράφεται, η οποία αποτελείται από δύο υπολογιστές PC1 και PC2, συνδεόμενους με δύο δρομολογητές router1, router2.

```
name: lab1

topology:
        nodes:
                PC1:
                        kind: linux
                        image: wbitt/network-multitool:latest

                PC2:
                        kind: linux
                        image: wbitt/network-multitool:latest

                router1:
                        kind: linux
                        image: quay.io/frrouting/frr:10.0.2
                        binds:
                           - config/daemons:/etc/frr/daemons
                           - config/router1.conf:/etc/frr/frr.conf

                router2:
                        kind: linux
                        image: quay.io/frrouting/frr:10.0.2
                        binds:
                           - config/daemons:/etc/frr/daemons
                           - config/router2.conf:/etc/frr/frr.conf


        links:
                - endpoints: ["PC1:eth1", "router1:eth1"]
                - endpoints: ["PC2:eth1", "router2:eth1"]
                - endpoints: ["router1:eth2", "router2:eth2"]

```

Εικόνα 2: *Ενδεικτικό αρχείο τοπολογίας YAML*



Εικόνα 3: *Σχηματική αναπαράσταση ενδεικτικής τοπολογίας*

Στο παραπάνω αρχείο με το **name** ορίζεται το όνομα της τοπολογίας (lab1). Η περιγραφή των χαρακτηριστικών της τοπολογίας γίνεται στο πεδίο **topology**, το οποίο όπως φαίνεται από τη στοίχιση περιλαμβάνει όλα τα στοιχεία που ακολουθούν. Κάτω από το **nodes** ορίζονται με τα επιθυμητά ονόματα όλοι οι κόμβοι / συσκευές της δικτυακής τοπολογίας (PC1, PC2, router1, router2). Το Containerlab ονομάζει τα containers που θα δημιουργηθούν σύμφωνα με το εξής σχήμα: clab-{{lab-name}}-{{node-name}}. Για τον κάθε κόμβο καθορίζεται στο πεδίο **kind** ο τύπος του (linux) και στο **image** δίνεται το όνομα του docker image με βάση το οποίο θα δημιουργηθεί το container (wbitt/network-multitool και quay.io/frrouting/frr) και η έκδοση του (latest και 10.0.2). Το όνομα του image θα πρέπει να δίνεται στη μορφή ***[registry]/repository[:tag]***. Παραδείγματος χάριν στο παραπάνω αρχείο wbitt → registry, network-multitool → repository , latest → tag. Εάν το image υπάρχει τοπικά τότε θα χρησιμοποιηθεί απευθείας χωρίς να κατέβει ξανά από το registry. Σε αντίθετη περίπτωση κατά την ανάπτυξη της τοπολογίας μόλις το Containerlab φτάσει σε εκείνο το σημείο του κώδικα αναζητεί αυτόματα το registry και το repository, κατεβάζει το image και συνεχίζει με τις υπόλοιπες εντολές. Αν δεν οριστεί ρητά το registry τότε το image αναζητείται στο προεπιλεγμένο registry, το Docker Hub. Για την σύνδεση αρχείων των container με αρχεία του host μηχανήματος χρησιμοποιείται ο μηχανισμός bind mount του Docker. Στο πεδίο **binds** δίνονται με τη μορφή λίστας τα ονόματα των αρχείων που ανήκουν στα container ακολουθούμενα από αυτά του host (config/daemons: /etc/frr/daemons).

Ακολουθεί ο καθορισμός των ζεύξεων μεταξύ των κόμβων στο πεδίο **links**, στο οποίο ορίζονται οι ζεύξεις χρησιμοποιώντας τις διεπαφές των κόμβων. Στο παραπάνω παράδειγμα παρατηρείται ότι έχουν οριστεί τρεις συνδέσεις από τις οποίες η πρώτη είναι μια ζεύξη ανάμεσα στην διεπαφή eth1 του router1 και την διεπαφή eth1 του PC1.

Επιπλέον δύναται να προστεθεί το πεδίο **startup-config**, στο οποίο μπορούν με ένα αρχείο .cfg να προσδιοριστούν διάφορες λεπτομέρειες για το configuration του κόμβου, όπως διεύθυνση IP, υποδίκτυο, ρυθμίσεις πρωτοκόλλων και άλλα. [15, 25]

4.4 Βασικές εντολές

Η εντολή deploy

Χρησιμοποιείται για να ξεκινήσει την ανάπτυξη (deployment) μιας δικτυακής τοπολογίας με βάση το YAML αρχείο τοπολογίας. Η εντολή αυτή αναλαμβάνει να δημιουργήσει και να διασυνδέσει τα containers που ορίζονται στο αρχείο. Παρακάτω φαίνονται οι πιο συχνές χρήσεις της εντολής, ενώ η πλήρη περιγραφή της είναι εδώ: https://containerlab.dev/cmd/deploy/

- *containerlab deploy -t [YAML_file_path]*
  Το path συνήθως είναι είτε github link(π.χ. https://github.com/hellt/clab-test-repo/), είτε το πλήρες path του επιθυμητού .yml αρχείου τοπικά (π.χ. /etc/containerlab/lab-examples/srlfrr01/srlfrr01.clab.yml). Αν ο χρήστης βρίσκεται ήδη στο directory του αρχείου τότε αρκεί το όνομα του αρχείου τοπολογίας (srlfrr01.clab.yml).

- *containerlab deploy -t [YAML_file_path] --reconfigure*
  Χρησιμοποιείται για εφαρμογή των αλλαγών, σε περίπτωση τροποποίησης του αρχείο τοπολογίας.

- *containerlab deploy -t [YAML_file_path --name [lab_name]*
  Με την παράμετρο *--name* ορίζεται νέο όνομα τοπολογίας διαφορετικό από αυτό που δόθηκε μέσα στο αρχείο YAML.

- *containerlab deploy -t [YAML_file_path] --node-filter [node1] , [node2], … , [nodeX]*
  Με την επιλογή --node-filter είναι εφικτή η επιλογή μόνο κάποιων από τους κόμβους που περιγράφονται στο αρχείο τοπολογίας για deploy και η αγνόηση των υπολοίπων.

Η εντολή destroy

Χρησιμοποιείται για να καταστρέψει/διαγράψει μια τοπολογία δικτύου, αφαιρώντας όλα τα containers, τις συνδέσεις τους και τυχόν άλλους σχετικούς πόρους που είχαν δημιουργηθεί κατά την εκτέλεση της εντολής deploy.
Παρακάτω φαίνονται οι πιο συχνές χρήσεις της εντολής, ενώ η πλήρη περιγραφή της είναι εδώ: https://containerlab.dev/cmd/destroy/

- *containerlab destroy -t [YAML_file_path]*
  Διαγράφεται η τοπολογία που έχει δημιουργηθεί με βάση το εκάστοτε αρχείο YAML.

- *containerlab destroy -t [YAML_file_path] --cleanup*
  Η προσθήκη *--cleanup* χρησιμοποιείται για να διαγραφεί ταυτόχρονα και ο φάκελος της τοπολογίας με όλα τα αρχεία.

- *containerlab destroy -a*
  Με την παράμετρο *-a* διαγράφονται όλα τα υπάρχοντα containers.

- *containerlab destroy -t [YAML_file_path] --node-filter [node1] , [node2], … , [nodeX]*

Όπως και στην εντολή deploy με την επιλογή --node-filter δύναται η διαγραφή μόνο κάποιων από τους κόμβους που περιγράφονται στο αρχείο τοπολογίας.

Η εντολή inspect

Χρησιμοποιείται για την παροχή πληροφοριών για μια υπάρχουσα τοπολογία. Παρακάτω φαίνονται οι πιο συχνές χρήσεις της εντολής, ενώ η πλήρη περιγραφή της είναι εδώ: https://containerlab.dev/cmd/inspect/

- *containerlab inspect -t [YAML_file_path]*
  Παροχή πληροφοριών για την τοπολογία που αντιστοιχεί στο YAML αρχείο που προσδιορίζεται.

- *containerlab inspect -t [YAML_file_path] --details*
  Η παράμετρος *--details* παρέχει περισσότερες λεπτομέρειες.

- *containerlab inspect -t --all*
  Με την προσθήκη *--all* παρέχονται λεπτομέρειες για όλες τις τοπολογίες.

- *containerlab inspect -t [YAML_file_path] –interfaces*
  Συνοπτική παρουσίαση των διεπαφών της τοπολογίας.

- *containerlab inspect --name [lab_name]*
  Αναζήτηση πληροφοριών για μια τοπολογία με βάση το όνομα της και όχι το αρχείο YAML.

Η εντολή exec

Επιτρέπει την εκτέλεση εντολών μέσα σε κάποιο κόμβο (container) μιας τοπολογίας. Παρακάτω φαίνονται οι πιο συχνές χρήσεις της εντολής, ενώ η πλήρη περιγραφή της είναι εδώ: https://containerlab.dev/cmd/exec/

- *containerlab exec -t [YAML_file_path] --cmd [command]*
  Εκτέλεση της εκάστοτε εντολής *[command]* σε όλους τους κόμβους που περιλαμβάνει η τοπολογία που αντιστοιχεί στο YAML αρχείο.

- *containerlab exec -t [YAML_file_path] --label [key=value] --cmd [command]*
  Με την προσθήκη της παραμέτρου *label* φιλτράρονται οι κόμβοι της τοπολογίας στους οποίους θα εκτελεστεί η εντολή. Όπου *key* το πεδίο με βάση το οποίο θα γίνει το φιλτράρισμα (node name, kind, etc) και value η τιμή των επιθυμητών κόμβων.

Η εντολή graph

Δίνει τη δυνατότητα οπτικοποίησης μιας τοπολογίας. Παρακάτω φαίνονται οι πιο συχνές χρήσεις της εντολής, ενώ η πλήρη περιγραφή της είναι εδώ: https://containerlab.dev/cmd/graph/

- *containerlab graph -t [YAML_file_path]*
  Δημιουργεί ένα γράφημα τη τοπολογίας το οποίο είναι διαθέσιμο μέσω ενός τοπικού Web Server στη διεύθυνση *http://127.0.0.1:50080*.

- *containerlab graph -t [YAML_file_path] --node-filter*
  Με την προσθήκη της παραμέτρου *--node-filter* επιλέγεται ένα υποσύνολο της τοπολογίας για γραφική αναπαράσταση.

4.5 Δικτύωση στο Containerlab

Μια από τις βασικότερες προκλήσεις στην χρήση container που αντιμετώπισε το Containerlab είναι η δημιουργία εικονικών ζεύξεων τόσο μεταξύ container και host όσο και μεταξύ των container.

Η σύνδεση container-host είναι ιδιαίτερα σημαντική και προσφέρει πολλά πρακτικά οφέλη. Δίνεται η δυνατότητα της μεταξύ τους επικοινωνίας επιτρέποντας στον host να έχει πρόσβαση στα container και να μπορεί να τα διαχειριστεί αν χρειαστεί, όπως για παράδειγμα σε περίπτωση που προκύψει κάποια δυσλειτουργία στις ρυθμίσεις της τοπολογίας. Επιπλέον μέσω του host τα container έχουν πρόσβαση στο διαδίκτυο και μπορούν να κάνουν λήψη πακέτων και ενημερώσεων.

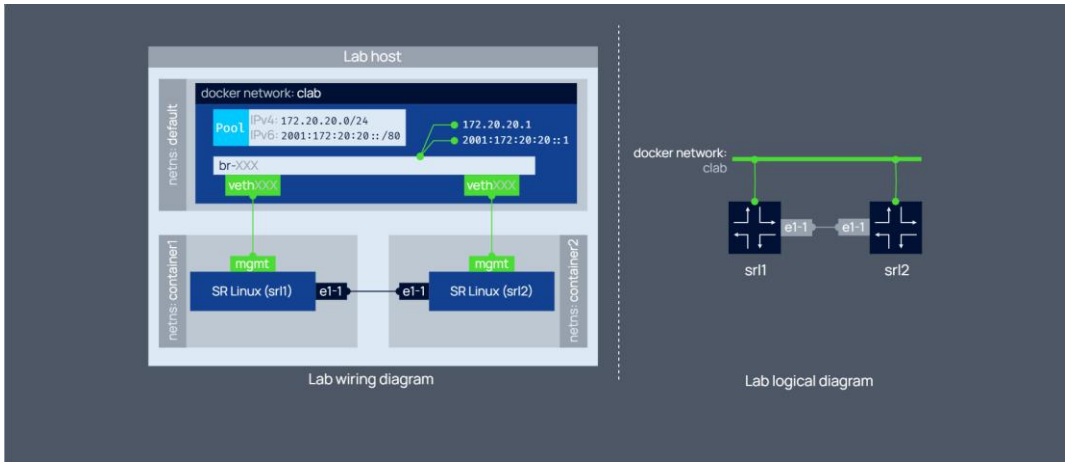
Για να επιτευχθεί η σύνδεση αυτή κάθε container κατά τη δημιουργία του αποκτά μια εικονική διεπαφή (veth) η οποία δεσμεύεται για την σύνδεση σε ένα διαχειριστικό δίκτυο, ένα docker bridge network με την ονομασία clab, βασισμένο σε μια Linux Bridge. Όλες οι εικονικές διεπαφές των container συνδέονται σε μια bridged διεπαφή του host η οποία ανήκει στο ίδιο δίκτυο και αποτελεί την προεπιλεγμένη πύλη για τα container. Η IPv4 διεύθυνση υποδικτύου για το clab είναι 172.20.20.0/24 και η πρώτη IP η 172.20.20.1 ανήκει πάντα στην bridged διεπαφή του host. Η IPv6 διεύθυνση υποδικτύου είναι η 3fff:172:20:20::/64 και η πρώτη IP που αποδίδεται στην διεπαφή του host είναι 3fff:172:20:20::1.

Το Containerlab προσφέρει και την επιλογή παραμετροποίησης του διαχειριστικού δικτύου clab. Αρκεί η προσθήκη ενός έξτρα πεδίου **mgmt** στην αρχή του YAML αρχείου της τοπολογίας ακριβώς κάτω από το όνομα αυτής (name). Εντός του mgmt μπορεί αρχικά να αλλαχτεί το όνομα του από clab σε οτιδήποτε άλλο στο πεδίο **network**. Ομοίως μπορεί να μεταβληθεί στο πεδίο **bridge** και το όνομα της linux bridge που

δημιουργείται για την υλοποίηση του διαχειριστικού δικτύου, το οποίο από προεπιλογή έχει τη μορφή br-<network-id>. Πέρα από τον ορισμό ενός νέου ονόματος, η λειτουργία αυτή μπορεί να χρησιμοποιηθεί και για να αποτραπεί η δημιουργία μιας νέας linux bridge και εναλλακτικά να επαναχρησιμοποιηθεί μια ήδη υπάρχουσα γέφυρα, η οποία διατηρεί τις IPv4, IPv6 διευθύνσεις της εάν και εφόσον της έχουν αποδοθεί. Επιπλέον είναι εφικτός ο ορισμός οποιασδήποτε διεύθυνσης υποδικτύου με χρήση των **ipv4-subnet** και **ipv6-subnet** και ο ορισμός προκαθορισμένης πύλης για το υποδίκτυο με **ipv4-gw** και **ipv6-gw**. Εάν πέρα από τον ορισμό διεύθυνσης υποδικτύου ο χρήστης επιθυμεί να αποδώσει και συγκεκριμένες διευθύνσεις στις εικονικές διεπαφές των container που συνδέονται στην bridged διεπαφή του host τότε μπορεί να το κάνει προσθέτοντας στην περιγραφή του κάθε κόμβου εντός του **node** τα πεδία **mgmt-ipv4** και **mgmt-ipv6**. Απαιτείται ωστόσο προσοχή ώστε να αποδοθούν διευθύνσεις σε όλα τα υπάρχοντα container, καθώς εάν κάποια έχουν διευθύνσεις ορισμένες από το χρήστη και κάποια λάβουν αυτόματα είναι πολύ πιθανή η σύγκρουση διευθύνσεων. Παρέχεται επίσης και η δυνατότητα προσδιορισμού του εύρους του υποδικτύου εντός των πεδίων **ipv4-range** και **ipv6-range**, σε περίπτωση που στο ίδιο διαχειριστικό δίκτυο βρίσκονται και κόμβοι που ανήκουν σε άλλες εφαρμογές και όχι στο Containerlab και πρέπει να αποφευχθεί η σύγκρουση διευθύνσεων. Στο πεδίο **mtu** ο χρήστης μπορεί να αλλάξει την προκαθορισμένη τιμή της MTU. Η αλλαγή αυτή θα οδηγήσει κάθε διεπαφή που είναι συνδεδεμένη στο διαχειριστικό δίκτυο να λάβει την ίδια τιμή MTU.

Η υλοποίηση συνδέσεων μεταξύ των container είναι απαραίτητη ώστε να είναι δυνατή η ανάπτυξη δικτυακών τοπολογιών όπως ακριβώς τις σχεδιάζει ο χρήστης. Επιτυγχάνεται με τη δημιουργία point-to-point ζεύξεων με τη χρήση “veth pairs”. Ένα veth pair προσομοιώνει στην ουσία ένα καλώδιο Ethernet με δύο άκρα. Το κάθε άκρο του ζεύγους αποτελεί διεπαφή του ενός από τα δύο συνδεόμενα container της ζεύξης και ανήκει στο network namespace του.

Τα παρακάτω διαγράμματα απεικονίζουν γραφικά τους δύο παραπάνω τρόπους συνδέσεων για μια τοπολογία αποτελούμενη από δύο κόμβους βασισμένους στο image SR Linux της Nokia.



Εικόνα 4: *Γραφική αναπαράσταση των δικτυακών συνδέσεων εντός του Containerlab*

Μετά την ανάπτυξη μιας τοπολογίας το Containerlab αποθηκεύει στατικές εγγραφές DNS στο αρχείο /etc/hosts ώστε ο χρήστης να μπορεί να έχει πρόσβαση στα container χρησιμοποιώντας τα DNS ονόματα τους. Για κάθε κόμβο αποθηκεύεται μια εγγραφή για την IPv4 του και μια για την IPv6 του, ενώ το όνομα του κόμβου έχει τη μορφή *clab-[labName]-[nodeName]*. [25]

4.6 Ροή Εκτέλεσης

Η ανάπτυξη μιας τοπολογίας γίνεται όπως προαναφέρθηκε με την εντολή deploy. Με την εκτέλεση αυτής της εντολής ακολουθείται η παρακάτω αλληλουχία βημάτων ώστε να εξασφαλιστεί η ομαλή δημιουργία της τοπολογίας και η ορθή λειτουργία της.

1. **Ανάγνωση και Επικύρωση Αρχείου Τοπολογίας**

   Το Containerlab διαβάζει το YAML αρχείο, ελέγχει τη συντακτική ορθότητα του και επιβεβαιώνει ότι όλα τα απαιτούμενα πεδία (name, topology, nodes, links) είναι παρόντα. Τυχόν σφάλματα όπως λάθος στοίχιση των πεδίων ή τυπογραφικά λάθη στην τοποθεσία του image εντοπίζονται και αναφέρονται διακόπτοντας την συνέχιση της διαδικασίας.

2. **Δημιουργία Διαχειριστικού Δικτύου**

   Εάν δεν υπάρχει ήδη, τότε δημιουργείται το διαχειριστικό δίκτυο, ένα docker bridge network, το οποίο χρησιμοποιείται με τον τρόπο που αναλύθηκε προηγουμένως για να επιτευχθεί η επικοινωνία host και container. Αν στο πεδίο mgmt του αρχείου της τοπολογίας έχουν καθοριστεί κάποιες παράμετροι τότε οι αντίστοιχες προσαρμογές στο δίκτυο γίνονται σε αυτή τη φάση.

3. **Δημιουργία Containers**

   Για κάθε κόμβο που ορίζεται στο YAML, το Containerlab δημιουργεί με βάση την περιγραφή του ένα container με όνομα της μορφής *clab-[lab-name]-[node-name]*. Σε αυτό το στάδιο εκτελούνται και ενδεχόμενα bind-mounts, αν έχουν οριστεί.

4. **Δημιουργία Συνδέσεων**

   Δημιουργούνται ζεύγη εικονικών διεπαφών (**veth pairs**) ώστε να συνδεθούν τα container όπως ακριβώς ορίζει το πεδίο links του αρχείο τοπολογίας.

5. **Εφαρμογή Παραμετροποίησης**

   Οποιαδήποτε παραμετροποίηση έχει καθοριστεί στο αρχείο τοπολογίας για τα containers εφαρμόζεται τώρα. Αντιγράφονται επίσης εντός των κόμβων αυτόματες παραμετροποιήσεις μέσω αρχείων .cfg.

6. **Δημιουργία DNS Εγγραφών**

   Το */etc/hosts* του host ενημερώνεται με στατικές εγγραφές DNS για όλους τους καινούριους κόμβους ώστε να είναι προσβάσιμοι με ονόματα στη μορφή *clab-[labName]-[nodeName]*.

Αφού ολοκληρωθεί επιτυχώς η παραπάνω διαδικασία η τοπολογία έχει υλοποιηθεί και πλέον ο χρήστης μπορεί να συνδεθεί στους κόμβους της είτε με ssh χρησιμοποιώντας το όνομα του container ή την IP του στο διαχειριστικό δίκτυο, είτε να συνδεθεί στο cli ή στο shell του με τις εξής εντολές:

docker exec -it *clab-[labName]-[nodeName] [nodeCli]*

docker exec -it *clab-[labName]-[nodeName] [shell_command]*

όπου nodeCli το Cli του κάθε image (για Nokia SR Linux → sr_cli , για FRR → vtysh) και όπου shell_command → bash ή sh .

Αντίστοιχα η εντολή destroy ακολουθεί την αντίθετη διαδικασία διαγράφοντας διεπαφές, containers, συνδέσεις και εγγραφές DNS. [25]

5. Θεωρητικό υπόβαθρο τοπολογιών

Στο παρόν κεφάλαιο παρουσιάζονται τα βασικότερα πρωτόκολλα δρομολόγησης (RIP, OSPF, BGP) και οι τεχνολογίες διασύνδεσης (ECMP, VXLAN, Linux Bridges, Open vSwitch) που αποτελούν τη βάση των τοπολογιών οι οποίες υλοποιήθηκαν στο Containerlab στα πλαίσια της διπλωματικής εργασίας και θα παρουσιαστούν σε επόμενο κεφάλαιο. Η θεωρητική αυτή ανάλυση κρίνεται απαραίτητη για σαφήνεια και πληρότητα σχετικά με τις δυνατότητες, τα πλεονεκτήματα αλλά και τους περιορισμούς κάθε συστήματος δρομολόγησης και διασύνδεσης δικτύου, καθώς και τον τρόπο με τον οποίο αυτά συνδυάζονται για την κατασκευή σύγχρονων και αποδοτικών δικτύων.

Η προσέγγιση που ακολουθείται είναι ιεραρχική: αρχικά εξετάζονται τα παλαιότερα πρωτόκολλα (RIP), στη συνέχεια τα πιο σύγχρονα (OSPF, BGP) και τέλος οι τεχνολογίες που επιτρέπουν την κλιμάκωση και ευελιξία στα δίκτυα (ECMP, VXLAN, EVPN, OVS).

5.1 RIP

Το Routing Information Protocol (RIP) είναι ένα από τα παλαιότερα πρωτόκολλα εσωτερικής δρομολόγησης διανύσματος απόστασης (distance-vector routing protocols). Βασίζεται στον αλγόριθμο διανύσματος απόστασης (distance vector algorithm), ο οποίος εντοπίζει την βέλτιστη διαδρομή μεταξύ δύο κόμβων, δηλαδή την διαδρομή με το ελάχιστο κόστος. Ως μετρικό (metric) για τον καθορισμό της βέλτιστης διαδρομής στο RIP χρησιμοποιείται ο αριθμός των βημάτων (hop count) από τον έναν κόμβο στον άλλο. Ο αριθμός αυτός αντιπροσωπεύει πρακτικά τον αριθμό των ενδιάμεσων συσκευών από τις οποίες διέρχονται τα πάντα μέχρι να φτάσουν από την πηγή στον προορισμό τους. Μέσω αυτού του μετρικού είναι εφικτή η σύγκριση εναλλακτικών διαδρομών και η άμεση εύρεση της καλύτερης εξ' αυτών, η οποία επιλέγεται και εισάγεται στον πίνακα δρομολόγησης. Αξίζει ωστόσο να σημειωθεί ότι η επιλεγμένη διαδρομή στο RIP είναι με βεβαιότητα αυτή με τη μικρότερη απόσταση μεταξύ των κόμβων αλλά όχι απαραίτητα η γρηγορότερη, καθώς άλλες παράμετροι όπως το εύρος ζώνης δεν λαμβάνονται υπόψη.

Η διαχειριστική απόσταση του πρωτοκόλλου RIP είναι 120. Το μετρικό αυτό αποτελεί ένα κριτήριο για την επιλογή διαδρομών που χρησιμοποιούν διαφορετικά πρωτόκολλα. [53] Αφορά την ποιότητα και την αξιοπιστία της εγγραφής.
Ένας από τους βασικότερους αρχιτεκτονικούς περιορισμούς του RIP είναι ότι ο μέγιστος επιτρεπόμενος αριθμός βημάτων είναι 15, δηλαδή μπορεί να γίνει μετάδοση δεδομένων μέσω 16 κόμβων το ανώτερο. Αυτό σημαίνει ότι οποιαδήποτε διαδρομή απαιτεί από 16 βήματα και πάνω θεωρείται ως άπειρη απόσταση και συνεπώς μη

προσβάσιμη. Με τον τρόπο αυτό αποφεύγονται ωστόσο οι σχηματισμοί βρόχων και η επ' αόριστόν κυκλοφορία πακέτων εντός του δικτύου.

Το RIP χρησιμοποιεί το πρωτόκολλο UDP στο στρώμα μεταφοράς κατά την αποστολή δεδομένων και τη θύρα προορισμού 520. Περιλαμβάνει έναν μηχανισμό περιοδικής ενημέρωσης για τη διάδοση πληροφοριών δρομολόγησης σε όλο το δίκτυο. Κάθε 30 δευτερόλεπτα, οι δρομολογητές εντός του δικτύου με δυνατότητα RIP μεταδίδουν ενημερώσεις που περιέχουν τους πίνακες δρομολόγησής τους. Μέσω αυτών των ενημερώσεων οι γειτονικοί δρομολογητές λαμβάνουν πληροφορίες για την τρέχουσα κατάσταση του δικτύου, συμπεριλαμβανομένου του απαιτούμενου αριθμού των βημάτων για τον κάθε προορισμό. Σε περίπτωση που παρέλθει διάστημα 180 δευτερολέπτων χωρίς ενημέρωση για κάποιον προορισμό τότε αυτός θεωρείται μη προσβάσιμος ενώ μετά από άλλα 60 δευτερόλεπτα (240 από την τελευταία ενημέρωση) η εγγραφή διαγράφεται από τους πίνακες δρομολόγησης. Διατηρούνται επίσης εγγραφές για εναλλακτικές διαδρομές με μεγαλύτερο κόστος από την επιλεγμένη και σε περίπτωση αδυναμίας χρήσης της επιλεγμένης τοποθετούνται στον πίνακα δρομολόγησης ως κύριες πλέον. Οποιαδήποτε τέτοια αλλαγή κοινοποιείται σε όλους τους γείτονες σταδιακά. Αυτή η περιοδική ανταλλαγή πληροφοριών διατηρεί συγχρονισμένους και ενημερωμένους πίνακες δρομολόγησης εξασφαλίζοντας την συνοχή του δικτύου και την αποτελεσματική ανταπόκριση σε αλλαγές στην τοπολογία, όπως προσθήκη η αφαίρεση δρομολογητών και μεταβολές στην συνδεσιμότητα αυτών.

Υπάρχουν τρεις εκδόσεις του RIP, τα RIPv1, RIPv2 και RIPng. Το RIPv1 χρησιμοποιεί ταξικές διευθύνσεις και εκπομπή (broadcast) ενώ δεν υποστηρίζει VLSM (Variable Length Subnet Mask) και πιστοποίηση αυθεντικότητας. Το RIPv2 αποτελεί βελτίωση του RIPv1, σε αντίθεση με το οποίο υποστηρίζει VLSM και επιτρέπει την πιστοποίηση αυθεντικότητας ενώ χρησιμοποιεί αταξικές διευθύνσεις και πολλαπλή διανομή (multicast). Το RIPng σημαίνει RIP επόμενης γενιάς και είναι μια επέκταση του RIPv2 που υποστηρίζει IPv6.

Το RIP είναι ένα πρωτόκολλο εύκολο στη διαμόρφωση και με ελάχιστη χρήση CPU, κατάλληλο για μικρά δίκτυα. Η ελαχιστοποίηση του αριθμού των βημάτων για την επιλογή της καλύτερης διαδρομής δίνει προτεραιότητα στην απλότητα και την ευκολία υλοποίησης, ειδικά σε σενάρια όπου οι διακυμάνσεις του εύρους ζώνης δεν αποτελούν κρίσιμη παράμετρο. Παρά τα οφέλη αυτά η χρήση του συνοδεύεται από σημαντικούς περιορισμούς. Ένας από αυτούς αφορά την επεκτασιμότητα και τον χρόνο σύγκλισης. Καθώς το μέγεθος του δικτύου αυξάνεται, οι περιοδικές ενημερώσεις του RIP και η εξάρτηση από τον αριθμό βημάτων ενδέχεται να εισάγουν καθυστερήσεις στην προσαρμογή του δικτύου σε μεταβολές. Σε μεγαλύτερα δίκτυα προτιμώνται πιο εξελιγμένα πρωτόκολλα δρομολόγησης σχεδιασμένα για να χειρίζονται πολύπλοκες τοπολογίες και να παρέχουν ταχύτερους χρόνους σύγκλισης. Επιπλέον η συνεχής σύνδεση με γειτονικούς δρομολογητές για ενημέρωση των πινάκων δρομολόγησης, δημιουργεί μεγάλη ποσότητα κίνησης επιβαρύνοντας το

δίκτυο. Ένας ακόμη αξιοσημείωτος κίνδυνος είναι η δημιουργία βρόχων δρομολόγησης, στους οποίους παράγονται συνεχώς ενημερώσεις για τις ίδιες διαδρομές, με συνεχώς αυξανόμενη απόσταση. Το αποτέλεσμα είναι να μεταδίδονται στο δίκτυο λανθασμένες πληροφορίες, οδηγώντας σε ανεπάρκειες ή διακοπή της ομαλής λειτουργίας. Για τον μετριασμό αυτού του κινδύνου, το RIP χρησιμοποιεί διάφορες τεχνικές όπως ο μηχανισμός του διαχωρισμένου ορίζοντα (split-horizon), η δηλητηρίαση διαδρομής (route poisoning), holddown timers αλλά και όπως προαναφέρθηκε ο καθορισμός μέγιστου επιτρεπόμενου αριθμού βημάτων που ανέρχεται σε 15. Ο διαχωρισμός ορίζοντα εμποδίζει τους δρομολογητές να διαφημίζουν μια διαδρομή πίσω στον δρομολογητή από τον οποίο έλαβαν αρχικά την πληροφορία για αυτή τη διαδρομή. Η δηλητηρίαση διαδρομής περιλαμβάνει τη σήμανση μη προσβάσιμων διαδρομών με άπειρη απόσταση (16) ώστε να υποδειχθεί ως μη διαθέσιμη και να θεωρηθεί άκυρη. Ένα holddown timer διαρκεί από προεπιλογή 180 δευτερόλεπτα στο RIP και κατά τη διάρκεια αυτή αγνοείται οποιαδήποτε ενημέρωση για μια μη προσβάσιμη διαδρομή εάν δεν προέρχεται από τον δρομολογητή που διαφήμισε αρχικά τη διαδρομή. Εάν ο δρομολογητής αυτός διαφημίσει ξανά τη διαδρομή το holddown timer σταματά και η πληροφορία γίνεται δεκτή.

[47, 48, 49, 50, 51, 53, 82]

5.2 OSPF

Το Open Shortest Path First (OSPF) είναι ένα πρωτόκολλο εσωτερικής δρομολόγησης (Interior Gateway Protocol – IGP) βασισμένο στην κατάσταση σύνδεσης (link-state). Η λειτουργία του βασίζεται στην χρήση του αλγορίθμου ελάχιστης διαδρομής (Shortest Path First – SPF) του Dijkstra για την εύρεση της καλύτερης διαδρομής και την προσθήκη της στον πίνακα δρομολόγησης. Η μετρική που χρησιμοποιείται από τον αλγόριθμο αυτόν για την επιλογή διαδρομής είναι το κόστος της ζεύξης. Ο αριθμός αυτός είτε προσδιορίζεται χειροκίνητα είτε υπολογίζεται αυτόματα από το OSPF σύμφωνα με τον τύπο:

$$\frac{Reference\ Bandwidth}{Interface\ Bandwidth}$$

Η προεπιλεγμένη τιμή εύρους ζώνης αναφοράς (Reference Bandwidth) είναι 100 Mbps αλλά μπορεί να αλλαχθεί από τον χρήστη. Η ελάχιστη τιμή που μπορεί να λάβει το κόστος μιας ζεύξης είναι 1. Η διαδρομή που επιλέγεται είναι αυτή με το χαμηλότερο κόστος.

Μεταξύ των δρομολογητών μιας τοπολογίας με OSPF γίνεται ανταλλαγή πληροφοριών για την κατάσταση των ζεύξεων με στόχο όλοι οι δρομολογητές τελικά

να έχουν γνώση για το σύνολο της τοπολογίας. Για να συμβούν οι παραπάνω ανταλλαγές πληροφοριών δημιουργούνται σχέσεις γειτνίασης μεταξύ των δρομολογητών. Η διαδικασία αυτή ξεκινά με την αποστολή από τον έναν δρομολογητή Hello πακέτων τα οποία θα περιλαμβάνουν το Router ID του εν δυνάμει γείτονα. Μόλις αυτός λάβει τα Hello πακέτα ακολουθεί ο έλεγχος ορισμένων παραμέτρων, όπως υποδίκτυο, MTU, Area ID, είδος περιοχής, Hello Interval, Dead Interval, ώστε να εξακριβωθεί ότι ταυτίζονται οι τιμές τους και στους δύο γείτονες. Αφού επιβεβαιωθούν τα παραπάνω ξεκινά η αμφίδρομη επικοινωνία των δύο δρομολογητών και θεωρούνται πλέον γείτονες. Στο OSPF τα δεδομένα ενθυλακώνονται κατευθείαν στα IP πακέτα IP με αριθμό πρωτοκόλλου 89 χωρίς να χρησιμοποιείται κάποιο πρωτόκολλο μεταφοράς (UDP, TCP).

Κάθε δρομολογητής αποθηκεύει πληροφορίες για την τοπολογία και την δρομολόγηση σε τρεις πίνακες:

- Πίνακας γειτόνων → Περιλαμβάνει τους γείτονες του δρομολογητή και πληροφορίες για αυτούς.
- Πίνακας τοπολογίας → Περιλαμβάνει πληροφορίες για την δομή της τοπολογίας και όλες τις διαθέσιμες διαδρομές μεταξύ των δρομολογητών της περιοχής του.
- Πίνακας δρομολόγησης → Σε αυτόν αποθηκεύονται οι τρέχουσες επιλεγμένες διαδρομές.

Το OSPF ομαδοποιεί τους δρομολογητές σε περιοχές (areas). Όλοι οι δρομολογητές μιας περιοχής έχουν τον ίδιο πίνακα τοπολογίας, αλλά δεν έχουν πληροφορίες για τους δρομολογητές άλλων περιοχών. Με την τεχνική αυτή απαιτείται λιγότερος χρόνος για την εκτέλεση του αλγορίθμου SPF και λιγότερες ενημερώσεις δρομολόγησης. Κάθε περιοχή έχει ένα αναγνωριστικό (area ID) εκφρασμένο είτε σε μορφή IP διεύθυνσης είτε σε δεκαδική μορφή. Για παράδειγμα το ID της περιοχής μηδέν συμβολίζεται ισοδύναμα είτε ως 0 είτε ως 0.0.0.0. Ειδικότερα η περιοχή 0 (area 0) ονομάζεται περιοχή κορμού (backbone area) και κάθε άλλη περιοχή στο δίκτυο OSPF πρέπει να συνδέεται με αυτή. Οι περιοχές διακρίνονται σε κανονικές (standard) και απολήξεις (stub). Οι κανονικές περιοχές συμπεριλαμβάνουν στους πίνακες τους εγγραφές όχι μόνο για εσωτερικές διαδρομές και διαδρομές μεταξύ περιοχών OSPF αλλά και για εξωτερικές εκτός OSFP διαδρομές. Αντίθετα ο πίνακας δρομολόγησης μια περιοχής απόληξης περιέχει μεν όλες τις εσωτερικές διαδρομές για το δίκτυο OSPF αλλά μόνο μία προκαθορισμένη διαδρομή για όλους τους εκτός δικτύου OSPF προορισμούς. Μια υποκατηγορία των περιοχών απολήξεων είναι οι περιοχές πλήρους απόληξης (totally stubby area), οι οποίες διαθέτουν εγγραφές για διαδρομές εντός της περιοχής και μια προκαθορισμένη διαδρομή για όλους τους εκτός περιοχής προορισμούς.

Οι δρομολογητές στο OSPF ταξινομούνται στις εξής κατηγορίες ανάλογα με τη θέση τους στην τοπολογία:

- Εσωτερικός Δρομολογητής (Internal Router – IR) → Έχει μόνο γείτονες που ανήκουν στην ίδια περιοχή με αυτόν.
- Δρομολογητής Κορμού (Backbone Router – BR) → Περιλαμβάνει τουλάχιστον μια διεπαφή στην περιοχή κορμού.
- Area Border Router (ABR) → Συνδέει μια η περισσότερες περιοχές με την περιοχή κορμού.
- Autonomous System Boundary Router (ASBR) → Συνδέει μια ή περισσότερες περιοχές με ένα δίκτυο εξωτερικό του OSPF, το οποίο χρησιμοποιεί άλλο πρωτόκολλο δρομολόγησης ή στατικές διαδρομές.
- Designated Router (DR) → Αποτελεί την πηγή διάδοσης των ενημερώσεων και των πληροφοριών σε μια περιοχή. Όλοι οι άλλοι δρομολογητές της περιοχής στέλνουν τις ενημερώσεις τους σε αυτόν και αυτός τις προωθεί κατάλληλα. Ο ρόλος αυτός ανατίθεται στον δρομολογητή με την μεγαλύτερη προτεραιότητα (Router Priority) και σε περίπτωση ισοψηφίας σε αυτόν με το μεγαλύτερο Router ID.
- Backup Designated Router (BDR) → Αναλαμβάνει τη θέση του DR σε περίπτωση αδυναμίας αυτού να ανταποκριθεί. Είναι αυτός με την δεύτερη μεγαλύτερη προτεραιότητα ή Router ID.

Μεταξύ των δρομολογητών ανταλλάσσονται διαφημίσεις LSA (Link State Advertisement), οι οποίες περιγράφουν τις συνδέσεις και τα κόστη τους. Οι πληροφορίες αυτές αποθηκεύονται στην LSDB (Link State DataBase), μια βάση δεδομένων ίδια για όλους τους δρομολογητές μιας περιοχής. Υπάρχουν διάφοροι τύποι LSA. Οι σημαντικότεροι και πιο χρήσιμοι είναι οι ακόλουθοι:

- Type 1 – Router LSAs → Περιγράφει έναν δρομολογητή σε μια περιοχή και μεταδίδεται από κάθε δρομολογητή μόνο στην περιοχή του.
- Type 2 – Network LSAs → Περιγράφει τη λίστα των δρομολογητών που είναι συνδεδεμένοι σε ένα δίκτυο και μεταδίδεται από τον DR μόνο στην περιοχή του.
- Type 3 – Summary LSAs → Περιγράφει διαδρομές προς δίκτυα σε άλλη περιοχή OSPF του ίδιου Αυτόνομου Συστήματος (AS). Διαδίδεται από τον ABR μέσω της περιοχής κορμού σε άλλες περιοχές.
- Type 4 – ASBR Summary LSAs → Περιγράφει έναν δρομολογητή ASBR. Διαδίδεται από τον ABR μιας περιοχής που περιέχει σε άλλες περιοχές.
- Type 5 – AS external LSAs → Περιγράφει μια εξωτερική (εκτός δικτύου OSPF) διαδρομή. Διαδίδεται από τον ASBR σε όλες τις περιοχές του OSPF δικτύου.

Το OSPF αποτελεί ένα από τα πιο ισχυρά πρωτόκολλα δρομολόγησης, λόγω της γρήγορης σύγκλισης, η οποία προκύπτει από την άμεση ενημέρωση για οποιαδήποτε

αλλαγή στην τοπολογία του δικτύου και συνεπώς την εξασφάλιση ταχείας επαναφοράς διαδρομών χωρίς βρόχους. Υποστηρίζει μεγάλα και σύνθετα δίκτυα, προσφέροντας ουσιαστικά απεριόριστη διάμετρο, ενώ η επεκτασιμότητά του το καθιστά ιδανικό για συνεχώς αναπτυσσόμενα περιβάλλοντα. Επιπλέον, παρέχει αποτελεσματική και αξιόπιστη μεταφορά ενημερώσεων δρομολόγησης με χαμηλή κατανάλωση πόρων, καθώς αποστέλλει μόνο τις απαραίτητες πληροφορίες για αλλαγές. Το OSPF διαθέτει δυνατότητες όπως ισοκατανομή κίνησης σε πολλαπλές διαδρομές, δρομολόγηση ανάλογα με τον τύπο υπηρεσίας, καθώς και μηχανισμούς ελέγχου ταυτότητας για μεγαλύτερη ασφάλεια. Για τον λόγο αυτό χρησιμοποιείται εκτενώς σε μεγάλα εταιρικά και πολυσύνθετα δίκτυα, ενώ σε σύγκριση με το RIP επιτυγχάνει καλύτερη αξιοποίηση συνδέσεων, υψηλότερη απόδοση και μικρότερο χρόνο καθυστέρησης.

Παρά τα σημαντικά του πλεονεκτήματα, το OSPF παρουσιάζει αυξημένη πολυπλοκότητα στη ρύθμιση και διαχείρισή του και απαιτεί εξειδικευμένες γνώσεις, γεγονός που το καθιστά λιγότερο κατάλληλο για μικρά ή απλά δίκτυα, όπου ένα πιο ελαφρύ πρωτόκολλο, όπως το RIP, μπορεί να καλύψει τις ανάγκες με μικρότερο κόστος διαχείρισης. Επιπλέον, λόγω της λεπτομερούς χαρτογράφησης της τοπολογίας, μπορεί να επιβαρύνει τη μνήμη και την επεξεργαστική ισχύ των δρομολογητών, ειδικά σε μεγάλα δίκτυα με πολλούς κόμβους. Η διαδικασία συγχρονισμού και ανταλλαγής LSAs (Link-State Advertisements) ενδέχεται να δημιουργήσει προσωρινά αυξημένο φόρτο σε περιπτώσεις μεγάλων αλλαγών.

[48, 49, 50, 51, 54, 55, 56, 62]

5.3 BGP

Το Border Gateway Protocol (BGP) είναι ένα πρωτόκολλο εξωτερικής δρομολόγησης μεταξύ αυτόνομων συστημάτων (Inter-Autonomous System Routing Protocol). Πρόκειται για ένα πρωτόκολλο διανύσματος διαδρομών (Path-Vector Routing Protocol). Σε αντίθεση με τα πρωτόκολλα δρομολόγησης διανύσματος απόστασης (distance-vector routing protocols) και τα πρωτόκολλα κατάστασης σύνδεσης (link-state), ένα πρωτόκολλο διανύσματος διαδρομών αποθηκεύει διανύσματα με τις διαδρομές προς τους διάφορους προορισμούς, συμπεριλαμβάνοντας χαρακτηριστικά των διαδρομών αυτών, όπως το δίκτυο προορισμού και τα αυτόνομα σύστημα από τα οποία διέρχεται το πακέτο στη διάρκεια της διαδρομής. [44]

Ένα Αυτόνομο Σύστημα (Autonomous System – AS) είναι ένα σύνολο δικτύων και δρομολογητών που λειτουργούν υπό κοινή διαχείριση και έχουν ενιαία πολιτική δρομολόγησης. Εντός του αυτόνομου συστήματος, μπορεί να χρησιμοποιούνται ένα ή περισσότερα πρωτόκολλα εσωτερικής δρομολόγησης (IGP) και διαφορετικές μετρικές για την επιλογή βέλτιστων διαδρομών, ωστόσο προς τα άλλα Αυτόνομα

Συστήματα παρουσιάζεται ως μια ενιαία και συνεκτική οντότητα. Κάθε AS χαρακτηρίζεται από έναν μοναδικό ακέραιο αριθμό ASN (AS Number). [46]

Ο ρόλος του BGP είναι να εξασφαλίζει την επικοινωνία μεταξύ των διαφορετικών αυτόνομων συστημάτων. Προκειμένου να επιτευχθεί αυτή η ανταλλαγή πληροφοριών μεταξύ AS εγκαθίστανται συνδέσεις οι οποίες χαρακτηρίζονται ως σύνοδοι BGP (BGP sessions) και ταξινομούνται σε δύο κατηγορίες:

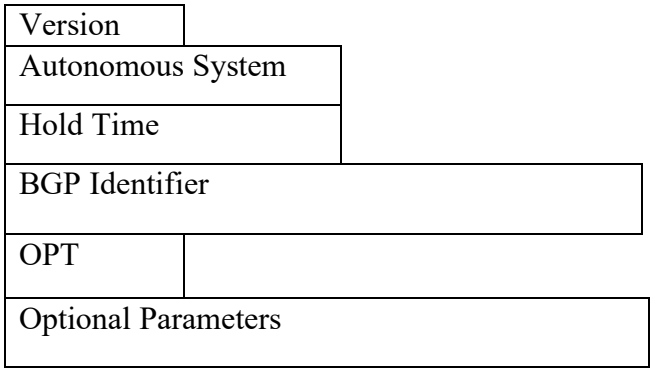
- internal BGP (iBGP) → Τα μέλη της συνόδου BGP (συνομιλητές - BGP speakers) ανήκουν στο ίδιο AS και δεν είναι απαραίτητα απευθείας συνδεδεμένα.
- external BGP (eBGP) → Τα μέλη της συνόδου BGP ανήκουν σε διαφορετικά AS και είναι απευθείας συνδεδεμένα.

Στις συνόδους BGP χρησιμοποιείται ως πρωτόκολλο μεταφοράς το TCP και ως θύρα προορισμού η θύρα 179. Αφού σχηματιστεί η σύνδεση TCP απομένει να επιβεβαιωθούν κάποιες παράμετροι ώστε να εγκατασταθεί σχέση γειτνίασης μεταξύ των δρομολογητών (BGP Neighbors). Έπειτα γίνεται ανταλλαγή των πινάκων δρομολόγησης. Ο κάθε δρομολογητής διατηρεί έναν πίνακα γειτόνων στους οποίους αναγγέλλει τις καλύτερες διαδρομές που έχει επιλέξει από τον πίνακα διαδρομών του. Οι γείτονες με τη σειρά τους λαμβάνουν τις διαδρομές αυτές, τις αξιολογούν συγκρίνοντας τες με αυτές που έχουν οι ίδιοι στον πίνακα διαδρομών τους και εάν κάποια από τις νέες υπερτερεί τότε την εισάγουν στον πίνακα δρομολόγησης τους. Η αξιολόγηση μιας διαδρομής και η επιλογή της καλύτερης γίνεται με βάση μια σειρά κριτήριων, ιεραρχημένη από το ισχυρότερο στο πιο αδύναμο. Το πρώτο κριτήριο που δεν οδηγεί σε ισοβαθμία διαδρομών καθορίζει την καλύτερη διαδρομή. Τα βασικότερα από αυτά είναι τα παρακάτω:

- Βάρος – WEIGHT → Αποτελεί τοπική παράμετρο του δρομολογητή και η διαδρομή με το υψηλότερο βάρος υπερισχύει. Διαδρομές με πηγή τον ίδιο τον δρομολογητή έχουν προκαθορισμένη τιμή 32768 και όλες οι άλλες έχουν βάρος 0.
- Τοπική Προτίμηση – LOCAL_PREF → Δηλώνει την τοπική προτίμηση του δρομολογητή για μια συγκεκριμένη διαδρομή. Όσο υψηλότερη η τιμή της τόσο καλύτερη θεωρείται η διαδρομή.
- Προτιμώνται διαδρομές που ορίστηκαν τοπικά εντός του AS έναντι διαδρομών που έγιναν γνωστές μέσω eBGP.
- Μήκος Διαδρομής – AS_PATH → Επιλέγεται η συντομότερη διαδρομή με τον λιγότερο αριθμό βημάτων, όπου ως βήματα θεωρούνται τα διαφορετικά AS που θα πρέπει να διασχίσει το πακέτο, μετρούμενα από τα ASN τους.

Όπως προαναφέρθηκε μεταξύ των BGP δρομολογητών ανταλλάσσονται μηνύματα κατά τη διάρκεια TCP συνδέσεων. Τα μηνύματα αυτά διαχωρίζονται σε 4 κατηγορίες:

- **OPEN** → Είναι το πρώτο μήνυμα που στέλνεται και από τις δύο πλευρές αμέσως μετά την δημιουργία της σύνδεσης TCP. Μέσω αυτού γίνεται ο καθορισμός των απαραίτητων παραμέτρων και η εγκατάσταση της συνόδου BGP. Η σχηματική του αναπαράσταση είναι η εξής:



Εικόνα 5: *Δομή BGP OPEN μηνύματος*

Version: Καταλαμβάνει 1 byte και υποδεικνύει τον αριθμό έκδοσης πρωτοκόλλου του μηνύματος

Autonomous System: Καταλαμβάνει 2 bytes και υποδεικνύει τον αριθμό Αυτόνομου Συστήματος του αποστολέα (ASN).

Hold Time: Καταλαμβάνει 2 bytes και δηλώνει τον προτεινόμενο από τον αποστολέα αριθμό δευτερολέπτων για το χρονοδιακόπτη αναμονής (hold timer). Το hold timer αντιπροσωπεύει τον μέγιστο χρόνο αναμονής χωρίς ανανέωση μια συνόδου BGP μέχρι αυτή να θεωρηθεί μη διαθέσιμη. Πρέπει να οριστεί σε μηδέν ή τουλάχιστον 3 δευτερολέπτα.

BGP Identifier: Καταλαμβάνει 4 bytes και αποτελεί το αναγνωριστικό του αποστολέα. Η τιμή του ισούται με μια διεύθυνση IP που ανήκει στον αποστολέα.

OPT: Καταλαμβάνει 1 byte και υποδεικνύει το συνολικό μήκος του πεδίου Optional Parameters (Προαιρετικές Παράμετροι) σε byte. Εάν η τιμή αυτού του είναι μηδέν, δεν υπάρχουν προαιρετικές παράμετροι.

Optional Parameters: Περιλαμβάνει μια λίστα προαιρετικών παραμέτρων για την BGP σύνοδο. Η κάθε παράμετρος κωδικοποιείται με τη μορφή <Τύπος Παραμέτρου, Μήκος Παραμέτρου, Τιμή Παραμέτρου>.

- **UPDATE** → Αποτελούν τα βασικότερα BGP μηνύματα καθώς μέσω αυτών συντελείται η μεταφορά πληροφοριών δρομολόγησης μεταξύ των γειτόνων. Χρησιμοποιούνται για τη ανακοίνωση διαθέσιμων διαδρομών και τον διαμοιρασμό των χαρακτηριστικών τους ή την απόσυρση μη διαθέσιμων, λειτουργίες που μπορούν να εκτελεστούν και ταυτόχρονα σε ένα μήνυμα. Έχει την ακόλουθη δομή:

Withdrawn Routes Length	
Withdrawn Routes (variable)	
Total Path Attribute Length	
Path Attributes (variable)	
Network Layer Reachability Information (variable)	

Εικόνα 6: *Δομή BGP UPDATE μηνύματος*

Withdrawn Routes Length: Έχει μήκος 2 bytes και αποτελεί το μήκος του πεδίου Withdrawn Routes (Αποσυρόμενες Διαδρομές). Εάν η τιμή του είναι μηδενική τότε σημαίνει δεν αποσύρεται καμία διαδρομή και το πεδίο Withdrawn Routes απουσιάζει.

Withdrawn Routes (variable): Πρόκειται για μεταβλητό πεδίο, το οποίο περιλαμβάνει μια λίστα με τα προθέματα δικτύου που δεν είναι πλέον διαθέσιμα και αποσύρονται.

Total Path Attribute Length: Έχει μήκος 2 bytes και αποτελεί το μήκος του πεδίου Path Attributes. Εάν η τιμή του είναι μηδενική τότε σημαίνει ότι τόσο το πεδίο Path Attributes όσο και το πεδίο Network Layer Reachability Information απουσιάζει.

Path Attributes (variable): Αποτελεί επίσης μεταβλητό πεδίο το οποίο περιέχει τα χαρακτηριστικά κάθε διαδρομής που ανακοινώνεται στο μήνυμα.

Network Layer Reachability Information (variable): Μεταβλητό πεδίο με περιεχόμενο μια λίστα προθεμάτων διευθύνσεων IP τα οποία αντιπροσωπεύουν τους διαφορετικούς προορισμούς που ανακοινώνονται.

- **KEEPALIVE** → Χρησιμοποιούνται για να επιβεβαιώσουν ότι η σύνδεση μεταξύ των BGP δρομολογητών παραμένει ενεργή. Στέλνονται σε τακτά χρονικά διαστήματα όταν δεν υπάρχουν άλλες ενημερώσεις, ώστε να αποτραπεί η λήξη του Hold Timer και η ακύρωση της συνόδου λόγω αδράνειας. Συνήθως στέλνονται κάθε 60 sec, διάστημα ίσο με το 1/3 της διάρκειας του Hold Timer. Το μήκος του πεδίου είναι 19 byte και περιλαμβάνει μόνο την επικεφαλίδα BGP.

- **NOTIFICATION** → Αποστέλλονται όταν προκύψει κάποιο σφάλμα στη σύνοδο BGP. Περιλαμβάνουν πληροφορίες για το είδος του σφάλματος και μόλις αποσταλούν τερματίζεται η σύνοδος. Στο παρακάτω σχήμα φαίνεται η μορφή αυτών των μηνυμάτων:

Error Code
Error Subcode
Data (variable)

Εικόνα 7: *Δομή BGP NOTIFICATION μηνύματος*

Error Code: Πεδίο μήκους 1 byte που περιγράφει το είδος του NOTIFICATION μηνύματος, σύμφωνα με τον ακόλουθο πίνακα:

Error Code	NOTIFICATION Type
1	Message Header Error
2	OPEN Message Error
3	UPDATE Message Error
4	Hold Timer Expired
5	Finite State Machine Error
6	Cease

Εικόνα 8: *Είδη μηνυμάτων NOTIFICATION*

Error Subcode: Πεδίο μήκους 1 byte το οποίο περιγράφει πιο συγκεκριμένα το σφάλμα που εντοπιστήκε.

Data (variable): Μεταβλητού μήκους πεδίο, το οποίο χρησιμοποιείται για αναγνώριση των αιτιών που προκάλεσαν την αποστολή του NOTIFICATION μηνύματος.

Το BGP είναι το βασικό πρωτόκολλο δρομολόγησης μεταξύ Αυτόνομων Συστημάτων (AS) στο Διαδίκτυο καθώς η επεκτασιμότητα του, και η ικανότητα υποστήριξης τεράστιου αριθμού διαδρομών το καθιστούν το πλέον κατάλληλο για τη διαχείριση της παγκόσμιας δρομολόγησης. Επιπλέον παρέχει ευελιξία πολιτικών και εμπεριέχει πολύπλοκους κανόνες δρομολόγησης με δυνατότητα εξατομίκευσης στις ανάγκες του χρήστη.

Πάρα την ιδιαίτερη αξία του το BGP παρουσιάζει αρκετές αδυναμίες που το καθιστούν ευάλωτο σε επιθέσεις και σφάλματα. Αρχικά δεν διαθέτει μηχανισμούς για να ελέγχει την ακεραιότητα και την αυθεντικότητα της προέλευσης των BGP μηνυμάτων, γεγονός που αφήνει το σύστημα εκτεθειμένο σε κακόβουλα ή εσφαλμένα μηνύματα. Επιπλέον, δεν υπάρχει τρόπος να επαληθευτεί η συσχέτιση μεταξύ ενός προθέματος δικτύου και του AS που το ανακοινώνει, δημιουργώντας εύφορο έδαφος για επιθέσεις τύπου *prefix hijacking*. Τέλος, το BGP δεν διασφαλίζει την ακρίβεια και την αξιοπιστία των attributes που περιέχονται σε ένα μήνυμα UPDATE, επιτρέποντας την ενδεχόμενη κυκλοφορία παραπλανητικών ή αλλοιωμένων πληροφοριών στη δρομολόγηση. Αυτές οι θεμελιώδεις αδυναμίες, σε συνδυασμό με την αργή σύγκλιση και την πολυπλοκότητα διαχείρισής του, καθιστούν το BGP ένα ευάλωτο πρωτόκολλο.

[41, 42, 43, 45, 57]

5.4 ECMP

Το Equal Cost Multi-Path (ECMP) συνιστά μια σημαντική τεχνική δρομολόγησης, η οποία αξιοποιεί την ύπαρξη πολλαπλών διαδρομών ίσου κόστους για το διαμοιρασμό της κίνησης. Σε αντίθεση με τα πρωτόκολλα δρομολόγησης που παρουσιάστηκαν προηγουμένως (RIP, OSPF, BGP) το ECMP δεν επιλέγει μια μοναδική διαδρομή προς χρήση αλλά μοιράζει την κίνηση σε όλες τις διαθέσιμες διαδρομές με το ίδιο κόστος. Με τον τρόπο αυτό αντί να παραμένει αχρησιμοποίητη μια ισοδύναμη εναλλακτική διαδρομή, τίθεται σε λειτουργία, συμβάλλοντας στην εξισορρόπηση της κίνησης (traffic load balancing) και την καλύτερη εκμετάλλευση των διαθέσιμων πόρων, ενισχύοντας την απόδοση του δικτύου.

Το βασικότερο ζήτημα που προκύπτει κατά την λειτουργία του ECMP είναι ο τρόπος με τον οποίο θα γίνει ο διαμοιρασμός της κίνησης στις διαθέσιμες διαδρομές. Η διαδικασία αυτή διενεργείται σύμφωνα με διάφορες τεχνικές, οι σημαντικότερες από τις οποίες είναι οι εξής:

- Εξισορρόπηση φορτίου ανά πακέτο – Per-Packet Load Balancing → Σύμφωνα με την τεχνική αυτή κάθε μεμονωμένο πακέτο χρησιμοποιεί με κυκλικό τρόπο μια διαφορετική, εκ των διαθέσιμων, διαδρομή. Με τον τρόπο αυτό διασφαλίζεται ο ακριβής διαμοιρασμός του φορτίου. Ελλοχεύει ωστόσο ο κίνδυνος άφιξης των πακέτων στον προορισμό τους με λανθασμένη σειρά (out-of-order delivery) λόγω καθυστερήσεων του δικτύου, με αποτέλεσμα την αλλοίωση της πληροφορίας που μεταφέρουν.

- Εξισορρόπηση φορτίου ανά ροή – Per-Flow Load Balancing → Η μέθοδος αυτή δεν αντιμετωπίζει κάθε πακέτο μεμονωμένα αλλά φροντίζει όλα τα πακέτα μιας ροής να ακολουθήσουν την ίδια διαδρομή. Η λειτουργία αυτή βασίζεται στην χρήση ενός αλγόριθμου κατακερματισμού (hashing) ο οποίος εκτελείται από τον δρομολογητή και παίρνει ως δεδομένα εισόδου συνήθως τα εξής 5 πεδία της επικεφαλίδας του πακέτου: IP διεύθυνση πηγής, IP διεύθυνση προορισμού, θύρα πηγής, θύρα προορισμού, πρωτόκολλο. Με βάση αυτά ο αλγόριθμος παράγει μια τιμή κατακερματισμού (hash value) σύμφωνα με την οποία επιλέγεται μία από τις διαθέσιμες διαδρομές. Όλα τα πακέτα της ίδιας ροής θα έχουν τις ίδιες τιμές εισόδου στην επικεφαλίδα τους, οπότε και η τιμή κατακερματισμού θα είναι η ίδια, με αποτέλεσμα να ακολουθούν την ίδια διαδρομή. Με τον τρόπο αυτό αποφεύγεται η αναδιάταξη των πακέτων και η παράδοση τους σε λανθασμένη σειρά (out-of-order delivery) αλλά δεν είναι δυνατό να εγγυηθεί ο ισότιμος διαμοιρασμός του φορτίου. Η αιτία αυτού είναι ότι γίνεται κατανομή των ροών χωρίς να λαμβάνεται υπόψιν ο όγκος των δεδομένων τους. Συνεπώς δεν συντελείται εξισορρόπηση του ρυθμού μετάδοσης ψηφίου (bit rate) και υπάρχει ο

κίνδυνος συμφόρησης του δικτύου με αποτέλεσμα να προκύψουν καθυστερήσεις και να μειωθεί η συνολική απόδοση.

- Weighted ECMP → Η τεχνική αυτή κατανέμει την κίνηση αναλογικά συνυπολογίζοντας το εύρος ζώνης της κάθε διαθέσιμης διαδρομής. Αποφεύγεται η υπερφόρτωση μιας διαδρομής με χαμηλή χωρητικότητα τη στιγμή που μια σύνδεση υψηλής χωρητικότητας μένει υποχρησιμοποιημένη. Το κύριο πλεονέκτημα αυτής της μεθόδου είναι ότι γίνεται αποδοτικότερη χρήση των πόρων του δικτύου, καλύπτοντας τις περιπτώσεις στις οποίες οι διαδρομές δεν είναι ομοιόμορφες.

Συνολικά το ECMP αποτελεί μια ευέλικτη τεχνολογία αρκετά χρήσιμη για την διαχείριση δικτύων μεγάλης κλίμακας σχετικά με την βελτιστοποίηση των διαθέσιμων πόρων και την αποφυγή συμφόρησης. Σε κάθε περίπτωση ωστόσο παρουσιάζει αξιοσημείωτες αδυναμίες λόγω της αυξημένης πολυπλοκότητας του και της δυσκολίας απολύτως ισορροπημένου καταμερισμού φορτίου με ταυτόχρονη αξιόπιστη μετάδοση πληροφορίας.

[40, 58, 59, 60, 61]

5.5 Μεταγωγείς και γέφυρες

Ένα μεταγωγέας (switch) αποτελεί ένα τμήμα ενεργού φυσικού εξοπλισμού, το οποίο χρησιμοποιείται για την σύνδεση πολλαπλών συσκευών εντός ενός τοπικού δικτύου (LAN – Local Area Network). Σε αντίθεση με τους δρομολογητές, που λειτουργούν στο στρώμα δικτύου και χρησιμοποιούν τις IP διευθύνσεις των συσκευών για να υλοποιήσουν συνδέσεις εντός ενός δικτύου αλλά και κυρίως μεταξύ διαφορετικών δικτύων, οι μεταγωγείς λειτουργούν στο στρώμα ζεύξης δεδομένων και χρησιμοποιούν τις διευθύνσεις MAC για να συνδέουν συσκευές μέσα στο ίδιο δίκτυο. Μια παρόμοια προγενέστερη συσκευή δικτύου είναι τα hubs, τα οποία λειτουργούν στο φυσικό επίπεδο και αναμεταδίδουν όλα τα εισερχόμενα πακέτα σε όλες τις διαθέσιμες θύρες χωρίς να λαμβάνουν υπόψιν τις MAC διευθύνσεις όπως οι μεταγωγείς. Η στοχευμένη αυτή ανακατεύθυνση των πακέτων από τους μεταγωγείς μόνο στις συσκευές που τα χρειάζονται μειώνει σημαντικά την συμφόρηση του δικτύου και βελτιώνει την απόδοση του. Η τεχνολογία των μεταγωγέων μοιάζει αρκετά με αυτή των γεφυρών (bridges). Η βασική τους διαφορά εντοπίζεται στο πλήθος των θυρών. Οι γέφυρες έχουν μόνο δύο θύρες ενώ οι μεταγωγείς επιτρέπουν την ταυτόχρονη λειτουργία πολλών θυρών.

Η ευρεία χρήση των τεχνικών εικονικοποίησης και η προσομοίωση δικτύων οδήγησε στην ανάγκη δημιουργίας εικονικών μηχανισμών που θα λειτουργούν σαν φυσικοί μεταγωγείς. Έτσι υλοποιήθηκαν εικονικές γέφυρες και μεταγωγείς με λειτουργία

σχεδόν ταυτόσημη με αυτή του φυσικού εξοπλισμού. Πρόκειται για μηχανισμούς λογισμικού οι οποίοι επιτρέπουν την σύνδεση των εικονικών κόμβων με εξωτερικά φυσικά δίκτυα μέσω των φυσικών καρτών δικτύου. Η λειτουργία τους βασίζεται στην εκμάθηση MAC διευθύνσεων και την αποθήκευση τους σε πίνακες προώθησης, ώστε κατά η διάδοση των πακέτων να γίνεται στην κατάλληλη διεπαφή. Σε περίπτωση που η MAC διεύθυνση προορισμού του πακέτου είναι άγνωστη και απουσιάζει από τον πίνακα προώθησης τότε το πακέτο εκπέμπεται προς όλες τις διεπαφές εκτός αυτής από την οποία λήφθηκε (flooding). Επιπλέον υποστηρίζουν εικονικά τοπικά δίκτυα (Virtual Local Area Networks), τα οποία επιτρέπουν τον εικονικό διαχωρισμό ενός φυσικού δικτύου και την απομόνωση τμημάτων του στο στρώμα ζεύξης δεδομένων. Οι μεταγωγείς είναι ικανοί να κατευθύνουν κίνηση μεταξύ διαφορετικών VLAN με τη χρήση ετικετών (VLAN tags) σύμφωνα με το πρότυπο IEEE 802.1Q. Η ετικέτα αυτή περιλαμβάνει το αναγνωριστικό του VLAN (VLAN ID) και με βάση αυτήν οι μεταγωγείς προωθούν τα πλαίσια κατάλληλα εντός του VLAN. Το μέγεθος του πεδίου VLAN ID είναι 12 bit, οπότε τα διαθέσιμα VLAN που μπορούν να δημιουργηθούν είναι 2^{12}=4096. Οι τιμές 0 και 4095 δεσμεύονται για ειδικές χρήσεις οπότε υπάρχουν συνολικά 4094 διαφορετικά VLAN ID.

Οι εικονικοί μεταγωγείς είναι ευέλικτοι και οικονομικοί καθώς βασίζονται σε λογισμικό και αξιοποιούν τις ήδη υπάρχουσες φυσικές κάρτες δικτύου. Ωστόσο συχνά παρέχουν περιορισμένες δυνατότητες παρακολούθησης και διαχείρισης της κίνησης δυσχεραίνοντας την εφαρμογή πολιτικών ασφαλείας και την ανίχνευση δυσλειτουργιών.

[65, 66, 67, 68, 69, 70, 71]

5.6 VXLAN

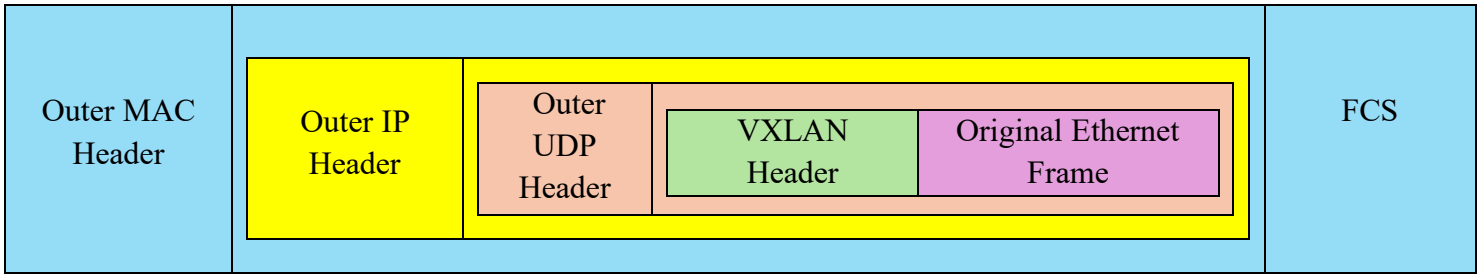
Η εξάπλωση των τεχνικών εικονικοποίησης δημιούργησε υψηλές απαιτήσεις υποστήριξης μεγάλους πλήθους εικονικών μηχανημάτων από τα σύγχρονα Data Centers, τα οποία έχουν την ανάγκη να μπορούν να εξυπηρετούν πολλαπλούς πελάτες σε μια φυσική υποδομή (multitenancy) παρέχοντας απομόνωση σε επίπεδο δικτύου. Η συμβατική λύση για απομόνωση τμημάτων δικτύου στο στρώμα ζεύξης δεδομένων με την χρήση VLAN αποδεικνύεται ανεπαρκής. Αρχικά το ανώτατο όριο 4094 VLAN IDs αποτελεί έναν πολύ μεγάλο περιορισμό για τις ανάγκες των Data Centers και ελλοχεύει κινδύνους συγκρούσεων λόγω επικαλυπτόμενων VLAN IDs. Επιπλέον η ενσωμάτωση VLAN σε πολύ μεγάλα δίκτυα δεν είναι καθόλου εύχρηστη καθώς απαιτούν ρύθμιση σε κάθε έναν μεταγωγέα που διαχειρίζεται εικονικά μηχανήματα, κάνοντας τη διαχείριση τους ιδιαίτερα πολύπλοκη. Τέλος η άμεση εξάρτηση των VLAN από το στρώμα ζεύξης δεδομένων καθιστά αδύνατη την μετακίνηση εικονικών μηχανών και συστημάτων από έναν host σε έναν άλλο χωρίς να μεσολαβήσει ένα διάστημα διακοπής της λειτουργίας τους.

Το VXLAN (Virtual eXtensible Local Area Network) αποτελεί μια τεχνολογία εικονικοποίησης δικτύου που επινοήθηκε για να αντιμετωπίσει τα παραπάνω προβλήματα. Η λειτουργία του βασίζεται στην έννοια του δικτύου επικάλυψης (overlay network). Ένα δίκτυο επικάλυψης είναι ένα λογικό δίκτυο τοποθετούμενο πάνω από ένα φυσικό. Στην περίπτωση του VXLAN το δίκτυο επικάλυψης είναι ένα δίκτυο στρώματος δικτύου (Layer 2) το οποίο ονομάζεται VXLAN Segment και υλοποιείται πάνω από το υποκείμενο (underlay network), ήδη υπάρχον στρώμα μεταφοράς (Layer 3). Το σχήμα που ακολουθείται περιλαμβάνει την ενθυλάκωση πλαισίων Ethernet σε δεδομενογράμματα UDP, τα οποία δρομολογούνται κανονικά από υποκείμενο δίκτυο του στρώματος μεταφοράς.

Κάθε VXLAN Segment χαρακτηρίζεται από το VNI (VXLAN Network Identifier), ένα αναγνωριστικό μήκους 24 bit. Η ύπαρξη αυτού του αναγνωριστικού επιτρέπει την ταυτόχρονη ύπαρξη έως και 16 εκατομμυρίων διαφορετικά VXLAN Segments, όριο το οποίο ξεπερνάει κατά πολύ τα 4094 VLAN IDs. Το VNI βρίσκεται σε μια εξωτερική επικεφαλίδα που ενθυλακώνει το αρχικό πλαίσιο Ethernet και εξασφαλίζει την απομόνωση της σχετιζόμενης με ένα VXLAN Segment κίνησης. Συνεπώς αποφεύγονται οι συγκρούσεις ακόμα και σε περιπτώσεις επικάλυψης των MAC διευθύνσεων.

Ένα θεμελιώδες στοιχείο της λειτουργίας του VXLAN είναι το VTEP (VXLAN Tunnel End Point). Διακρίνονται σε VTEP πηγής (source VTEP) και VTEP προορισμού (destination VTEP). Ένα VTEP πηγής είναι υπεύθυνο για την ενθυλάκωση (encapsulation) του αρχικού πλαισίου που λαμβάνει από τον αποστολέα σε πακέτα VXLAN και την μετάδοση τους στον προορισμό τους μέσω του δικτύου IP. Αντίστοιχα ένα VTEP προορισμού αναλαμβάνει την αποθυλάκωση (decapsulation) των πακέτων VXLAN στα αρχικά πλαίσια δεδομένων, ώστε να προωθηθούν στον τελικό τους προορισμό αυτούσια όπως στάλθηκαν από την πηγή. Ένα VTEP μπορεί να είναι μια ανεξάρτητη συσκευή όπως ένας φυσικός ή εικονικός μεταγωγέας, ένας δρομολογητής ή ακόμα και ένας Linux host.

Στο παρακάτω σχήμα αναπαρίσταται η δομή ενός VXLAN πακέτου, στην οποία φαίνεται η ακριβής ακολουθία των επικεφαλίδων που ενθυλακώνονται.



Εικόνα 9: *Ενθυλάκωση επικεφαλίδων σε ένα VXLAN πακέτο*

Outer MAC Header: Στα πεδία Destination MAC Address και Source MAC Address περιλαμβάνονται αντίστοιχα η MAC διεύθυνση προορισμού που αντιστοιχεί στο επόμενο στο επόμενο βήμα που απαιτείται για το VTEP προορισμού και η MAC διεύθυνση πηγής του VTEP πηγής που ενθυλακώνει το αρχικό πλαίσιο.

Outer IP Header: Η Source IP Address αντιστοιχεί στην διεύθυνση IP του VTEP πηγής και η Destination IP Address στην διεύθυνση IP του VTEP προορισμού. Επιπλέον η τιμή του πεδίου Protocol είναι 17 (UDP) καθώς το πρωτόκολλο στρώματος μεταφοράς που χρησιμοποιείται είναι το UDP.

Outer UDP Header: Η UDP θύρα προορισμού που έχει αποδοθεί στο VXLAN είναι η 4789 και αποτελεί την τιμή του πεδίου Destination Port. Η θύρα πηγής προσδιορίζεται δυναμικά με βάση μια συνάρτηση κατακερματισμού (hash) και βρίσκεται στο εύρος 49152-65535.

VXLAN Header: Η επικεφαλίδα VXLAN αποτελείται από 8 byte και έχει την εξής μορφή:

FLAGS RRRRIRRR	RESERVED
VNI	RESERVED

Εικόνα 10: *Δομή επικεφαλίδας VXLAN*

Το πεδίο VNI αποτελείται από 24 bits και περιλαμβάνει το αναγνωριστικό του VXLAN segment που χρησιμοποιείται.
Το πεδίο FLAGS καταλαμβάνει 8 bits από τα οποία τα 7 συμβολιζόμενα ως R πρέπει να είναι μηδενικά κατά την μετάδοση, ενώ κατά την λήψη δεδομένων αγνοούνται. Το ψηφίο I πρέπει να ορίζεται σε 1 για έγκυρο VNI.
Τα δύο πεδία RESERVED (24 και 8 bits) πρέπει και αυτά να είναι μηδενικά κατά την μετάδοση και κατά την λήψη δεδομένων αγνοούνται.

Original Ethernet Frame: Συνιστά το αρχικό πακέτο που στάλθηκε και παραμένει αναλλοίωτο καθ' όλη τη διάρκεια της μετάδοσης

FCS: Το Frame Check Sequence χρησιμοποιείται για τον έλεγχο της ακεραιότητας των δεδομένων κατά τη μετάδοσης και την ανίχνευση σφαλμάτων.

Για την εκμάθηση των MAC διευθύνσεων το VXLAN χρησιμοποιεί έναν μηχανισμό που ονομάζεται Flood-and-Learn. Όταν ένα VTEP λαμβάνει ένα πακέτο για πρώτη φορά από έναν κόμβο ενημερώνεται για την MAC του και την αποθηκεύει στον πίνακα προώθησης του. Σε περίπτωση που το VTEP λάβει ένα πακέτο και δεν έχει αποθηκευμένη στον πίνακα προώθησης του την MAC διεύθυνση προορισμού του πλημμυρίζει (flood) το πακέτο σε όλους τους άλλους VTEP του ίδιου VXLAN

Segment. Στη συνέχεια ο κόμβος στον οποίο αντιστοιχεί η MAC απαντάει και έτσι το VTEP μαθαίνει την αντιστοιχία και αποθηκεύει την MAC στον πίνακα προώθησης του για μελλοντική χρήση.

Η τεχνική του VXLAN προσέφερε δραστικές λύσεις για σύγχρονα δικτυακά περιβάλλοντα όπως τα Data Centers και όχι μόνο. Η αύξηση της επεκτασιμότητας σε έως και 16 εκατομμύρια λογικά δίκτυα, η βελτιωμένη ανθεκτικότητα του και η ευελιξία στη διαχείριση πολλών συστημάτων καθιέρωσε την ευρεία χρήση του. Ιδιαίτερη προσοχή στη χρήση του προκειμένου να αποφευχθούν δυσλειτουργίες χρειάζεται στην ορισμό της MTU, καθώς το επιπλέον επίπεδο που ενθυλακώνεται υπερφορτώνει τα πακέτα (overhead) και υπάρχει ο κίνδυνος θρυμματισμού των πακέτων (fragmentation) με πιθανό αποτέλεσμα καθυστερήσεις και απώλεια πακέτων ή αλλοίωση του αρχικού μηνύματος. Επιπλέον η προσέγγιση του VXLAN για την εκμάθηση των MAC διευθύνσεων ενδέχεται να δημιουργήσει συμφόρηση σε μεγάλα δίκτυα.

[72, 73, 75, 76, 77, 78, 79, 80, 81, 82, 83 84]

5.7 EVPN

Το EVPN (Ethernet Virtual Private Network) αποτελεί μια επέκταση του BGP, λειτουργεί στο επίπεδο ελέγχου (control plane) και σχεδιάστηκε για να προσφέρει αποδοτικότερη εκμάθηση MAC διευθύνσεων και να αντιμετωπίσει τα προβλήματα που προκύπτουν από τον μηχανισμό flood and learn του VXLAN.

Το EVPN επιτρέπει την ταυτόχρονη λήψη πληροφοριών για MAC και IP διευθύνσεις. Η λειτουργία βασίζεται στην τεχνική ARP Suppression, στοχεύοντας στην δραστική μείωση της κίνησης ARP και την εξάλειψη για flooding. Τα VTEPs μαθαίνουν τοπικά αντιστοιχίες MAC και IP διευθύνσεων των άμεσα συνδεδεμένων κόμβων και έπειτα ανταλλάσσουν αυτές τις πληροφορίες μεταξύ τους μέσω πρωτοκόλλου BGP και τις αποθηκεύουν στους πίνακες προώθησης τους. Με τον τρόπο αυτό κάθε VTEP έχει την απαραίτητη γνώση για να αποστείλει τα πακέτα για κάθε προορισμό του δικτύου στο σωστό VTEP χωρίς να προηγηθεί flooding.

Επιπλέον το EVPN υποστηρίζει multi-homing με all-active redundancy. Προσφέρει δηλαδή πλεονάζουσα συνδεσιμότητα (redundant connectivity) στο τελικό σύστημα του διακομιστή – πελάτη συνδέοντας το δύο ή περισσότερες συσκευές δικτύου, οι οποίες μπορούν να προωθούν ενεργά την κίνηση ταυτόχρονα. Σημειώνεται ότι όλες οι διαθέσιμες διαδρομές θεωρούνται ενεργές και ικανές να μεταδώσουν κίνηση και δεν υπάρχει κύρια και εφεδρική διαδρομή. Έτσι επιτυγχάνεται εξισορρόπηση φορτίου αλλά και γρήγορη σύγκλιση καθώς σε περίπτωση δυσλειτουργίας μιας διαδρομής ή αποτυχίας ενός VTEP να μεταδώσει τα δεδομένα, η πληροφορία για την αποτυχία

αυτή μεταδίδεται στα υπόλοιπα VTEP μέσω BGP και αυτά σταματούν να στέλνουν κίνηση προς την μη διαθέσιμη διαδρομή, μεταφέροντας την κίνηση σε άλλη ήδη ενεργή διαδρομή.

Στο EVPN υπάρχουν 5 διαφορετικά είδη ανακοινώσεων, καθένα από τα οποία μεταφέρει διαφορετικό είδος πληροφορίας:

- Route Type 1: Αποτελεί μια ανακοίνωση Ethernet Auto-Discovery (EAD) που χρησιμοποιείται για την διαφήμιση πληροφοριών όπως το αναγνωριστικό ενός Ethernet segment, το αναγνωριστικό ετικέτας Ethernet (Ethernet Tag ID).

- Route Type 2: Διαφημίζει πληροφορίες προσβασιμότητας του τελικού κόμβου, συμπεριλαμβανομένων των διευθύνσεων MAC και IP του ή αυτών του VTEP.

- Route Type 3: Θεωρείται ως διαδρομή πολλαπλής διανομής. Αυτή η διαδρομή χρησιμοποιείται για τη δυναμική ανακάλυψη VTEP και τη δυναμική σύνδεση VXLAN tunnels.

- Route Type 4: Χρησιμοποιείται για την διαφήμιση του αναγνωριστικού τμήματος Ethernet, του μήκους της διεύθυνσης IP και της διεύθυνσης IP του δρομολογητή προέλευσης.

- Route Type 5: Είναι μια διαδρομή προθέματος IP που χρησιμοποιείται για την διαφήμιση εσωτερικών υποδικτύων IP και εξωτερικών διαδρομών σε ένα δίκτυο VXLAN.

Η χρήση της τεχνικής του EVPN σε συνδυασμό με το VXLAN παρέχει ένα πολύ πιο ισχυρό δίκτυο επικάλυψης με υψηλότερες δυνατότητες κλιμάκωσης και επεκτασιμότητας. Ενισχύει επίσης σημαντικά περιβάλλοντα όπως τα Data Centers λόγω ευρείας υποστήριξης πολλών πελατών που αξιοποιούν τους ίδιους πόρους αλλά διατηρούν τα δεδομένα τους απομονωμένα (multi-tenancy). Η μέθοδος που ακολουθεί το EVPN για την εκμάθηση των MAC διευθύνσεων ελαχιστοποιεί τον κίνδυνο υπερχείλισης του δικτύου και συμβάλλει στην αποδοτικότερη διαχείριση της κίνησης. Σε κάθε περίπτωση η ενσωμάτωση του σε δίκτυα θα πρέπει να γίνεται με προσοχή στις ρυθμίσεις του καθώς λόγω της αλληλεπίδρασης πρωτοκόλλων (VXLAN, BGP) αυξάνεται η πολυπλοκότητα του και ο εντοπισμός πιθανών προβλημάτων απαιτεί την παρακολούθηση τόσο του data plane (VXLAN) όσο και του control plane (BGP).

[74, 75, 76, 85]

6. Παρουσίαση των images που χρησιμοποιήθηκαν

Η επιλογή κατάλληλων images σε μια τοπολογία είναι ιδιαίτερα σημαντική. Θα πρέπει να εξασφαλίζεται ότι αυτά περιέχουν τα απαραίτητα εργαλεία και διαθέτουν τα επιθυμητά χαρακτηριστικά προκειμένου ώστε να προσομοιωθεί με ακρίβεια το δικτυακό σύστημα.

Στα πλαίσια της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκαν τρία images, η έκδοση 10.0.2 του FRR image, η πιο πρόσφατη έκδοση του Wbitt/network-multitool και ένα ειδικά διαμορφωμένο docker image για δοκιμές Equal-Cost-Multi-Path (ECMP).

6.1 FRR

Το FRR (Free Range Routing) είναι ένα δωρεάν λογισμικό δρομολόγησης ανοικτού κώδικα για συστήματα UNIX και LINUX. Αποτελεί συνέχεια και εξέλιξη του παλιότερου αντίστοιχου λογισμικού Quagga, το οποίο δημιουργήθηκε ως διακλάδωση (fork) του αρχικού Zebra. Ο πυρήνας του είναι ο δαίμονας zebra, υπεύθυνος για την μεταφορά εντολών μεταξύ πρωτοκόλλων δρομολόγησης και λειτουργικού συστήματος. Υποστηρίζει μεταξύ άλλων τα πρωτόκολλα BGP, OSPF, RIP, IS-IS, PIM, LDP, BFD, PBR, EIGRP και NHRP. Για την υλοποίηση κάθε πρωτοκόλλου είναι υπεύθυνος ένας ξεχωριστός δαίμονας. Η αρχιτεκτονική αυτή του FRR παρέχει μεγάλη ευελιξία στη διαχείριση, καθώς μπορούν να ενεργοποιηθούν μόνο οι απαραίτητοι για την εκάστοτε τοπολογία δαίμονες, εξοικονομώντας με τον τρόπο αυτό πόρους και περιορίζοντας την επιφάνεια επίθεσης (attack surface). [26]

Το docker image του FRR βρίσκεται στο repository https://quay.io/repository/frrouting/frr. Έχει συχνές κυκλοφορίες νέων εκδόσεων, τακτικές διορθώσεις ασφαλείας και βελτιστοποιήσεις, γεγονός που ενισχύει τη σταθερότητα και την αξιοπιστία του. Επιπλέον χρησιμοποιείται και σε εμπορικές διανομές του Linux όπως Cumulus Linux και SONiC.

Το FRR image περιλαμβάνει τον δαίμονα zebra αλλά και τους επιμέρους δαίμονες των πρωτοκόλλων και η διαχείριση του γίνεται μέσω του vtysh, ενός ενιαίου περιβάλλοντος γραμμής εντολών (CLI), χωρίς να υπάρχει ανάγκη για σύνδεση σε κάθε δαίμονα ξεχωριστά. Πρόκειται για ένα αρκετά ελαφρύ image με μέγεθος περίπου 100-150MB και μικρές απαιτήσεις σε CPU, RAM, το οποίο προσφέρει τη δυνατότητα παράλληλης χρήσης δεκάδων FRR container με μικρούς χρόνους εκκίνησης. Τα χαρακτηριστικά αυτά ευνοούν σημαντικά την ανάπτυξη μεγάλων τοπολογιών και τον έλεγχο σύνθετων σεναρίων δρομολόγησης με συνδυασμό πρωτοκόλλων. Ένα FRR container λειτουργεί στις τοπολογίες κατά βάση ως

εικονικός δρομολογητής και προσφέρει ένα αρκετά ρεαλιστικά περιβάλλον προσομοίωσης. Στο Containerlab ένας κόμβος με kind → linux μπορεί να φορτώσει στη συνέχεια το image του FRR. Για προσαρμογή των daemons και εφαρμογή παραμετροποιήσεων χρησιμοποιούνται bind mounts με τα αρχεία /etc/frr/daemons και /etc/frr/frr.conf αντίστοιχα.

6.2 Wbitt/network-multitool

Το wbitt/network-multitool είναι ένα γενικής χρήσης image βασισμένο στη διανομή Alpine του Linux και βρίσκεται στο εξής repository https://github.com/wbitt/Network-MultiTool. Η διανομή Alpine αποτελεί μια ιδανική βάση με γρήγορες ταχύτητες εκκίνησης και χαμηλή κατανάλωση πόρων. Συμπεριλαμβάνει πληθώρα χρήσιμων εργαλείων για καταγραφή πακέτων (tcpdump), παρακολούθηση συνδέσεων (netstat), τροποποίηση διεπαφών (ip, ethtool), DNS clients (nslookup, dig), μέτρηση bandwidth (iperf3) και πολλά άλλα (curl, ping, traceroute). Έχει μικρό μέγεθος, περίπου 80MB, γεγονός που ενισχύεται από την απουσία περιττών προεγκατεστημένων πακέτων και συνεπώς την άσκοπη κατανάλωση πόρων. Αντίθετα δίνει στον χρήστη την επιλογή να προσθέσει χειροκίνητα τα πακέτα που χρειάζεται. Η συνηθέστερη χρήση του στις τοπολογίες είναι ως host στα άκρα της. Μπορεί ωστόσο να χρησιμοποιηθεί και σε ενδιάμεσα σημεία για την ανίχνευση προβλημάτων. Παρέχει σημαντικές διαγνωστικές δυνατότητες για το δίκτυο, αποτελώντας ένα πολύ κατάλληλο image για την επιβεβαίωση της ορθής λειτουργίας πρωτοκόλλων δρομολόγησης χωρίς να απαιτούνται εξωτερικά εργαλεία που αυξάνουν την πολυπλοκότητα. [29]

Ακολουθούν κάποιες από τις βασικότερες εντολές του:

- **ip link set**
  ip link set [interface] up
  ip link set [interface] down
  ip link delete [interface]
  ip link set dev [current-interface-name] name [new-interface-name]
  ip link set dev [interface] mtu [size]
  ip link add [interface] type [type]
  ip link show [interface]

- **ip address**
  ip address add [IP] dev [interface]
  ip address change [IP] dev [interface]
  ip address del [IP] dev [interface]
  ip address replace [IP] dev [interface]
  ip address show dev [interface]

- **ip route**
  ip route list
  ip route flush
  ip route flush to [destination]
  ip route flush dev [interface]
  ip route add [destination] via [gateway]
  ip route del [destination] via [gateway]
  ip route change [destination] via [gateway]
  ip route replace [destination] via [gateway]

6.3 ECMP

Προκειμένου να δοκιμαστεί η λειτουργία του ECMP χρειάζεται η δημιουργία κίνησης με υψηλή ταχύτητα και η δυνατότητα λεπτομερούς ελέγχου των παραμέτρων των παραγόμενων πακέτων ώστε να υπάρχει ποικιλία διευθύνσεων IP ή θυρών πηγής και προορισμού με αποτέλεσμα να δημιουργηθούν διαφορετικές ροές οι οποίες θα κατανεμηθούν σε διαφορετικά μονοπάτια. Τα τυπικά images βασισμένα σε Ubuntu/Debian δεν περιλαμβάνουν εργαλεία με τέτοιες επιδόσεις στην παραγωγή κίνησης. Για παράδειγμα το iperf3 μπορεί να δημιουργήσει TCP ή UDP συνδέσεις αλλά μόνο σε περιορισμένο αριθμό με χαμηλούς ρυθμούς και χωρίς την δυνατότητα προσδιορισμού των χαρακτηριστικών των πακέτων. Για το λόγο αυτό για τις ανάγκες της παρούσας διπλωματικής στα πλαίσια διερεύνησης της τεχνικής ECMP δημιουργήθηκε ένα νέο εξατομικευμένο docker image, η λειτουργία του οποίου θα βασίζεται στο Netmap.

Το Netmap είναι ένα πλαίσιο εισόδου/εξόδου πακέτων υψηλής απόδοσης, το οποίο χρησιμοποιεί διάφορες μεθόδους βελτιστοποίησης ώστε να εξαλείψει υπερφορτώσεις στην επεξεργασία πακέτων (packet processing overhead) και να αυξήσει τους ρυθμούς μετάδοσης τους (packet throughput). Εφαρμόζει απλή αναπαράσταση πακέτων, ομαδοποίηση πακέτων και απευθείας σύνδεση εφαρμογών στο user space με θύρες καρτών δικτύου (NIC). Μια εφαρμογή με netmap API μπορεί να φτάσει μέγιστο ρυθμό απόδοσης τα 14,88 Mpps (throughput) σε θύρα 10 Gbit/sec χρησιμοποιώντας έναν πυρήνα CPU. Η ίδια εφαρμογή με τη στοίβα δικτύου του πυρήνα (Linux New API - NAPI) έχει μέγιστο ρυθμό απόδοσης μόνο 2 Mpps.

Ένα από τα βασικά εργαλεία του Netmap, είναι το pkt-gen, μια εφαρμογή που χρησιμοποιεί το Netmap API για την παραγωγή και τη λήψη πακέτων με πολύ υψηλή ταχύτητα. Στόχος του pkt-gen είναι να στέλνει και να δέχεται πακέτα με ρυθμούς τόσο υψηλούς που προσεγγίζουν την μέγιστη ταχύτητα του καλωδίου. Για το στόχο αυτό αφού δημιουργήσει τα πακέτα βασίζεται στο Netmap API για να σταλεί η κίνηση στο δίκτυο με ελάχιστη καθυστέρηση και χωρίς να μεσολαβήσει ο πυρήνας.

Το pkt-gen συγκεντρώνει όλα τα επιθυμητά χαρακτηριστικά για χρήση σε εργαστήρια με ECMP, καθώς δίνει τη δυνατότητα μαζικής παραγωγής και λήψης πακέτων με υψηλούς ρυθμούς, και ευελιξία στον προσδιορισμό των πεδίων των πακέτων (IP/MAC διευθύνσεις, TCP/UDP θύρες, μέγεθος), καθιστώντας εφικτή την δημιουργία πολλαπλών ροών κίνησης. [27, 28]

Το image που κατασκευάστηκε στα πλαίσια της διπλωματικής εργασίας ονομάστηκε pktgen-container και έχει ως βάση το ακόλουθο Dockerfile:

```
#Base image (Ubuntu 22.04)
FROM ubuntu:22.04

#Install dependencies
RUN apt update && apt install -y \
            build-essential \
            git \
            libpcap-dev \
            linux-headers-$(uname -r) \
            kmod \
            iproute2 \
            iputils-ping \
            net-tools \
            ethtool \
            && rm -rf /var/lib/apt/lists/*

#Clone the Netmap repository
WORKDIR /opt
RUN git clone https://github.com/luigirizzo/netmap.git && \
            cd netmap/LINUX && \
            ./configure && make && make install

# Set work directory
WORKDIR /opt/netmap/LINUX/apps

```

Εικόνα 11: *Dockerfile του image pktgen-container*

Για την κατασκευή του image χρησιμοποιείται Ubuntu 22.04 LTS ως βάση, ώστε να εξασφαλιστεί ένας σύγχρονος πυρήνας με σταθερότητα, ευρεία υποστήριξη πακέτων και πλήρη συμβατότητα με τα εργαλεία ανάπτυξης και τις εξαρτήσεις του Netmap Στη συνέχεια γίνεται η εγκατάσταση των απαραίτητων εξαρτήσεων:

- ***build-essential*** → Πακέτο με compilers και βασικά εργαλεία ανάπτυξης απαραίτητα για μεταγλώττιση από πηγαίο κώδικα (source code) (εργαλεία όπως gcc, make, dpkg-dev). Απαραίτητο για τη μεταγλώττιση του Netmap στη συνέχεια.
- ***git*** → Δωρεάν, ανοιχτού κώδικα σύστημα ελέγχου εκδόσεων, αναγκαίο για τη λήψη πηγαίου κώδικα από repositories σε Github/Gitlab. Επιτρέπει παρακάτω τη λήψη του Netmap από το επίσημο repository https://github.com/luigirizzo/netmap.git . [30]
- ***libpcap-dev*** → Βιβλιοθήκες ανάπτυξης για την βιβλιοθήκη libpcap, η οποία χρησιμοποιείται για καταγραφή πακέτων και παρακολούθηση δικτύου. Το Netmap και το pkt-gen βασίζονται σε αυτήν. [31]
- ***linux-headers-$(uname -r)*** → Η εντολή uname -r επιστρέφει το ακριβές όνομα της έκδοσης του πυρήνα και έπειτα εγκαθίστανται τα ανάλογα header files του Linux. Το Netmap χρειάζεται αυτά τα αρχεία διότι περιλαμβάνει kernel modules.

- ***kmod*** → Πακέτο με εργαλεία όπως lsmod, insmod, rmmod, modprobe για τη διαχείριση kernel modules. Με αυτά τα εργαλεία είναι δυνατή η προσθήκη και ο έλεγχος των kernel modules του Netmap. [33, 34]
- ***iproute2*** → Σύνολο εργαλείων για διαχείριση δικτύου με διάφορες χρήσεις όπως διαμόρφωση διεπαφών και δρομολόγηση (ip route, ip link, ip addr). [35]
- ***iputils-ping*** → Περιλαμβάνει την εντολή ping, αναγκαία για την επιβεβαίωση επικοινωνίας μεταξύ των κόμβων μιας τοπολογίας. [36]
- ***net-tools*** → Παλαιότερα εργαλεία διαχείρισης δικτύου, χρήσιμα για έλεγχο και διόρθωση σφαλμάτων (ifconfig, netstat, arp, hostname). [37]
- ***ethtool*** → Πακέτο με δυνατότητες παραμετροποίησης και ελέγχου δικτυακών διεπαφών. [38]

Στη συνέχεια χρησιμοποιείται η εντολή *rm -rf /var/lib/apt/lists/** για να διαγραφούν προσωρινά αρχεία που προέκυψαν από την εγκατάσταση των πακέτων ώστε να διατηρηθεί το μέγεθος του image όσο το δυνατόν μικρότερο και απαλλαγμένο από περιττά αρχεία.

Ακολουθεί λήψη του πηγαίου κώδικα (source code) του Netmap από το επίσημο repository https://github.com/luigirizzo/netmap.git και μετάβαση στον φάκελο netmap/LINUX στο οποίο περιλαμβάνονται τα kernel modules και οι εφαρμογές του Netmap. Εκεί με την εντολή ./configure γίνεται έλεγχος του συστήματος και προετοιμασία για μεταγλώττιση με προσαρμογή στον τρέχον πυρήνα. Η μεταγλώττιση του πηγαίου κώδικα γίνεται έπειτα με την εντολή make ενώ με την make install εγκαθίστανται οι βιβλιοθήκες και τα δυαδικά αρχεία που προέκυψαν από την μεταγλώττιση.

Τέλος ορίζεται ως προεπιλεγμένος τρέχων φάκελος (working directory) ο φάκελος apps του Netmap (/opt/netmap/LINUX/apps) στον οποίο βρίσκονται τα διάφορα εργαλεία όπως το pkt-gen. Με τον τρόπο αυτό όταν δημιουργείται το container ο χρήστης βρίσκεται ήδη στον κατάλληλο φάκελο για να χρησιμοποιήσει αυτές τις λειτουργίες.

Αφού ολοκληρωθεί η σύνταξη του Dockerfile, το image κατασκευάζεται με την εντολή *docker build -t pktgen-container.* Έπειτα επιβεβαιώνεται η ορθή του δημιουργία εκτελώντας *docker images | grep pktgen-container*

```
akoutsoni@clab:~/pktgen-container$ docker images | grep pktgen-container
pktgen-container              latest    0b795712da46   3 weeks ago     600MB
```

Εικόνα 12: *Επιβεβαίωση δημιουργίας image*

Ωστόσο η κατασκευή ενός docker image που θα περιέχει το Netmap δεν αρκεί για να μπορεί αυτό να χρησιμοποιηθεί εντός των container. Το Netmap δεν αποτελεί απλή εφαρμογή του user space, αλλά μια επέκταση του πυρήνα του Linux. Η λειτουργία του βασίζεται στην παράκαμψη της κανονικής στοίβας δικτύου οπότε απαιτείται τροποποίηση στον τρόπο που ο πυρήνας του host μηχανήματος χειρίζεται τις κάρτες δικτύου. Για το λόγο αυτό θα πρέπει τα module του Netmap να φορτωθούν και να μεταγλωττιστούν και στο host μηχάνημα.

Ακολουθούνται τα εξής βήματα:

- *cd /opt* → Μετάβαση στον φάκελο /opt ο οποίος χρησιμοποιείται συνήθως για προαιρετικά πακέτα και λογισμικό εκτός του λειτουργικού συστήματος.
- *git clone https://github.com/luigirizzo/netmap.git* → Λήψη του πηγαίου κώδικα του Netmap.
- *cd netmap/LINUX* → Μετάβαση στον φάκελο που περιέχει τα kernel modules του Netmap για Linux
- *./configure --kernel-sources=/lib/modules/$(uname -r)/build* → Προσαρμογή των απαραίτητων ρυθμίσεων για μεταγλώττιση του πηγαίου κώδικα με βάση την ακριβή έκδοση πυρήνα του συστήματος.
- *sudo make* → Μεταγλώττιση του πηγαίου κώδικα.
- *sudo make install* → Εγκατάσταση των kernel modules και των δυαδικών αρχείων.
- *sudo insmod ./netmap.ko* → Φορτώνεται δυναμικά το kernel module του Netmap στον πυρήνα ώστε να μπορούν να χρησιμοποιηθούν τα εργαλεία του. Η εντολή αυτή θα πρέπει να εκτελείται μετά από κάθε επανεκκίνηση του host συστήματος ώστε να είναι το module του Netmap φορτωμένο στη μνήμη.

Η επιβεβαίωση ότι το Netmap έχει φορτωθεί και μπορούν τα εργαλεία του να χρησιμοποιηθούν γίνεται ως εξής:
ls /dev/netmap

```
akoutsoni@clab:~$ ls /dev/netmap
/dev/netmap
```

Εικόνα 13: *Επιβεβαίωση φόρτωσης Netmap*

Τέλος, καθώς τα containers δεν έχουν δικό τους πυρήνα αλλά μοιράζονται με το λειτουργικό του host συστήματος τον δικό του, δεν έχουν πρόσβαση στον Netmap driver του πυρήνα. Θα πρέπει σε κάθε τοπολογία εντός του Containerlab να γίνεται bind-mount το αρχείο /dev/netmap του κάθε container με βάση το pktgen-container image με το /dev/netmap του host. Με τον τρόπο αυτό κατά την χρήση του το pkt-gen μπορεί να λειτουργήσει κανονικά έχοντας πρόσβαση στα αρχεία του /dev/netmap σαν αυτό να υπήρχε κανονικά εντός του container.

7. Εργαλεία και Προγράμματα

Στο παρόν κεφάλαιο παρουσιάζονται ορισμένα πρόσθετα εργαλεία και μηχανισμοί που χρησιμοποιήθηκαν συμπληρωματικά για την ανάλυση και την ορθή λειτουργία των προσομοιώσεων στο περιβάλλον του Containerlab. Πρόκειται για δύο εφαρμογές παρακολούθησης δικτυακής κίνησης (Wireshark και Edgeshark) και δύο εργαλεία για την εικονική διασύνδεση κόμβων στο στρώμα ζεύξης δεδομένων (Open vSwitch και Linux bridges).

7.1 Wireshark

Το Wireshark είναι ένας δωρεάν, ανοιχτού κώδικα (open-source) αναλυτής πακέτων. Χρησιμοποιείται για την ανάλυση ενός δικτύου, την ανίχνευση δυσλειτουργιών και ζητημάτων ασφαλείας, καθώς και για την επιβεβαίωση της λειτουργίας ενός δικτύου και για εκπαιδευτικούς σκοπούς. Υποστηρίζεται από τα περισσότερα λειτουργικά συστήματα συμπεριλαμβανομένων των Windows, Linux, Unix, macOS (από την έκδοση 11 και έπειτα). Αποτελεί ένα παθητικό εργαλείο διάγνωσης και ανάλυσης, το οποίο δεν μπορεί να επηρεάσει το δίκτυο ή να προειδοποιήσει για ενδεχόμενους κινδύνους και σφάλματα, παρά μόνο να δράσει διαγνωστικά και να βοηθήσει το χρήστη σε αυτές τις διαδικασίες. [63, 64, 86]
Οι βασικότερες παροχές του Wireshark είναι οι εξής:

- Καταγραφή πακέτων σε πραγματικό χρόνο από μια διεπαφή.
  Το Wireshark υποστηρίζει την καταγραφή πακέτων από ένα πλήθος τύπων δικτύων, όπως Ethernet, Wi-Fi, WLAN, Bluetooth, USB.

- Άνοιγμα και ανάγνωση αρχείων που περιέχουν καταγεγραμμένα δεδομένα τα οποία έχουν ληφθεί με άλλα προγράμματα καταγραφής πακέτων, όπως tcpdump/WinDump.

- Λεπτομερής παρουσίαση πληροφοριών πρωτοκόλλου για τα πακέτα που καταγράφονται.

- Αποθήκευση των καταγεγραμμένων πακέτων και των δεδομένων τους.

- Εξαγωγή πακέτων σε διάφορες μορφές αρχείων καταγραφής.
  Η προεπιλεγμένη μορφή αρχείου για είναι η *pcapng,* ενώ ευρέως χρησιμοποιούμενη είναι και η *pcap.*

- Επιλεγμένη εμφάνιση πακέτων με βάση πολλά κριτήρια / φίλτρα.

- Στοχευμένη αναζήτηση πακέτων με βάση πολλά κριτήρια / φίλτρα.

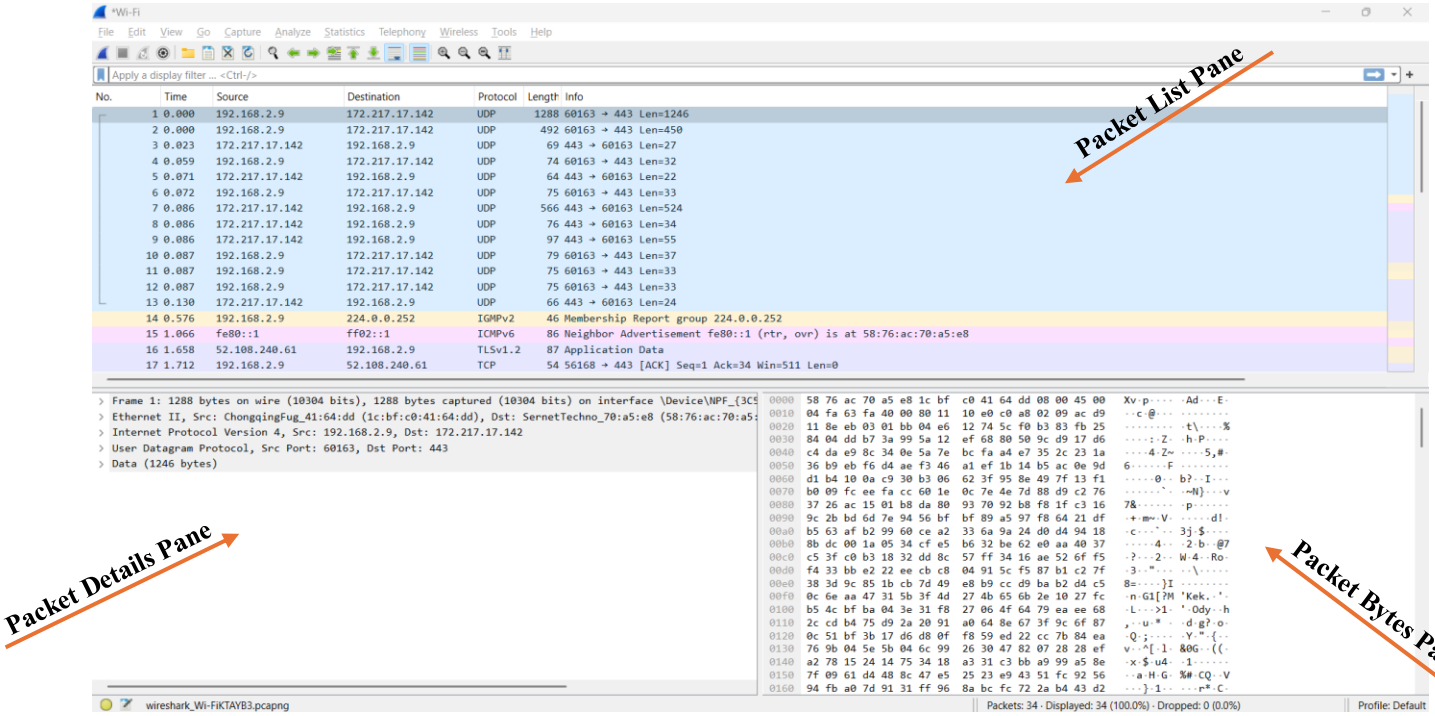
- Δημιουργία διαφόρων στατιστικών στοιχείων.

Στο UI του Wireshark η παρουσίαση των πακέτων γίνεται με τη χρήση των εξής 3 βασικών παραθύρων:

- Packet List Pane → Εμφανίζει μια σύνοψη κάθε πακέτου που έχει καταγραφεί. Επιλέγοντας ένα πακέτο σε αυτό το παράθυρο, εμφανίζονται για αυτό πληροφορίες στα άλλα δύο παράθυρα.
- Packet Details Pane → Παρουσιάζει λεπτομέρειες για το τρέχον επιλεγμένο πακέτο. Περιλαμβάνει σε μορφή δέντρου μια σύνοψη των πρωτοκόλλων του πακέτου και των πεδίων τους.
- Packet Bytes Pane → Αναπαριστά το τρέχον επιλεγμένο πακέτο σε δεκαεξαδική μορφή.

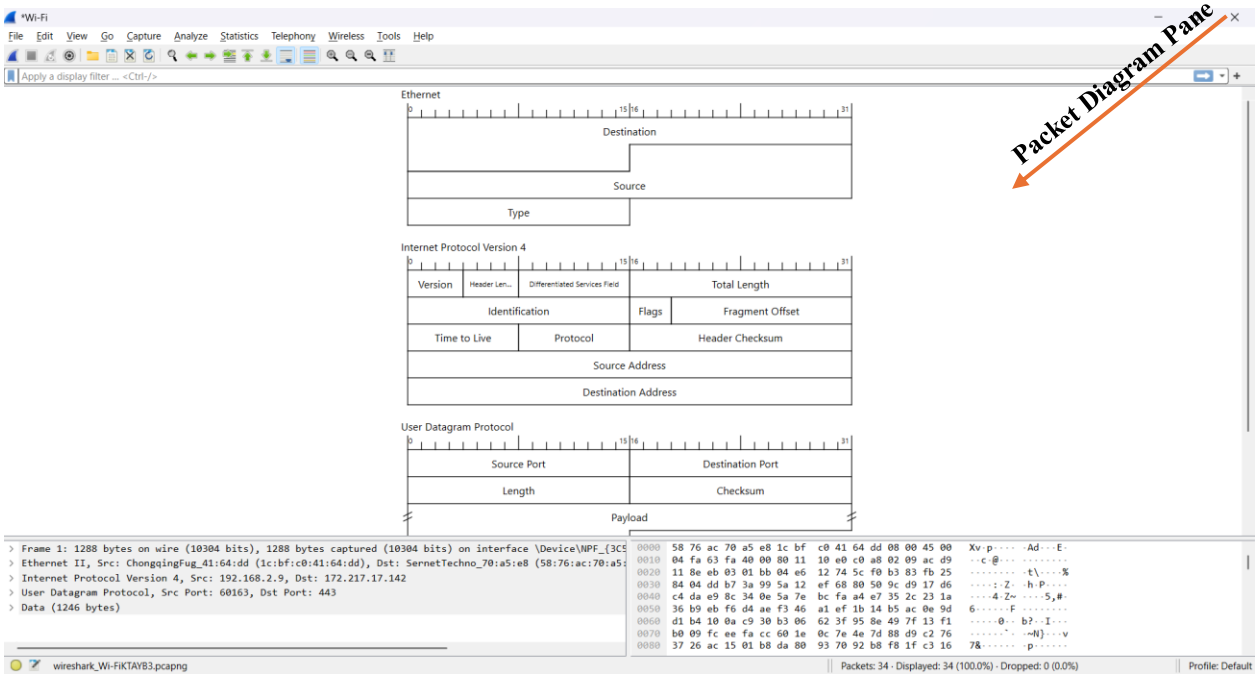
Παρέχεται και ένα τέταρτο παράθυρο, το οποίο όμως για να εμφανιστεί λαμβάνει τη θέση κάποιου από τα παραπάνω στην αρχική οθόνη. Ονομάζεται Packet Diagram Pane και απεικονίζει τα πακέτα σαν διαγράμματα.

Στην παρακάτω εικόνα φαίνεται η αρχική οθόνη του Wireshark με τα τρία βασικά του παράθυρα κατά τη διάρκεια μιας καταγραφής.



Εικόνα 14: *Αρχική οθόνη Wireshark με τα τρία βασικά του παράθυρα*

Σε περίπτωση που επιλεχθεί το Packet Diagram Pane στη θέση του Packet List Pane η αρχική οθόνη έχει την εξής μορφή:



Εικόνα 15: *Αρχική οθόνη Wireshark με το Packet Diagram Pane*

Όπως προαναφέρθηκε το Wireshark δίνει τη δυνατότητα εφαρμογής φίλτρων είτε κατά την καταγραφή πακέτων είτε έπειτα κατά την αναζήτηση και εμφάνιση τους. Ακολουθούν κάποια από τα κυριότερα φίλτρα, πολλά από τα οποία θα χρησιμοποιούνται κατά την υλοποίηση των δικτυακών τοπολογιών στο Containerlab του επόμενου κεφαλαίου.

Φίλτρα καταγραφής (Capture Filters)

- [src|dst] host <host> → Προσδιορισμός IP διεύθυνσης πηγής ή προορισμού.
- ether [src|dst] host <ehost> → Προσδιορισμός MAC διεύθυνσης πηγής ή προορισμού.
- gateway host <host> → Προσδιορισμός προκαθορισμένης πύλης.
- [tcp|udp] [src|dst] port <port> → Προσδιορισμός πρωτοκόλλου (tcp ή udp) και θύρας πηγής ή προορισμού.
- ip|ether proto <protocol> → Επιλογή στρώματος ζεύξης δεδομένων (Ethernet Layer) ή στρώματος δικτύου (IP Layer)

Φίλτρα εμφάνισης (Display Filters)

Αποτελούνται από την παράμετρο που επιδιώκεται να προσδιορισθεί και την τιμή που της αποδίδεται, χωρισμένες από έναν ενδιάμεσο τελεστή που δηλώνει τη μεταξύ τους σχέση. Οι παράμετροι που προσδιορίζονται μπορεί να είναι οποιαδήποτε παράμετρος περιλαμβάνεται στα πεδία του πακέτου και έχει το αντίστοιχο όνομα και συμβολισμό. Οι βασικότεροι τελεστές είναι:

- eq ή == → Δηλώνει ότι υπάρχει ισότητα.
- ne ή != → Δηλώνει ότι δεν υπάρχει ισότητα.

- gt ή > → Σχέση «μεγαλύτερο από».
- lt ή < → Σχέση «μικρότερο από».

Η εγκατάσταση του είναι αρκετά απλή. Για περιβάλλοντα Windows και macOS αρκεί εύρεση η λήψη της κατάλληλης για το εκάστοτε σύστημα έκδοσης στον επίσημο ιστότοπο https://www.wireshark.org/download.html. Για περιβάλλοντα UNIX χρησιμοποιούνται οι παρακάτω εντολές για κάθε έκδοση:

- Red Hat → *yum install wireshark wireshark-qt*
- Debian, Ubuntu → *apt install wireshark*
- FreeBSD → *pkg_add -r wireshark*

Στο πλαίσιο της παρούσας διπλωματικής, το Wireshark χρησιμοποιείται για την επαλήθευση και την ανάλυση της λειτουργίας των τοπολογιών. Μέσα από αυτό μπορεί να εξετάζεται σε πραγματικό χρόνο η ορθή ανταλλαγή μηνυμάτων με βάση τα πρωτόκολλα δρομολόγησης (π.χ. RIP, OSPF, BGP, VXLAN), αν τα πακέτα VXLAN είναι σωστά δομημένα και αν η κίνηση ECMP κατανέμεται ορθά.

[86]

7.2 Edgeshark

Το Edgeshark αποτελεί ένα εργαλείο ανάλυσης και απεικόνισης της δικτυακής κίνησης σε περιβάλλοντα με containers. Αναπαριστά τις συνδέσεις μεταξύ των containers και με τον host με τη μορφή μιας «εικονικής καλωδίωσης» και παρέχει πληροφορίες για διάφορες παραμέτρους του δικτύου, όπως διευθύνσεις IP και MAC, IP δρομολόγηση και ρυθμίσεις DNS. Επιπλέον προσφέρει τη δυνατότητα ενσωμάτωσης του Wireshark μέσω ενός πρόσθετου (plugin) πακέτου, καθιστώντας έτσι εφικτή την καταγραφή κίνησης σε πραγματικό χρόνο.

Το Edgeshark είναι σχεδιασμένο ως μια web-based εφαρμογή και το user interface της (UI) είναι προσβάσιμο μέσα από έναν HTML5 περιηγητή ιστού (browser). Η ιδιότητα αυτή εξαλείφει την ανάγκη εγκατάστασης desktop εφαρμογής και καθιστά δυνατή την ανάλυση της κίνησης από οποιοδήποτε σύστημα ακόμα και απομακρυσμένο αρκεί αυτό να έχει πρόσβαση στο εκάστοτε container ή host μηχάνημα. Διευκολύνεται έτσι η παρακολούθηση της κίνησης και η πρόσβαση στην ίδια πληροφορία σε πραγματικό χρόνο από ομάδες χρηστών, με αποτέλεσμα να επιταχύνεται και να διευκολύνεται η συνεργασία τους και η παράλληλη εργασία. Επιπλέον μπορούν να αποθηκευτούν τοπικά κάποια τμήματα της πληροφορίας που παρέχει το Edgeshark, όπως οι καταγραφές πακέτων, για περαιτέρω ανάλυση.

Διανέμεται ως ένα container image, το οποίο βρίσκεται στο repository https://github.com/siemens/edgeshark και αποτελείται από δύο υπηρεσίες σε μορφή

container (Ghostwire και Packetflix). Το Ghostwire είναι υπεύθυνο για την ανακάλυψη των δικτυακών συνδέσεων και παραμετροποιήσεων, ενώ το Packetflix για την προώθηση αυτών των πληροφοριών στο UI. Παρέχεται υποστήριξη μόνο για συστήματα Linux, για αρχιτεκτονικές linux/amd64 και linux/arm64, αλλά η απομακρυσμένη σύνδεση μπορεί να γίνει από οποιουδήποτε τύπου λειτουργικό σύστημα εφόσον υπάρχει πρόσβαση στον Linux host που φιλοξενεί το Edgeshark. Η ενσωμάτωση του σε έναν host για ορθή χρήση συνδυαστικά με το Containerlab γίνεται με την εξής εντολή [25]:

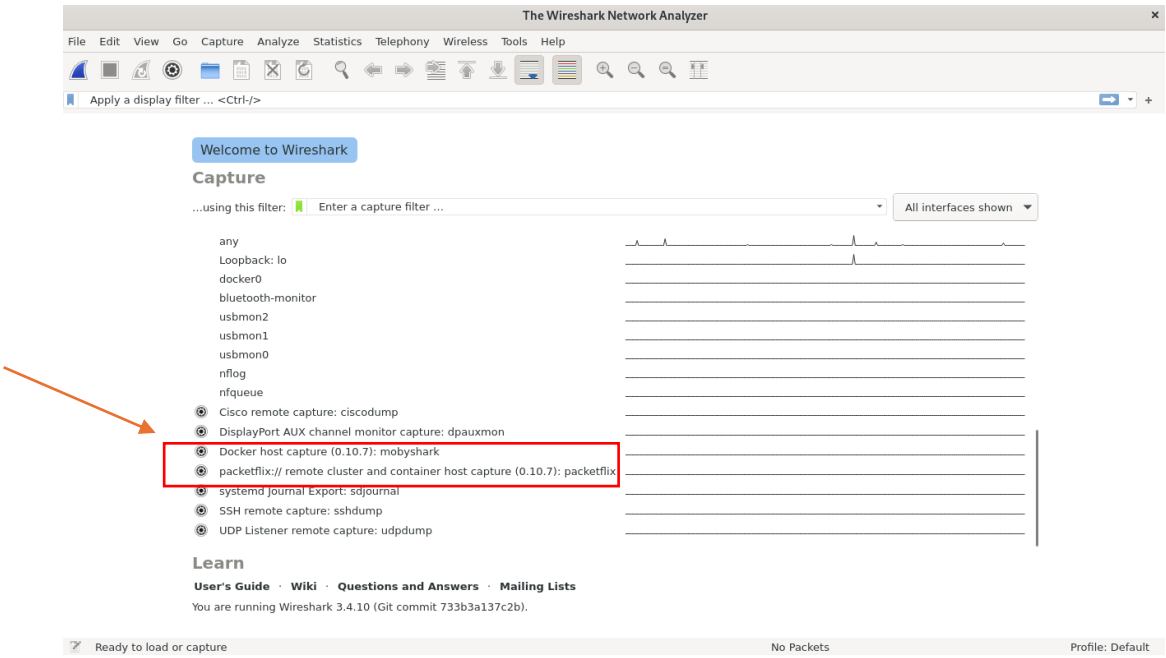
curl -sL https://github.com/siemens/edgeshark/raw/main/deployments/wget/docker-compose.yaml | DOCKER_DEFAULT_PLATFORM= docker compose -f - up -d

Για το Wireshark plugin θα πρέπει αρχικά να υπάρχει ήδη εγκατεστημένο το Wireshark και να έχει προστεθεί ο χρήστης στο Wireshark group με την εξής εντολή

sudo gpasswd -a $USER wireshark

όπου $USER το username του χρήστη.
Έπειτα απαιτείται λήψη και εγκατάσταση του κατάλληλου για το σύστημα plugin πακέτου από τον ακόλουθο σύνδεσμο https://github.com/siemens/cshargextcap/releases/latest .

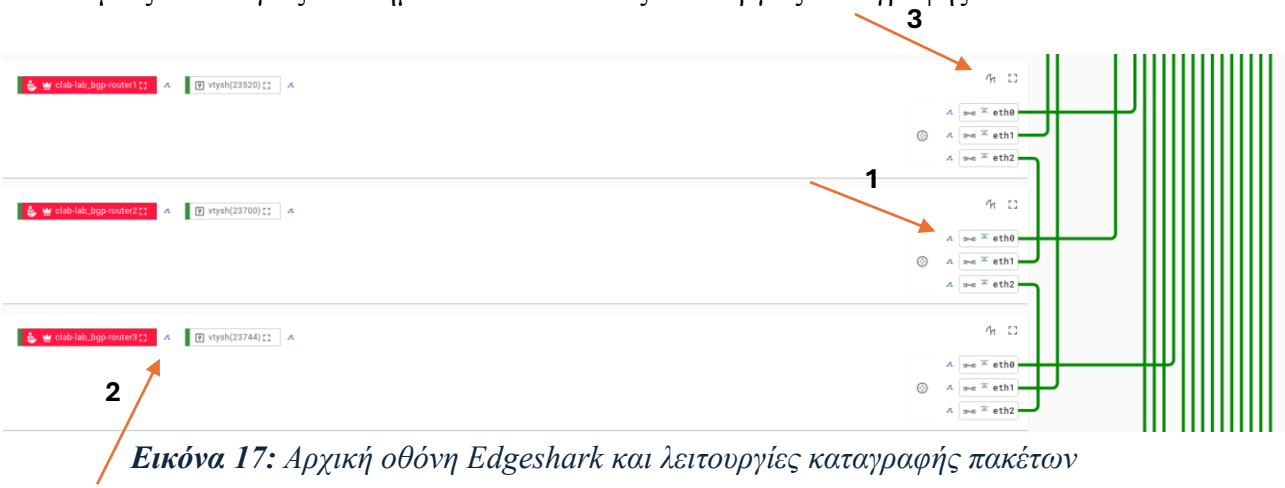
Μπορεί να επιβεβαιωθεί ότι το plugin πακέτο εγκαταστάθηκε σωστά και ανιχνεύθηκε από το Wireshark ανοίγοντας το Wireshark και ελέγχοντας ότι στην αρχική του έχουν προστεθεί οι εξής δύο external capture targets:



Εικόνα 16: *Επιβεβαίωση εγκατάστασης plugin πακέτου του Edgeshark*

Αφού ολοκληρωθεί η παραπάνω διαδικασία η πρόσβαση στο UI του Edgeshark γίνεται μέσω της θύρας 5001 από τον σύνδεσμο *http://0.0.0.0:5001*.

Ενδεικτικά στο παρακάτω σχήμα φαίνονται οι συνδέσεις μεταξύ των δρομολογητών μιας τοπολογίας και σημειώνονται κάποιες λειτουργίες καταγραφής πακέτων.



Εικόνα 17: *Αρχική οθόνη Edgeshark και λειτουργίες καταγραφής πακέτων*

1) Δυνατότητα εκκίνησης μιας καταγραφής σε μια διεπαφή ενός container.
2) Δυνατότητα εκκίνησης μιας καταγραφής σε όλες τις διεπαφές ενός container.
3) Δυνατότητα εκκίνησης μιας καταγραφής σε όσες και όποιες από τις διεπαφές ενός container επιλεχθούν.

Παρά την απουσία όλων των προηγμένων δυνατοτήτων του Wireshark, όπως στατιστικά γραφήματα και εξελιγμένα φίλτρα, το Edgeshark αποτελεί ένα ιδιαίτερα αξιόλογο και εύχρηστο εργαλείο για ανάλυση δικτυακής κίνησης σε περιβάλλοντα όπως το Containerlab, ενώ το plugin του Wireshark αντιμετωπίζει σε μεγάλο βαθμό αυτές τις ελλείψεις αποτελεσματικά συνδυάζοντας δύο χρήσιμα εργαλεία σε ένα ευέλικτο και αποδοτικό μέσο. Η απουσία συμβατότητας με όλα τα λειτουργικά συστήματα ωστόσο περιορίζει σημαντικά την ευρεία χρήση του.

[88, 89]

7.3 Linux Bridges

Ο όρος Linux Bridge αναφέρεται σε έναν εικονικό μηχανισμό του πυρήνα του Linux, ο οποίος λειτουργεί στο στρώμα ζεύξης δεδομένων και χρησιμοποιείται για τη διασύνδεση δύο κόμβων. Λειτουργεί σαν ένας μεταγωγέας Ethernet αλλά υλοποιείται εξ' ολοκλήρου στο λειτουργικό του συστήματος.

Η λειτουργία μιας Linux bridge βασίζεται στην εκμάθηση διευθύνσεων (MAC Learning) και την αποθήκευση τους σε έναν πίνακα προώθησης που ονομάζεται Forwarding Database (FDB). Ο πίνακας αυτός ενημερώνεται κάθε φορά που η γέφυρα δέχεται ένα πακέτο από μια διεπαφή ενός κόμβου και μαθαίνει την αντιστοιχία IP – MAC. Συνεπώς αν η Linux bridge λάβει πακέτα των οποίων η MAC υπάρχει καταχωρημένη στην FDB του τα προωθεί κατάλληλα, ενώ σε αντίθετη

περίπτωση γίνεται flooding και αποστέλλεται σε όλες τις διεπαφές, εκτός από εκείνη από την οποία προέρχεται. Παρέχει μεταξύ άλλων υποστήριξη για το πρωτόκολλο STP, για VLAN Tagging και VXLAN υλοποιήσεις.

Οι Linux bridges προσφέρουν απλότητα, καθώς είναι ήδη ενσωματωμένες ως μηχανισμός στον πυρήνα του Linux. Λειτουργούν σαν φυσική υποδομή διότι για τα ανώτερα επίπεδα του OSI (Layer 3 και πάνω) είναι διαφανείς. Στην περίπτωση του Containerlab μια Linux bridge λειτουργεί ως ένα σταθερό στοιχείο, το οποίο του προσφέρει την ευελιξία να μεταβάλλει την τοπολογία χωρίς να επηρεάζει την γέφυρα και να απασχολείται με την διαχείριση της.

Για την χρήση τους απαιτείται η εγκατάσταση των απαραίτητων πακέτων με την εντολή *sudo apt install bridge-utils* ώστε να καταστεί εφικτός ο χειρισμός και η παραμετροποίηση των γεφυρών στη συνέχεια με την εντολή *brctl* η οποία χρησιμοποιείται για την ρύθμιση, τον έλεγχο και την διαμόρφωση των Linux bridges.

[25, 91]

Οι βασικότερες υπο-εντολές της *brctl* είναι οι εξής [90]:

- brctl addbr [bridge_name] → Δημιουργεί την γέφυρα με το αντίστοιχο όνομα
- brctl delbr [bridge_name] → Διαγράφει την γέφυρα με το αντίστοιχο όνομα
- brctl addif [bridge_name] [interface] → Προσθέτει στην γέφυρα την αντίστοιχη διεπαφή
- brctl delif [bridge_name] [interface] → Διαγράφει από την γέφυρα την αντίστοιχη διεπαφή
- brctl show / brctl show [bridge_name] → Παρουσιάζει πληροφορίες για τις υπάρχουσες γέφυρες / για την συγκεκριμένη γέφυρα
- brctl showmacs [bridge_name] → Παρουσιάζει μια λίστα με τις διευθύνσεις MAC που γνωρίζει η συγκεκριμένη γέφυρα
- brctl stp [bridge_name] {on/off} → Ενεργοποιεί/Απενεργοποιεί το πρωτοκόλλο stp στην συγκεκριμένη γέφυρα
- brctl showstp [bridge_name] → Παρουσιάζει πληροφορίες σχετικά με τη λειτουργεία του stp στην συγκεκριμένη γέφυρα
- brctl setageing [bridge_name] [time] → Ορίζει το χρόνο time σε δευτερόλεπτα μετά από τον οποίο εάν δεν έχει ληφθεί κάποιο πλαίσιο από μια διεύθυνση MAC τότε η εγγραφή της γέφυρας για την διεύθυνση αυτή λήγει και διαγράφεται από τη βάση δεδομένων προώθησης (fdb).
- brctl setbridgeprio [bridge_name] [priority] → Ορίζει την προτεραιότητα της γέφυρας σε priority
- brctl setportprio [bridge_name] [port] [priority] → Ορίζει την προτεραιότητα μιας θύρας της γέφυρας σε priority
- brctl setpathcost [bridge_name] [port] [cost] → Ορίζει το κόστος ενός μονοπατιού μέσω μιας θύρας της γέφυρας σε cost

7.4 Open vSwitch

Το Open vSwitch (OVS) είναι ένας ανοικτού κώδικα πολυεπίπεδος (multilayer) εικονικός μεταγωγέας. Υποστηρίζει την κλασσική λειτουργία ενός μεταγωγέα στο στρώμα ζεύξης δεδομένων αλλά και πρόσθετα χαρακτηριστικά, όπως QoS (Quality of Service), Netflow, sFlow και OpenFlow.

Η αρχιτεκτονική του OVS βασίζεται σε δύο κύρια στοιχεία. Το πρώτο και σημαντικότερο είναι ο δαίμονας ovs-vswitchd, ο οποίος λειτουργεί στο userspace του λειτουργικού και είναι ο ίδιος για κάθε σύστημα. Το δεύτερο στοιχείο είναι ένα datapath kernel module το οποίο λειτουργεί στον πυρήνα του λειτουργικού συστήματος. Το datapath kernel module λαμβάνει αρχικά τα πακέτα είτε από μια φυσική είτε από μια εικονική κάρτα δικτύου. Ο τρόπος με τον οποίο θα διαχειριστούν τα πακέτα καθορίζεται από τον δαίμονα ovs-vswitchd. Εάν το datapath kernel module έχει λάβει εκ των προτέρων οδηγίες για το πως να χειριστεί ένα συγκεκριμένο είδος κίνησης που λαμβάνει τότε απλά εκτελεί αυτές τις οδηγίες. Εάν όχι τότε προωθεί τα πακέτα στον ovs-vswitchd, αυτός προσδιορίζει τις απαραίτητες ενέργειες και έπειτα προωθεί τα πακέτα και τις νέες οδηγίες πίσω στο datapath kernel module, το οποίο εκτελεί τώρα τις εντολές του δαίμονα και τις αποθηκεύει στην μνήμη του για μελλοντική διαχείριση παρόμοιας κίνησης.

Συγκριτικά με τις Linux bridges, το OVS προσφέρει ένα πιο ευέλικτο περιβάλλον, ειδικά για μεγάλης κλίμακας δίκτυα. Αυτό συμβαίνει διότι το OVS παρέχει τη δυνατότητα ενσωμάτωσης λειτουργιών, όπως το OpenFlow, ιδιαίτερα εύχρηστων σε περιβάλλοντα Data Center και SDN (Software-defined networking) αρχιτεκτονικές. Σε απλά σενάρια χρήσης του OVS σαν γέφυρα στο στρώμα ζεύξης δεδομένων δεν υπάρχουν ουσιαστικές διαφορές σε επίπεδο λειτουργικότητας σε σχέση με μια Linux bridge.
Η εγκατάσταση του σε συστήματα Ubuntu γίνεται με την εξής εντολή:

sudo apt install openvswitch-switch

Ακολουθούν κάποιες από τις βασικότερες εντολές του Openvswitch:

- ovs-vsctl add-br [bridge_name] → Δημιουργεί την γέφυρα με το αντίστοιχο όνομα
- ovs-vsctl del-br [bridge_name] → Διαγράφει την γέφυρα με το αντίστοιχο όνομα
- ovs-vsctl add-port [bridge_name] [interface] → Προσθέτει μια διεπαφή σε μια γέφυρα
- ovs-vsctl add-port [bridge_name] [interface] tag=[VLAN number] → Αντιστοιχίζει μια διεπαφή μιας γέφυρας με ένα VLAN
- ovs-vsctl show → Εμφανίζει μια σύνοψη με την τρέχουσα κατάσταση των γεφυρών, θυρών και διεπαφών που διαχειρίζεται το OVS
- ovs-vsctl list-br → Εμφανίζει μια λίστα με όλες τις υπάρχουσες γέφυρες
- ovs-vsctl list-ports [bridge_name] → Εμφανίζει ως λίστα τις θύρες μιας γέφυρας [92, 93]

8. Δικτυακές Τοπολογίες σε Containerlab

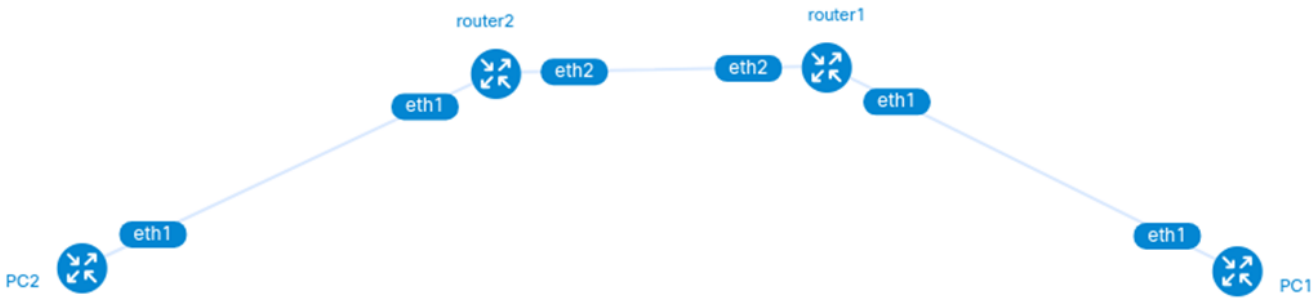
Στο παρόν κεφάλαιο παρουσιάζεται μια σειρά από δικτυακές τοπολογίες, οι οποίες υλοποιήθηκαν στο περιβάλλον του Containerlab. Σε αυτές αξιοποιήθηκαν τα πρωτόκολλα και οι μηχανισμοί που αναλύθηκαν στο κεφάλαιο 5 (RIP, OSPF, BGP, VXLAN, EVPN, ECMP, μεταγωγείς), η χρήση των οποίων γίνεται κατά κόρον σε περιβάλλοντα με εικονικές μηχανές, με στόχο να διερευνηθεί και να επιβεβαιωθεί η ορθή λειτουργία τους στο περιβάλλον του Containerlab με χρήση containers και να εξεταστούν οι απαιτούμενες προσαρμογές για την επίτευξη αυτού του στόχου. Όλα τα αρχεία και οι φάκελοι των τοπολογιών που δημιουργήθηκαν στα πλαίσια υλοποίησης των παρακάτω παραδειγμάτων μπορούν να βρεθούν ολοκληρωμένα στο εξής repository που δημιουργήθηκε για την παρούσα διπλωματική: https://github.com/annakts/Containerlab-Topologies.git [94]

8.1 LAB1

Η τοπολογία lab1 παρουσιάζει ένα απλό σενάριο στατικής δρομολόγησης. Αποτελείται από δύο τερματικούς κόμβους (PC1, PC2) και δύο δρομολογητές (router1, router2). Τα PC1, PC2 έχουν ως βάση τους το image wbitt/network-multitool και λειτουργούν ως απλοί hosts ενώ οι δρομολογητές υλοποιούνται με το FRR image ώστε να παραμετροποιηθούν κατάλληλα.

Ο φάκελος της τοπολογίας περιλαμβάνει 4 αρχεία, το YAML αρχείο με την περιγραφή της και τρία εκτελέσιμα αρχεία, τα PCs.sh, labdeploy.sh και labreconfigure.sh και έναν ακόμη φάκελο config με τα αρχεία παραμετροποίησης των δρομολογητών (.conf) και το αρχείο με τους δαίμονες του FRR. Στο αρχείο PCs.sh περιλαμβάνονται οι απαραίτητες εντολές για την ρύθμιση των διεπαφών των PC1, PC2 και τον ορισμό των στατικών διαδρομών. Τα labdeploy.sh και labreconfigure.sh χρησιμοποιούνται βοηθητικά για την αυτοματοποίηση της διαδικασίας ανάπτυξης της τοπολογίας και επαναδημιουργίας της σε περίπτωση αλλαγών στα αρχεία της αντίστοιχα. Με τον τρόπο αυτό δεν απαιτείται η σύνταξη και η εκτέλεση των κατάλληλων εντολών κάθε φορά για τις παραπάνω διαδικασίες, παρά μόνο η εκτέλεση του αντίστοιχου αρχείου, διευκολύνοντας έτσι τον χρήστη.

Μια σχηματική αναπαράσταση της τοπολογίας μπορεί να εμφανιστεί με χρήση της εντολής *sudo graph -t lab1.yml* και είναι η εξής:



Εικόνα 18: *Σχηματική αναπαράσταση τοπολογίας LAB1*

Οι διεπαφές eth1 του PC1 και eth1 του router1 ανήκουν στο υποδίκτυο 192.168.1.0/24 και λαμβάνουν IP 192.168.1.2 και 192.168.1.1 αντίστοιχα. Ομοίως οι διεπαφές eth1 του PC2 και eth1 του router2 ανήκουν στο υποδίκτυο 192.168.2.0/24 και λαμβάνουν IP 192.168.2.2 και 192.168.2.1 αντίστοιχα. Οι ενδιάμεσες διεπαφές eth2 των router1 και router2 που συνδέουν τους δύο δρομολογητές μεταξύ τους ανήκουν στο υποδίκτυο 172.17.17.0/30 και είναι 172.17.17.1 για τον router1 και 172.17.17.2 για τον router2. Για την επίτευξη της επικοινωνίας μεταξύ των δύο τερματικών κόμβων ορίζεται στον κάθε ένα στατική διαδρομή για το υποδίκτυο του άλλου μέσω της άμεσα συνδεδεμένης σε αυτόν διεπαφής του αντίστοιχου δρομολογητή. Συγκεκριμένα στο PC1 ορίζεται στατική διαδρομή για το υποδίκτυο 192.168.2.0/24 μέσω της 192.168.1.1 του router1 και στο PC2 ορίζεται στατική διαδρομή για το υποδίκτυο 192.168.1.0/24 μέσω της 192.168.1.1.

Ο καθορισμός των IP διευθύνσεων και των στατικών διαδρομών για τα PCs στο εκτελέσιμο αρχείο PCs.sh γίνεται με τη χρήση εντολών της μορφής

sudo docker exec clab-lab1-PC[X] [command] ,

όπου *[X]* ο αριθμός του εκάστοτε container (PC1, PC2) και όπου *[command]* η κατάλληλη εντολή συμβατή με το σύνολο εντολών του wbitt/network-multitool. Κατά την εκτέλεση του αρχείου PCs.sh οι εντολές αυτές εκτελούνται διαδοχικά εντός του κάθε κόμβου χωρίς να χρειάζεται είσοδος στη γραμμή εντολών του.
Για τους δρομολογητές η παραμετροποίηση γίνεται με τη χρήση εξωτερικών αρχείων router1.conf και router2.conf και bind-mount αυτών με το βασικό αρχείο διαμόρφωσης του FRR, το /etc/frr/frr.conf. Εντός αυτών των αρχείων παραμετροποίησης η μορφή των εντολών είναι ίδια με αυτή στο vtysh μιας FRR συσκευής.

Μετά την ανάπτυξη (deployment) της τοπολογίας, η οποία λόγω του μικρού μεγέθους των images διαρκεί πολύ λίγο, οι στατικές διαδρομές έχουν καταχωρηθεί σωστά όπως φαίνεται με την εκτέλεση της εντολής *ip route show* στα PC και της εντολής *show ip route* στο vtysh των routers. Για τα PC η εντολή *ip route show* μπορεί και να εκτελεστεί χωρίς να απαιτείται είσοδος στη γραμμή εντολών του κάθε PC ως εξής:

```
akoutsoni@clab:~/topologies/containerlab/lab1$ sudo docker exec -it clab-lab1-PC1 ip route show
[sudo] password for akoutsoni:
default via 172.20.20.1 dev eth0
172.20.20.0/24 dev eth0 proto kernel scope link src 172.20.20.12
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.2
192.168.2.0/24 via 192.168.1.1 dev eth1
akoutsoni@clab:~/topologies/containerlab/lab1$ sudo docker exec -it clab-lab1-PC2 ip route show
default via 172.20.20.1 dev eth0
172.20.20.0/24 dev eth0 proto kernel scope link src 172.20.20.4
192.168.1.0/24 via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth1 proto kernel scope link src 192.168.2.2
```

Για τους δρομολογητές χρειάζεται να προηγηθεί είσοδος σε vtysh μέσω της εντολής *sudo docker exec -it clab-lab1-router1 vtysh* και έπειτα να ελεγχθεί ο πίνακας διαδρομών:

```
router1# show ip route
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIP, O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 172.20.20.1, eth0, 02:11:58
C>* 172.17.17.0/30 is directly connected, eth2, 02:11:58
L>* 172.17.17.1/32 is directly connected, eth2, 02:11:58
C>* 172.20.20.0/24 is directly connected, eth0, 02:11:58
L>* 172.20.20.18/32 is directly connected, eth0, 02:11:58
C>* 192.168.1.0/24 is directly connected, eth1, 02:11:58
L>* 192.168.1.1/32 is directly connected, eth1, 02:11:58
S>* 192.168.2.0/24 [1/0] via 172.17.17.2, eth2, weight 1, 02:11:58
```

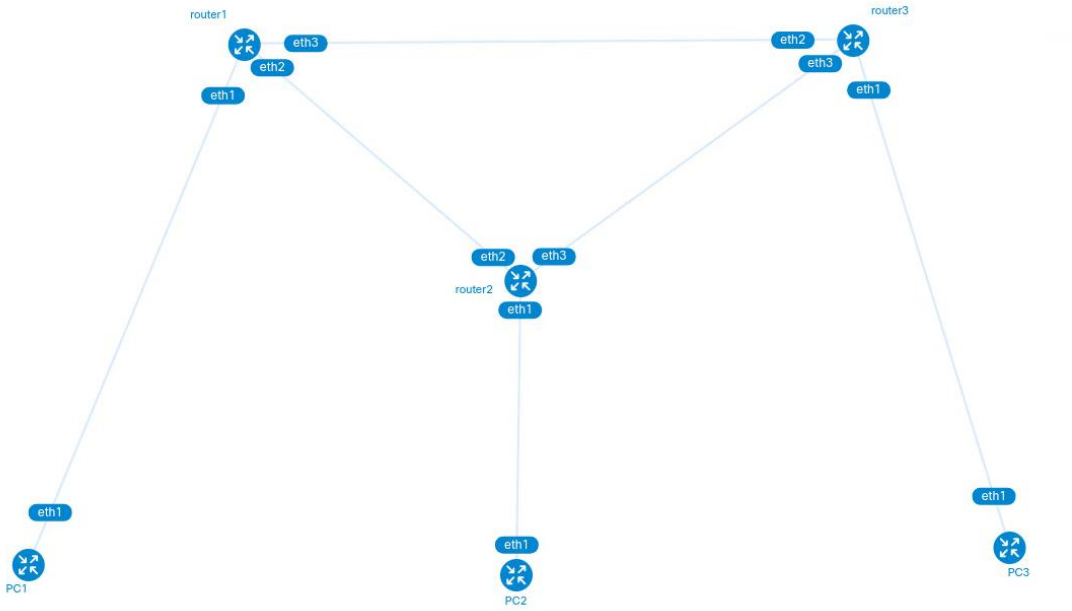
```
router2# show ip route
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIP, O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 172.20.20.1, eth0, 03:28:57
C>* 172.17.17.0/30 is directly connected, eth2, 03:28:57
L>* 172.17.17.2/32 is directly connected, eth2, 03:28:57
C>* 172.20.20.0/24 is directly connected, eth0, 03:28:57
L>* 172.20.20.19/32 is directly connected, eth0, 03:28:57
S>* 192.168.1.0/24 [1/0] via 172.17.17.1, eth2, weight 1, 03:28:57
C>* 192.168.2.0/24 is directly connected, eth1, 03:28:57
L>* 192.168.2.1/32 is directly connected, eth1, 03:28:57
```

Τέλος επιβεβαιώνεται η επικοινωνία των δύο PCs μέσω ping.

```
akoutsoni@clab:~/topologies/containerlab/lab1$ sudo docker exec -it clab-lab1-PC1 ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=62 time=0.093 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=62 time=0.088 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=62 time=0.072 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=62 time=0.074 ms
64 bytes from 192.168.2.2: icmp_seq=5 ttl=62 time=0.091 ms
^C
--- 192.168.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.072/0.083/0.093/0.008 ms
```

8.2 LAB_RIP

Η τοπολογία lab_rip υλοποιεί ένα απλό δικτυακό πλέγμα τριών δρομολογητών για τη λειτουργία των οποίων χρησιμοποιείται δυναμική δρομολόγηση με χρήση του πρωτοκόλλου RIP. Οι τρεις δρομολογητές (router1, router2, router3) συνδέονται μεταξύ τους σε τοπολογία πλήρους πλέγματος (full-mesh) και κάθε ένας από αυτούς συνδέεται με έναν τελικό κόμβο (PC1, PC2, PC3). Ακολουθεί σχηματική αναπαράσταση της τοπολογίας:



Εικόνα 19: *Σχηματική αναπαράσταση τοπολογίας LAB_RIP*

Όπως και στο προηγούμενο παράδειγμα τα PC1, PC2, PC3 είναι υλοποιημένα με το image wbitt/network-multitool, ενώ οι δρομολογητές είναι FRR containers. Κάθε ένας από τους τρεις τελικούς κόμβους ανήκει σε διαφορετικό υποδίκτυο, ενώ σε κάθε περίπτωση στο ίδιο υποδίκτυο ανήκει αντίστοιχα και η διεπαφή του γειτονικού δρομολογητή που είναι άμεσα συνδεδεμένη με το PC, η οποία ορίζεται και ως προεπιλεγμένη πύλη για το κάθε PC. Ο PC1 και η eth1 του router1 ανήκουν στο 192.168.1.0/24, ο PC2 και η eth1 του router2 ανήκουν στο 192.168.2.0/24, Ο PC3 και η eth1 του router3 ανήκουν στο 192.168.3.0/24. Οι ενδιάμεσες διεπαφές των δρομολογητών που χρησιμοποιούνται για τις μεταξύ τους συνδέσεις ανήκουν ανά δύο σε διαφορετικό υποδίκτυο. Συγκεκριμένα στη σύνδεση router1-router2 η διεπαφή eth2 και των δύο λαμβάνει IP από το 172.17.17.0/30, στη σύνδεση router2-router3 η διεπαφή eth3 και των δύο από το 172.17.17.8/30 και στη σύνδεση router1-router3 οι διεπαφές eth3 και eth2 αντίστοιχα ανήκουν στο 172.17.17.4/30.

Ιδιαίτερη προσοχή θα πρέπει να δοθεί στον ορισμό της προεπιλεγμένης πύλης. Κατά την ανάπτυξη της τοπολογίας ορίζεται από προεπιλογή από το Containerlab ως προκαθορισμένη πύλη κάθε container η bridged διεπαφή του host με διεύθυνση 172.20.20.1. Ένα πιθανό σφάλμα είναι να αμεληθεί η ύπαρξη ήδη προκαθορισμένης διαδρομής κατά την ρύθμιση των PCs, καθώς δεν είναι άμεσα εμφανές και ορισμένο

από τον χρήστη, και να χρησιμοποιηθεί η εντολή *ip route add default* , η οποία επιχειρεί να προσθέσει προεπιλεγμένη πύλη ενώ ήδη υπάρχει. Το αποτέλεσμα είναι να προκύψει η εξής προειδοποίηση από το Containerlab όταν εκτελείται η εντολή και να μην αποθηκευτεί η νέα προκαθορισμένη διαδρομή, όπως φαίνεται παρακάτω.

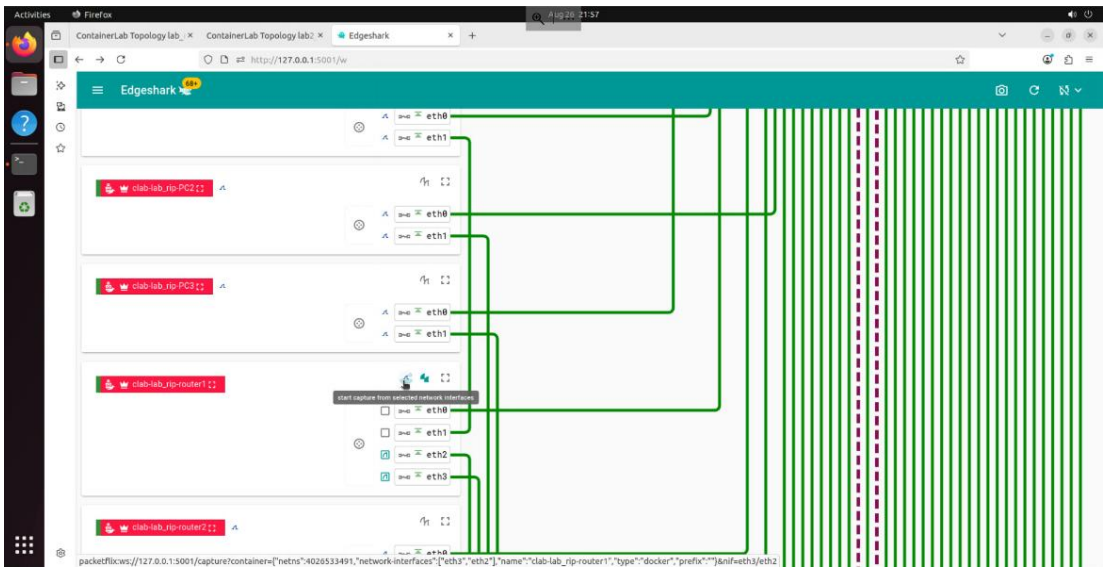
```
RTNETLINK answers: File exists
```

```
akoutsoni@clab:~/topologies/containerlab/lab_rip$ sudo docker exec clab-lab_rip-PC1 ip route show
default via 172.20.20.1 dev eth0
172.20.20.0/24 dev eth0 proto kernel scope link src 172.20.20.28
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.2
```

Για να αποφευχθεί αυτό θα πρέπει να χρησιμοποιηθεί η εντολή *ip route change default* , η οποία δεν επιδιώκει την δημιουργία προεπιλεγμένης πύλης από το μηδέν αλλά αλλάζει την ήδη υπάρχουσα.

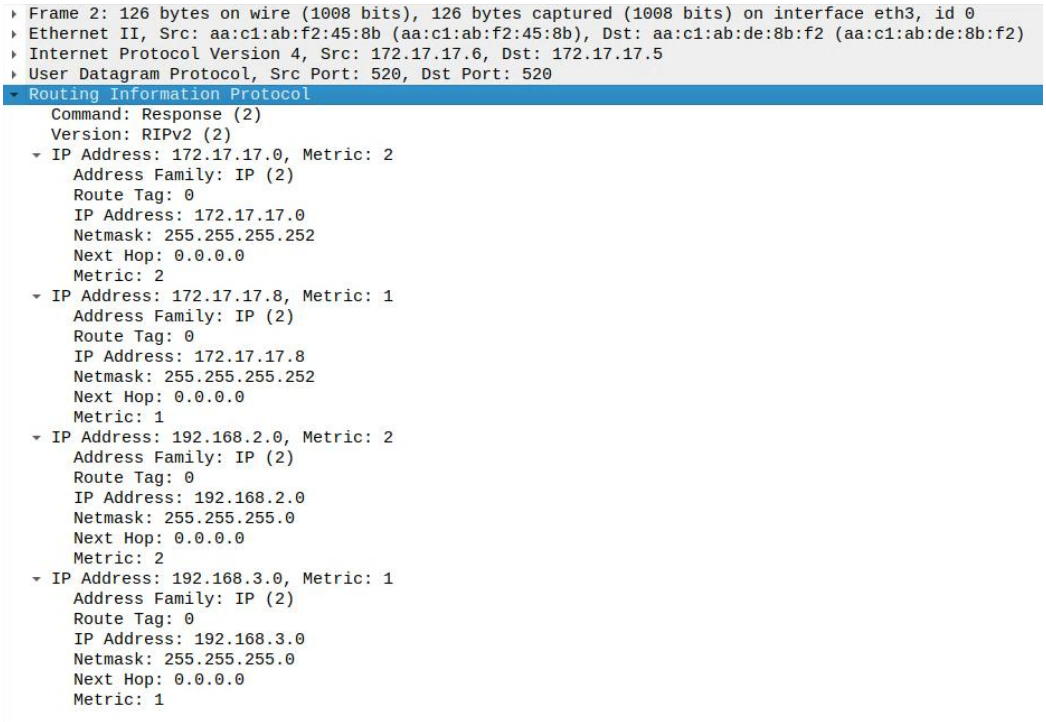
Στόχος είναι η επίτευξη της επικοινωνίας μεταξύ όλων των τελικών κόμβων. Εκτός από τον ορισμό προεπιλεγμένων πυλών μέσω του κατάλληλου δρομολογητή όπως προαναφέρθηκε, η επικοινωνία μεταξύ των δρομολογητών υλοποιείται μέσω του πρωτοκόλλου RIP. Σε κάθε δρομολογητή ενεργοποιείται το RIP και ανακοινώνονται όλα τα υποδίκτυα στα οποία συμμετέχει ο δρομολογητής με κάποια διεπαφή του. Για την ανακοίνωση των υποδικτύων χρησιμοποιείται η διεύθυνση υποδικτύου. Μέσω αυτής της διαδικασίας οι δρομολογητές ανταλλάσσουν διαδρομές και τελικά μαθαίνουν όλοι όλες τις διαθέσιμες και τις αποθηκεύουν στους πίνακες δρομολόγησης τους. Σε περίπτωση αποτυχίας κάποιας σύνδεσης οι δρομολογητές μαθαίνουν για αυτό μέσω ενημερώσεων RIP και αναπροσαρμόζουν τον πίνακα δρομολόγησης τους.

Ανοίγοντας την αρχική σελίδα του Edgeshark στη διεύθυνση http://0.0.0.0:5001 μπορεί όπως φαίνεται παρακάτω να εκκινήσει μια καταγραφή στις διεπαφές ενός δρομολογητή, έστω του router1 για να γίνει πιο κατανοητή η ανταλλαγή των πακέτων κατά τη λειτουργία του RIP.



Εικόνα 20: *Αρχική οθόνη Edgeshark*

Στη συνέχεια με ανακατεύθυνση στο Wireshark ξεκινάει η καταγραφή και παρατηρείται ότι ο δρομολογητής λαμβάνει τόσο από τον router2 όσο και από τον router3 μηνύματα RIP Response, με τα οποία γίνεται περιοδική ανταλλαγή πληροφοριών. Μέσω αυτών ο router1 μαθαίνει τις διαδρομές που του γνωστοποιούν οι άλλοι δρομολογητές και το κόστος τους (hop count) και ενημερώνει τον πίνακα δρομολόγησης τους. Τα μηνύματα αυτά περιλαμβάνουν μια επικεφαλίδα RIP με περιεχόμενο ξεχωριστές εγγραφές (Route Entries) για κάθε διαδρομή που ανακοινώνεται. Για παράδειγμα παρακάτω φαίνεται ένα μήνυμα RIP Response με πηγή τον router3 και προορισμό τον router1, το οποίο περιέχει δυο εγγραφές με κόστος (metric) 1, για τα υποδίκτυα του PC3 και του router2 που απέχουν ένα βήμα από τον router3 και δύο εγγραφές με κόστος (metric) 2, για τα υποδίκτυα του PC2 και του router1 που απέχουν ένα βήμα από τον router3. Σημειώνεται ότι ανακοινώνονται όλες οι εναλλακτικές διαδρομές και η επιλογή της καλύτερης που θα ενταχθεί στον πίνακα δρομολόγησης γίνεται από τον ίδιο τον δρομολογητή.

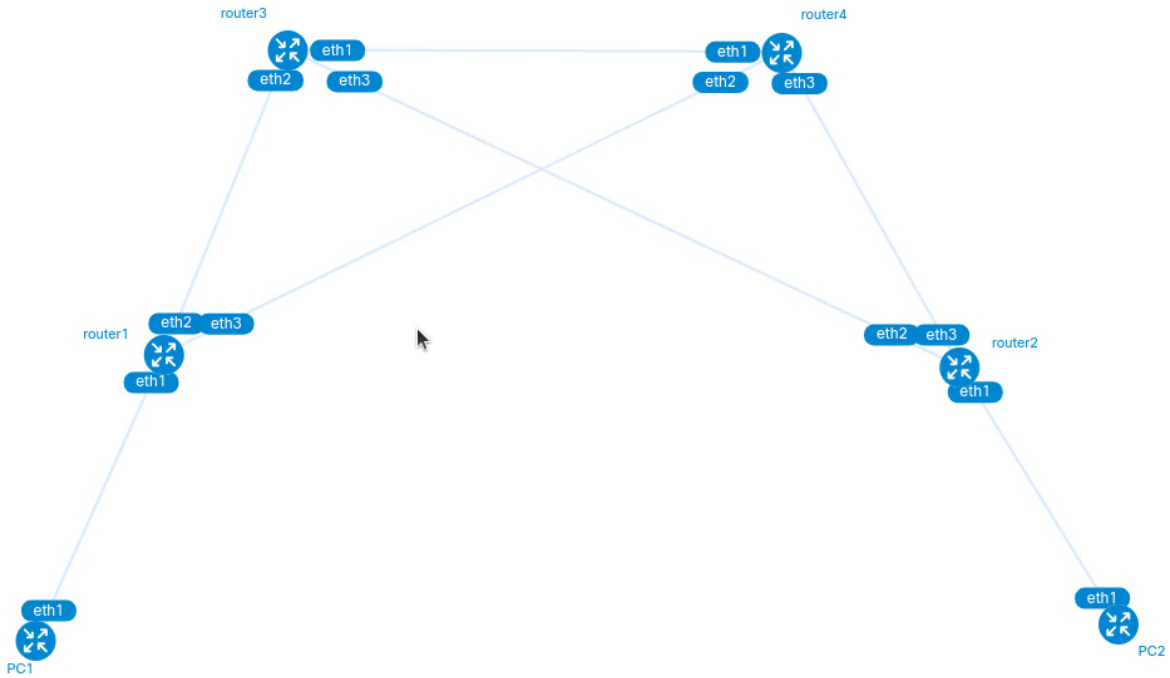


Εικόνα 21: *Μήνυμα RIP Response*

Οι ρυθμίσεις για τα PCs βρίσκονται σε ένα εκτελέσιμο αρχείο ενώ όλες οι παραμετροποιήσεις των δρομολογητών και του πρωτοκόλλου RIP γίνονται στα bind-mount αρχεία διαμόρφωσης. Επιπλέον γίνεται bind-mount με το αρχείο του FRR /etc/frr/daemons που περιλαμβάνει τους δαίμονες, ένα αρχείο daemons, το οποίο ενεργοποιεί μόνο τον δαίμονα ripd για το RIP και το zebra.

8.3 LAB2_RIP

Η τοπολογία lab2_rip συνιστά επίσης ένα σενάριο δυναμικής δρομολόγησης με χρήση του πρωτοκόλλου RIP. Η τοπολογία εδώ είναι πιο περίπλοκη, καθώς υπάρχουν δύο τελικοί κόμβοι PC1, PC2 που ανήκουν σε διαφορετικά υποδίκτυα (192.128.1.0/24 και 192.168.2.0/24) και 4 διαφορετικοί δρομολογητές, χωρισμένοι σε δύο επίπεδα. Η δομή της τοπολογίας είναι η ακόλουθη:



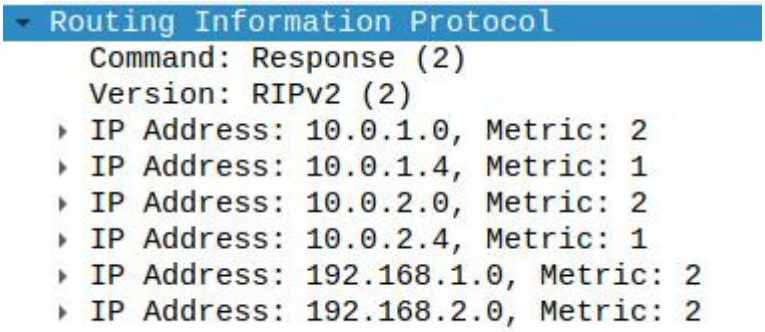
Εικόνα 22: *Σχηματική αναπαράσταση τοπολογίας LAB2_RIP*

Η παραπάνω μορφή επιτρέπει την δημιουργία πολλαπλών διαδρομών ικανών να υποστηρίξουν την επικοινωνία των δύο τελικών κόμβων PC1, PC2. Εφόσον οι δρομολογητές μοιράζονται πληροφορίες για τις διαθέσιμες διαδρομές μέσω RIP, η ύπαρξη τόσων εναλλακτικών διαδρομών αυξάνει την ανθεκτικότητα του δικτύου σε σφάλματα και βλάβες, διότι σε περίπτωση αδυναμίας μιας σύνδεσης η κίνηση ανακατευθύνεται άμεσα μέσω άλλου μονοπατιού.

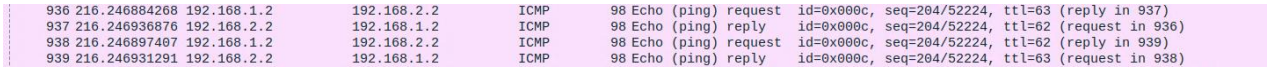
Η δυνατότητα αυτή του δικτύου για ανακατεύθυνση της κίνησης παρατηρείται πειραματικά στη συνέχεια μέσω του Wireshark. Αρχικά εκκινείται ένα ping από το PC1 προς το PC2 με την εντολή *sudo docker exec -it clab-lab2_rip-PC1 ping 192.168.2.2* και ταυτόχρονα αξιοποιώντας την επιλογή του Edgeshark για ταυτόχρονη καταγραφή σε πολλές διεπαφές πραγματοποιούνται δύο καταγραφές, μια στις διεπαφές eth1, eth2, eth3 του router3 και μια στις διεπαφές eth1, eth2, eth3 του router4. Στην αρχική κατάσταση παρατηρείται ICMP κίνηση στον router4 υποδεικνύοντας ότι για την επικοινωνία PC1-PC2 χρησιμοποιείται διαδρομή μέσω αυτού. Καταγράφονται επίσης RIP Response μηνύματα προερχόμενα από τον router4

τα οποία ανακοινώνουν κανονικά τα δίκτυα για τα οποία έχει διαδρομή ο δρομολογητής.

RIP Response:

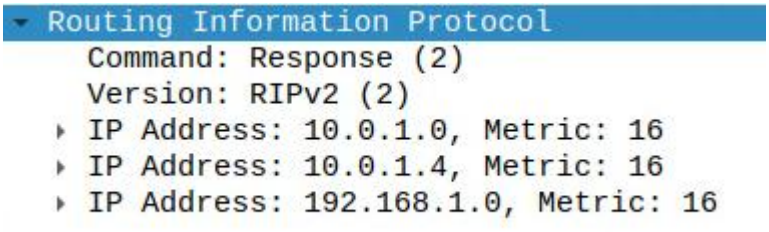


ICMP κίνηση:

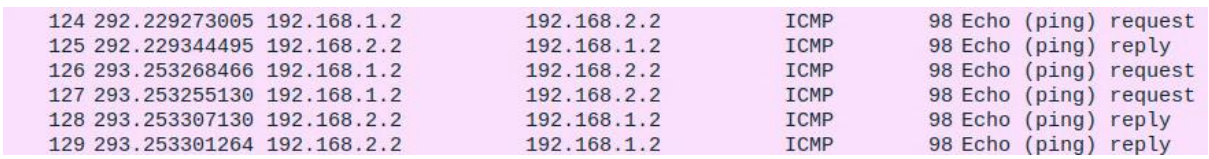


Στη συνέχεια πραγματοποιείται shutdown της διεπαφής eth2 του router4 ώστε να πραγματοποιηθεί απώλεια της διαδρομής. Παρατηρείται άμεση διακοπή καταγραφής κίνησης ICMP στον router4 και μια μικρή παύση στην επιτυχία του ping. Μετά από μερικά δευτερόλεπτα εντοπίζεται η εναλλακτική διαδρομή και γίνεται ανακατεύθυνση της κίνησης μέσω του router3, στον οποίο ξεκινάει να καταγράφεται ICMP κίνηση. Επιπλέον αμέσως μετά την πτώση της διεπαφής ο router4 στέλνει μήνυμα RIP Response στο οποίο αποδίδει στα μη προσβάσιμα πλέον μέσω αυτού δίκτυα κόστος (metric) 16 ώστε να αποσυρθούν οι αντίστοιχες διαδρομές.

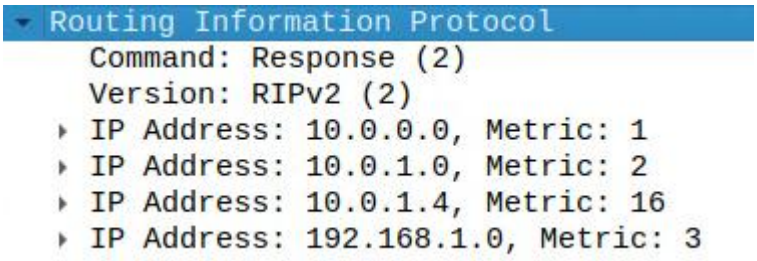
RIP Response του router4:



ICMP Κίνηση στον router3:



Λίγα δευτερόλεπτα αργότερα ο router4 μαθαίνει από τους γείτονες του εναλλακτικές διαδρομές για τα υποδίκτυα που προηγουμένως σηματοδότησε μη προσβάσιμα και εκπέμπει νέα μηνύματα RIP Response με επαναϋπολογισμένα κόστη.



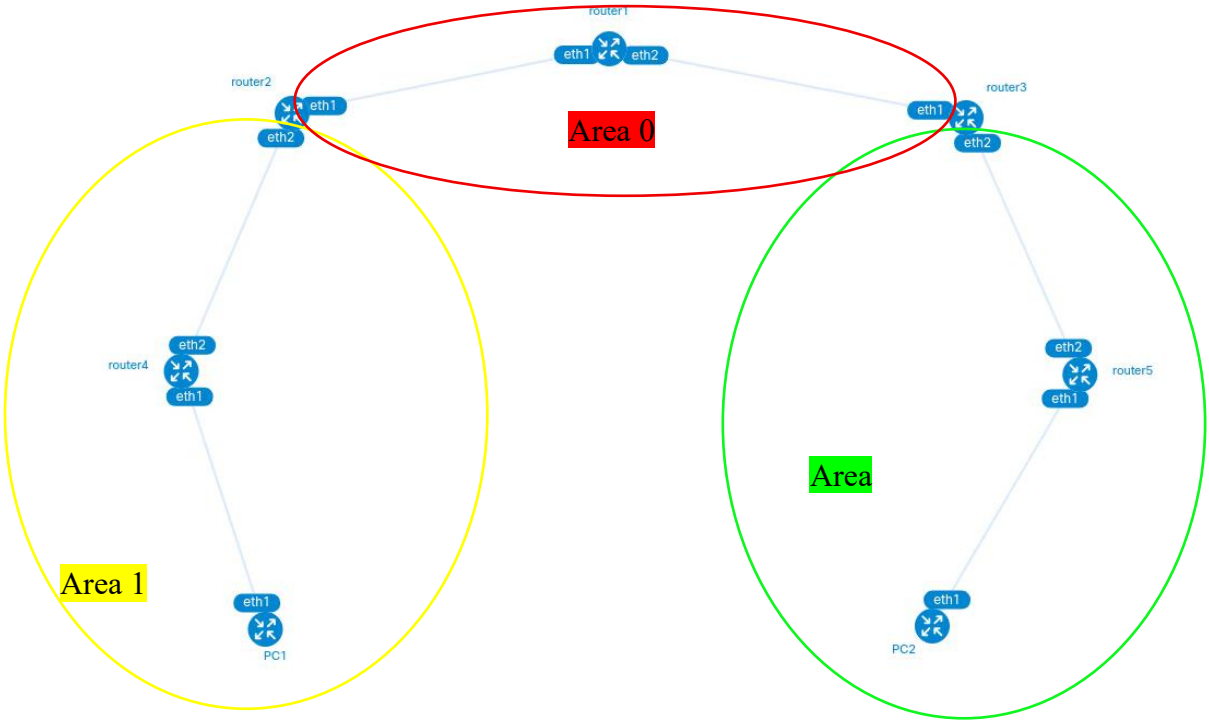
Όπως και προηγουμένως οι ρυθμίσεις για τα PCs γίνονται με τη χρήση ενός εκτελέσιμου αρχείου setup.sh και οι παραμετροποιήσεις των δρομολογητών με bind-mount αρχεία διαμορφώσεων. Κατά την ρύθμιση του πρωτοκόλλου RIP ακολουθείται μια εναλλακτική προσέγγιση και αντί να ανακοινώνεται το πρόθεμα του συγκεκριμένου δικτύου στο οποίο ανήκει μια διεπαφή δίνεται εντολή στον δαίμονα του RIP (ripd) να παρακολουθεί την κατάσταση της διεπαφής και να ανακοινώνει δυναμικά το υποδίκτυο στο οποίο αυτή ανήκει. Ακόμα και αν αυτό αλλάξει, ο δαίμονας συνεχίζει να παρακολουθεί τη διεπαφή, προσαρμόζεται στην μεταβολή και ανακοινώνει το νέο υποδίκτυο στο οποίο μεταφέρθηκε η διεπαφή. Για την επίτευξη αυτού του στόχου χρησιμοποιείται η εντολή network *eth[X]* , όπου *[X]* ο αριθμός της διεπαφής. Η πρακτική αυτή είναι ιδιαίτερα εύχρηστη και ευέλικτη, ειδικά για το περιβάλλον του Containerlab, στο οποίο τα containers μπορούν να επαναδημιουργούνται και να αλλάζουν εύκολα διευθύνσεις. Διευκολύνει δηλαδή τον πειραματισμό με τις διευθύνσεις και τις διασυνδέσεις των containers χωρίς να απαιτεί πολύπλοκη και χρονοβόρα παραμετροποίηση.

8.4 LAB_OSPF

Η παρούσα τοπολογία υλοποιεί ένα σενάριο δυναμικής δρομολόγησης βασισμένο στο πρωτόκολλο OSPF. Η αρχιτεκτονική που ακολουθείται είναι τυπική OSPF multi-area και στόχος είναι η επίτευξη της επικοινωνίας δύο τελικών κόμβων PC1, PC2 που συνδέονται σε διαφορετικές περιοχές, μέσω 5 ενδιάμεσων δρομολογητών. Οι δρομολογητές είναι διαχωρισμένοι σε 3 περιοχές. Ο router1 ανήκει στην περιοχή κορμού (area 0) και οι router4, router5 στις περιοχές 1 και 2 αντίστοιχα ως Internal Routers, στις οποίες συμμετέχουν αντίστοιχα τα υποδίκτυα 192.168.1.0/24 και 192.168.2.0/24 των PC1 και PC2. Ο router2 λειτουργεί ως ABR (Area Border Router) για την περιοχή 1 συνδέοντας την με την περιοχή κορμού (area 0) και ο router3 λειτουργεί ως ABR για την περιοχή 2 συνδέοντας την επίσης με την περιοχή κορμού.

Σε κάθε PC ορίζεται ως προκαθορισμένη πύλη η άμεσα συνδεδεμένη σε αυτόν διεπαφή του γειτονικού δρομολογητή. Οι δρομολογητές λαμβάνουν διευθύνσεις σε

διαφορετικό υποδίκτυο για κάθε point-to-point σύνδεση μεταξύ τους. Επιπλέον σε κάθε δρομολογητή ορίζεται και η loopback διεπαφή με σταθερή και μοναδική IP διεύθυνση με μάσκα υποδικτύου (subnet mask) /32 (255.255.255.255). Έτσι εξασφαλίζεται η ύπαρξη σταθερού ID ανεξάρτητου από την κατάσταση των διεπαφών. Ο κάθε δρομολογητής διαφημίζει στους υπόλοιπους της περιοχής του, τους γείτονες του δηλαδή τα υποδίκτυα στα οποία ανήκει και έτσι κλιμακωτά όλοι έχουν γνώση των διαθέσιμων διαδρομών για κάθε περιοχή. Για παράδειγμα ο router4 ενημερώνεται για τις OSPF διαδρομές των area 0 και area 2 από τον router2, με τον οποίο έχει εγκαταστήσει σχέση γειτνίασης. Ακολουθεί σχηματική αναπαράσταση της τοπολογίας όπως προκύπτει από την εντολή *graph* , με επισημασμένες τις OSPF περιοχές.



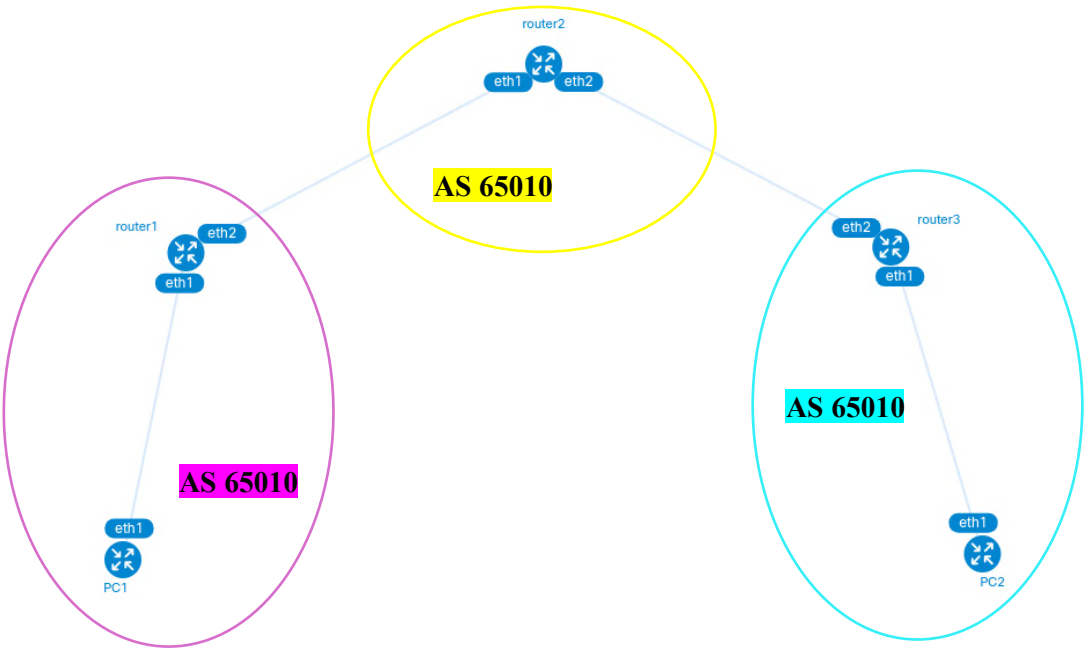
Εικόνα 23: *Σχηματική αναπαράσταση τοπολογίας LAB_OSPF*

Αμέσως μετά την εκκίνηση της τοπολογίας είτε για πρώτη φορά (χρήση αρχείου labdeploy.sh) είτε κατά την επαναδημιουργία της μετά από μεταβολές (χρήση αρχείου labreconfigure.sh) παρατηρήθηκε ότι απαιτούνται από 35 έως και 55 δευτερόλεπτα μέχρι η επικοινωνία των δύο PC να είναι εφικτή και το ping να επιτυγχάνει. Η καθυστέρηση αυτή οφείλεται εν μέρει στην διαδικασία σύγκλισης του OSPF, μέχρι δηλαδή να εγκατασταθούν οι σχέσεις γειτνίασης μεταξύ των δρομολογητών και να ανταλλαχθούν οι διαφημίσεις ώστε να μάθουν οι δρομολογητές τις διαθέσιμες διαδρομές και να μπορούν να στείλουν κατάλληλα τα πακέτα. Πέρα από αυτό, καθυστέρηση προστίθεται και από τον χρόνο εκκίνησης της υπηρεσίας FRR εντός των δρομολογητών. Παρόλο που το Containerlab ολοκληρώνει την ανάπτυξη της τοπολογίας και τα containers φαίνονται ενεργά, δεν συμβαίνει το ίδιο

και με το FRR και τον δαίμονα ospfd, τα οποία χρειάζονται κάποια παραπάνω δευτερόλεπτα. Το Containerlab εκκινεί όλους τους κόμβους μαζικά χωρίς να συμβάλλει με κάποιο τρόπο στον συγχρονισμό του OSPF, με πιθανό αποτέλεσμα να επιχειρείται η επικοινωνία των δύο PC πριν την συνολική δημιουργία και σύγκλιση του δικτύου. Η καθυστέρηση αυτή υφίσταται μόνο στην αρχή αμέσως μετά την ανάπτυξη (deployment) της τοπολογίας και από τη στιγμή που επιτευχθεί η επικοινωνία δεν ξαναπροκύπτει καθώς το δίκτυο έχει πλέον σταθεροποιηθεί. Συνεπώς τέτοιες μικρές καθυστερήσεις σε παρόμοιες τοπολογίες θα πρέπει να θεωρούνται φυσιολογικές στο περιβάλλον του Containerlab και αναμενόμενες χωρίς να εγείρουν προβληματισμούς για την ορθότητα της λειτουργίας του δικτύου.

8.5 LAB_BGP

Στο παράδειγμα αυτό εξετάζεται η λειτουργία του πρωτοκόλλου BGP. Στην τοπολογία που προσομοιώνεται στόχος είναι η επίτευξη της επικοινωνίας μεταξύ δύο διαφορετικών τοπικών υποδικτύων (LANs), τα οποία ανήκουν σε διαφορετικά αυτόνομα συστήματα και συνδέονται διαμέσου ενός τρίτου ενδιάμεσου αυτόνομου συστήματος (AS). Όπως φαίνεται και στο σχήμα που ακολουθεί χρησιμοποιούνται 3 δρομολογητές και 2 τελικοί κόμβοι. Ο router1 ανήκει στο AS 65010 και συνδέεται με το υποδίκτυο του PC1 (LAN1 – 192.168.1.0/24) , ενώ ο router3 ανήκει στο AS 65030 και συνδέεται με το υποδίκτυο του PC2 (LAN2 – 192.168.2.0/24). Ο ενδιάμεσος δρομολογητής router2 βρίσκεται στο AS 65020, το οποίο αποτελεί το transit αυτόνομο σύστημα της τοπολογίας και είναι υπεύθυνο για την διασύνδεση των άλλων δύο και διασφαλίζει ότι οι BGP ενημερώσεις που λαμβάνει από τον router1 για το AS 65010 μεταφέρονται στον router3 και αντίστροφα.



Εικόνα 24: *Σχηματική αναπαράσταση τοπολογίας LAB_BGP*

Οι σύνοδοι (BGP sessions) που εγκαθίστανται μεταξύ router1 – router2 και router3 – router2 αποτελούν χαρακτηριστικό παράδειγμα eBGP (external BGP) και μέσω αυτών διαφημίζονται τα LAN των PC1, PC2 από τους δρομολογητές 1 και 3 αντίστοιχα, ώστε αυτοί να μάθουν τις απαραίτητες για την επικοινωνία των δύο κόμβων εγγραφές. Η αρχιτεκτονική αυτή αντικατοπτρίζει την πραγματική λειτουργία του BGP στο Διαδίκτυο, όπου διαφορετικοί πάροχοι δικτύου ανταλλάσσουν πληροφορίες δρομολόγησης μέσω eBGP συνεδριών.

Με την είσοδο στο vtysh του FRR των δρομολογητών και τη χρήση της εντολής *show ip bgp* επιβεβαιώνεται η ορθή εκμάθηση των διαδρομών από τους κόμβους. Ενδεικτικά παρακάτω φαίνονται οι διαδρομές που έχει καταχωρημένες ο router1. Στο πεδίο Path εμφανίζεται η αλληλουχία αυτόνομων συστημάτων που ακολουθεί ένα πακέτο στην διαδρομή. Για παράδειγμα για το υποδίκτυο 192.168.2.0/24 ο router1 γνωρίζει ότι ένα πακέτο θα διέλθει διαδοχικά από τα αυτόνομα συστήματα 65020, 65030.

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0/24   0.0.0.0                  0         32768 i
*> 192.168.2.0/24   10.1.1.2                               0 65020 65030 i

Displayed 2 routes and 2 total paths
```

Στη συνέχεια χρησιμοποιείται η εντολή *docker container stats* για την παρατήρηση σε πραγματικό χρόνο των πόρων που καταναλώνονται από τα containers της τοπολογίας. Η ακόλουθη εικόνα αποτελεί ένα στιγμιότυπο των αποτελεσμάτων της παραπάνω εντολής κατά τη διάρκεια της εκτέλεσης ενός ping από το PC1 προς το PC2. Σημειώνεται ότι καθώς τα στατιστικά της *docker container stats* ανανεώνονται διαρκώς σε πραγματικό χρόνο, οι παρακάτω αριθμοί αποτελούν ενδεικτικά αποτελέσματα και αποτυπώνουν μια στιγμή της κατάστασης. Θεωρούνται ωστόσο αρκετά αντιπροσωπευτικά διότι λαμβάνονται σε μια στιγμή κατά την οποία το σύστημα έχει αναπτυχθεί πλήρως και έχει σταθεροποιηθεί χωρίς να παρουσιάζει σημαντικές αποκλίσεις.

```
cbb2d7c01ba0   clab-lab_bgp-router2   0.03%   29.23MiB / 15.61GiB   0.18%   8.87kB / 1.33kB   4.86MB / 90.1kB   22
e061e5f4b869   clab-lab_bgp-PC1       0.02%   3.309MiB / 15.61GiB   0.02%   8.56kB / 1.4kB    2.23MB / 16.4kB   3
6e4b18a2739f   clab-lab_bgp-router3   0.05%   27.86MiB / 15.61GiB   0.17%   8.22kB / 1.4kB    205kB / 90.1kB    22
b749201e1d16   clab-lab_bgp-router1   0.11%   28.03MiB / 15.61GiB   0.18%   7.98kB / 1.4kB    516kB / 90.1kB    22
47544c1392e3   clab-lab_bgp-PC2       0.00%   2.09MiB / 15.61GiB    0.01%   7.7kB / 1.4kB     0B / 16.4kB       2
```

Παρατηρείται ότι η χρήση της CPU είναι ιδιαίτερα χαμηλή και δεν ξεπερνάει σε ποσοστό κατανάλωσης το 0.2% ακόμα και από τους δρομολογητές, οι οποίοι εκτελούν το πρωτόκολλο BGP. Όπως αναμένεται οι δρομολογητές καταναλώνουν περισσότερη μνήμη RAM λόγω των δαιμόνων του FRR που υποστηρίζουν. Ωστόσο η χρήση μνήμης που γίνεται τόσο από τους τερματικούς κόμβους όσο και από τους δρομολογητές είναι αμελητέα σε σχέση με το όριο των 15.61 GiB του host συστήματος. Λόγω της ICMP κίνησης κατά τη διάρκεια του ping σημειώνεται εισερχόμενη αλλά και εξερχόμενη κίνηση που διέρχεται μέσω του δικτύου BGP σε όλους τους κόμβους της τοπολογίας. Σχετικά με το Block I/O δηλαδή τον όγκο

ανάγνωσης/εγγραφής από και προς τον αποθηκευτικό δίσκο παρατηρούνται αρκετά χαμηλές τιμές, οι οποίες οφείλονται στην μικρή αλληλεπίδραση των container με το δίσκο. Αρχικά η λειτουργία των containers, όπως δρομολόγηση, ρυθμίσεις BGP γίνεται κατά κύριο λόγο στη μνήμη χωρίς να απαιτεί συνεχείς αναγνώσεις και εγγραφές από τον δίσκο. Επιπλέον τα images των container είναι ήδη φορτωμένα οι παραμετροποιήσεις μέσω των bind-mount αρχείων εφαρμόζονται κατά την εκκίνηση και έπειτα διατηρούνται στην μνήμη RAM. Τέλος οι δρομολογητές εμφανίζουν 22 διεργασίες ο καθένας, γεγονός που οφείλεται στη λειτουργία του FRR και των δαιμόνων του, ενώ τα PCs μόλις 2–3 διεργασίες, καθώς λειτουργούν με απλούστερο τρόπο μόνο ως τερματικοί κόμβοι.

Οι παραπάνω μετρήσεις αναδεικνύουν σημαντικά πλεονεκτήματα των περιβαλλόντων με containers για δικτυακή προσομοίωση, όπως τον ελαφρύ και αποδοτικό τους χαρακτήρα. Στην προκειμένη περίπτωση το Containerlab αξιοποιεί τις μικρές απαιτήσεις σε πόρους των containers και τις διατηρεί όπως φαίνεται παραπάνω σε χαμηλά επίπεδα προσφέροντας παράλληλα την δυνατότητα προσομοίωσης ενός ρεαλιστικού σεναρίου το οποίο χρησιμοποιεί το BGP και μιμείται την πραγματική λειτουργία του στο διαδίκτυο.

8.6 LAB_OVS

Στην παρούσα τοπολογία αξιοποιείται ένας OVS (Open vSwitch) μεταγωγέας (OVS bridge) για την διασύνδεση δύο τερματικών κόμβων. Πρόκειται για μια απλή δικτυακή υλοποίηση, η οποία ωστόσο ακολουθεί μια διαφορετική προσέγγιση σε σχέση με τα προηγούμενα παραδείγματα και επιτυγχάνει την επικοινωνία των δύο τελικών κόμβων χωρίς τη χρήση κάποιου πρωτοκόλλου δρομολόγησης αλλά μέσω μιας εικονικής γέφυρας, η οποία λειτουργεί στο στρώμα ζεύξης δεδομένων.

Για την υλοποίηση της τοπολογίας χρησιμοποιείται αρχικά ένα αρχείο YAML με την περιγραφή των κόμβων της και των μεταξύ τους συνδέσεων. Η εικονική γέφυρα ορίζεται ως ένας κόμβος του οποίου το πεδίο kind έχει την τιμή ovs-bridge και στη συνέχεια για τις διεπαφές της χρησιμοποιείται ο συμβολισμός ovsp1, ovsp2. Επιπλέον χρησιμοποιείται ένα εκτελέσιμο αρχείο setup.sh για την εκχώρηση IP διευθύνσεων στα PCs και την ενεργοποίηση των διεπαφών τους.

Πριν την ανάπτυξη της τοπολογίας θα πρέπει να δοθεί ιδιαίτερη προσοχή στο γεγονός ότι το Containerlab δεν μπορεί να δημιουργήσει από μόνο του τη γέφυρα. Χρησιμοποιεί τον τύπο ovs-bridge στο YAML αρχείο και ορίζει συνδέσεις για τις διεπαφές αυτής της γέφυρας, μόνο εφόσον η γέφυρα υπάρχει ήδη στο σύστημα. Συνεπώς είναι απαραίτητο να προηγηθεί του deployment της τοπολογίας η δημιουργία της γέφυρας μέσω του Open vSwitch με χρήση της εντολής *ovs-vsctl add-br myovs ,* όπου myovs το όνομα της γέφυρας το οποίο εντοπίζεται και εντός του

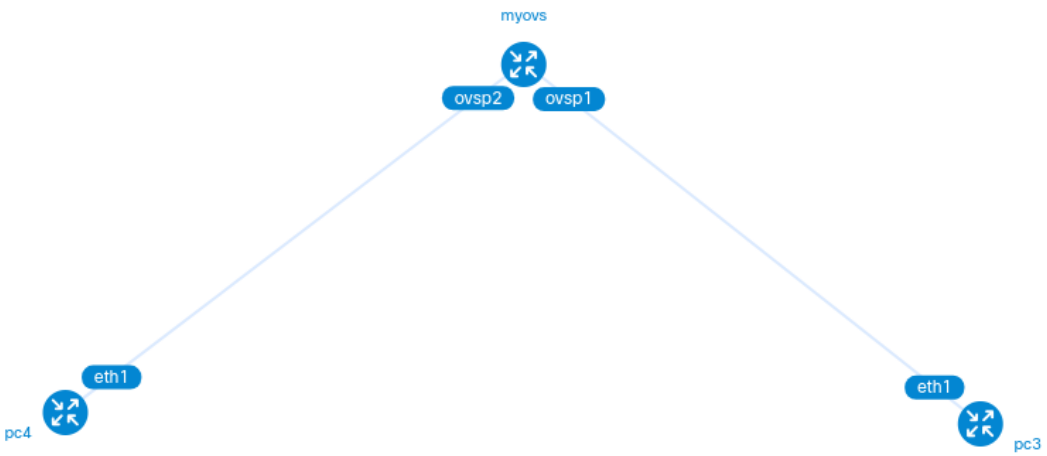
YAML αρχείου. Η επιβεβαίωση της ορθής δημιουργίας της μπορεί να γίνει με την εντολή *ovs-vsctl show*:

```
    Bridge myovs
        Port myovs
            Interface myovs
                type: internal
    ovs_version: "2.13.8"
```

Στη συνέχεια γίνεται η ανάπτυξη της τοπολογίας με *deploy -t lab_ovs.yml* και η εκτέλεση του setup.sh για την απόδοση των διευθύνσεων. Με εκτέλεση και πάλι της εντολής *ovs-vsctl show* θα πρέπει εάν η γέφυρα έχει αποκτήσει σωστά τις απαιτούμενες διεπαφές αυτές να εμφανίζονται ως εξής:

```
f5787afc-05f0-4f6d-9cad-6b67e135f895
    Bridge myovs
        Port myovs
            Interface myovs
                type: internal
        Port ovsp2
            Interface ovsp2
        Port ovsp1
            Interface ovsp1
    ovs_version: "2.13.8"
```

Με *clab graph -t lab_ovs.yml* προσφέρεται μια οπτικοποίηση της τοπολογίας και επιβεβαιώνεται και σχηματικά η ορθή δημιουργία της τοπολογίας και η παρεμβολή της γέφυρας μεταξύ των δύο PCs.:

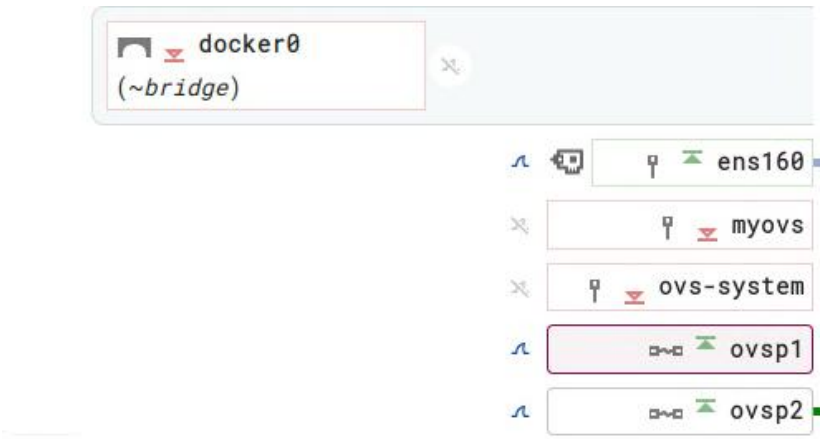


Εικόνα 25: *Σχηματική αναπαράσταση τοπολογίας LAB_OVS*

Τέλος επιβεβαιώνεται η επικοινωνία των δύο τελικών κόμβων:

```
root@LINUX:/etc/containerlab/lab_ovs# sudo docker exec clab-lab_ovs-pc4 ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.102 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.091 ms
```

Το ακόλουθο στιγμιότυπο από το Edgeshark απεικονίζει ξεκάθαρα ότι η γέφυρα myovs δημιουργείται σε επίπεδο λειτουργικού συστήματος, ενώ η διασύνδεση με τα containers γίνεται με εικονικά ζεύγη Ethernet (veth pairs), τα οποία συνδέουν το namespace του container με το OVS. Η γέφυρα OVS τοποθετείται σαν ένας ενδιάμεσος κόμβος και στη συνέχεια οι διεπαφές της (ovsp1, ovsp2) χρησιμοποιούνται από το Containerlab όπως ένα οποιοδήποτε άλλο δίκτυο. Στην περίπτωση αυτή το Containerlab λειτουργεί κυρίως ως ενορχηστρωτικό εργαλείο, συνδυάζοντας υπάρχουσες τεχνολογίες και προσφέροντας ευελιξία.



Εικόνα 26: *Απεικόνιση διεπαφών OVS στο Edgeshark*

8.7 LINUX_BRIDGE

Μια εναλλακτική υλοποίηση της προηγούμενης τοπολογίας μπορεί να γίνει με τη χρήση μιας Linux bridge στη θέση της OVS γέφυρας. Και οι δυο προσεγγίσεις έχουν ως στόχο τη διασύνδεση δύο τελικών κόμβων στο στρώμα ζεύξης δεδομένων διαμέσου μιας γέφυρας.

Η μεθοδολογία που ακολουθείται είναι σε γενικές γραμμές η ίδια. Η γέφυρα θα πρέπει και πάλι να δημιουργηθεί εξωτερικά διότι το Containerlab δεν δύναται να την κατασκευάσει εξ' αρχής παρά μόνο να την χρησιμοποιήσει. Για την κατασκευή της γέφυρας χρησιμοποιείται η εντολή *sudo brctl addbr br01* και έπειτα απαιτείται ενεργοποίηση της με την εντολή *sudo ip link set br01 up*. Το YAML αρχείο περιγράφει την τοπολογία και χρησιμοποιεί την γέφυρα br01 ως ένα κόμβο με το kind ορισμένο ως bridge και τις διεπαφές του στη μορφή eth1, eth2.

Λεπτομέρειες για τη γέφυρα ώστε να ελεγχθεί η λειτουργία της δίνονται με *sudo brctl show br01*.

```
bridge name     bridge id               STP enabled     interfaces
br01            8000.7a93afcf23a9       no              eth1
                                                        eth2
```

Η επικοινωνία μεταξύ των δύο κόμβων μέσω της br01 πραγματοποιείται αποκλειστικά στο στρώμα ζεύξης δεδομένων χωρίς να εμπλέκονται διευθύνσεις IP για δρομολόγηση. Η γέφυρα br01 δηλαδή, δεν εξετάζει τις IP διευθύνσεις αλλά προωθεί το πλαίσιο με βάση τον πίνακα προώθησης MAC διευθύνσεων που διαθέτει. Κατά τη διάρκεια του πρώτου ping ο προορισμός είναι άγνωστος και τότε η γέφυρα κάνει flooding τα πλαίσια που λαμβάνει σε όλες τις διεπαφές της μέχρι να λάβει απάντηση και να καταγράψει στον πίνακα προώθησης της τη συσχέτιση θύρας – MAC διεύθυνσης. Στη συνέχεια η προώθηση γίνεται στοχευμένα. Πριν από οποιαδήποτε ανταλλαγή πακέτων, ο πίνακας παραμένει κενός, καθώς η γέφυρα δεν έχει καμία πληροφορία για τους συνδεδεμένους κόμβους, ενώ οι εγγραφές διαγράφονται μετά από ένα διάστημα αδράνειας.

Ο πίνακας προώθησης της br01 πριν από την εκτέλεση κάποιου ping ή μετά τη λήξη του ageing timer (300 sec) όπως προκύπτει από την εντολή *sudo brctl showmacs br01,* περιέχοντας εγγραφές μόνο για τις εικονικές διεπαφές που ανήκουν στην γέφυρα:

```
port no mac addr                is local?       ageing timer
  2     aa:c1:ab:1e:89:cb       yes                0.00
  2     aa:c1:ab:1e:89:cb       yes                0.00
  1     aa:c1:ab:a9:16:47       yes                0.00
  1     aa:c1:ab:a9:16:47       yes                0.00
```

Ο πίνακας προώθησης της br01 μετά την εκτέλεση κάποιου ping και πριν τη λήξη του ageing timer (300 sec) περιλαμβάνει όπως φαίνεται επιπλέον τις MAC διευθύνσεις των pc1, pc2 και την θύρα στην οποία αντιστοιχούν, ενώ διευκρινίζεται ότι δεν πρόκειται για τοπικές διευθύνσεις (is local? → no) :

```
port no mac addr                is local?       ageing timer
  2     aa:c1:ab:1e:89:cb       yes                0.00
  2     aa:c1:ab:1e:89:cb       yes                0.00
  1     aa:c1:ab:6d:6d:ab       no                 2.43
  2     aa:c1:ab:85:ea:23       no                 2.43
  1     aa:c1:ab:a9:16:47       yes                0.00
  1     aa:c1:ab:a9:16:47       yes                0.00
```

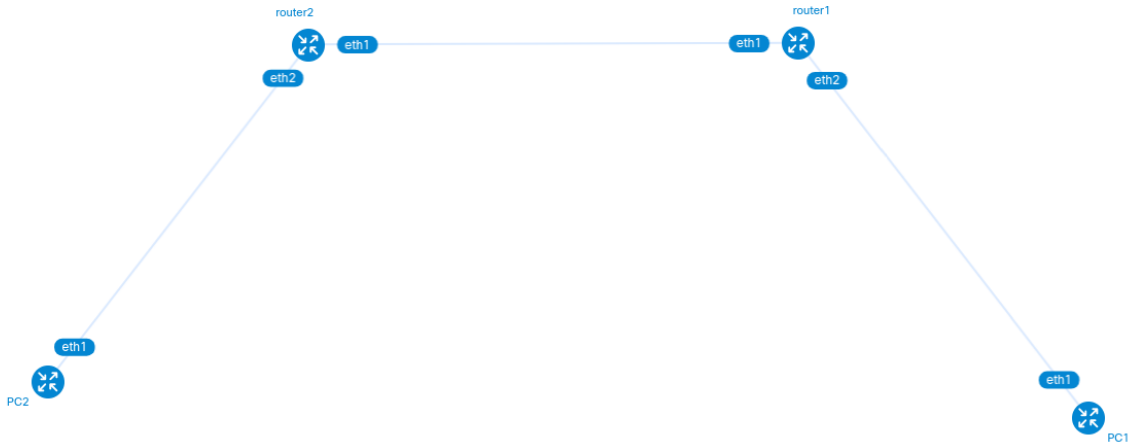
Εκτελώντας traceroute από το pc2 προς το pc1 για την καταγραφή των ενδιάμεσων κόμβων από τους οποίους διέρχονται τα πακέτα κατά την επικοινωνία τους παρατηρείται όπως αναμενόταν ένα μόνο βήμα με την διεύθυνση του pc1. Το γεγονός αυτό οφείλεται στην λειτουργία της γέφυρας στο στρώμα ζεύξης δεδομένων και

συνεπώς την απουσία της ως πρόσθετο βήμα στην έξοδο του traceroute που λειτουργεί στο στρώμα δικτύου.

```
traceroute to 192.168.1.1 (192.168.1.1), 30 hops max, 46 byte packets
 1  192.168.1.1 (192.168.1.1)  0.008 ms  0.017 ms  0.007 ms
```

8.8 LAB_VXLAN_PTP

Η συγκεκριμένη τοπολογία υλοποιεί ένα σενάριο διασύνδεσης δύο τελικών κόμβων με χρήση της τεχνικής VXLAN. Τα δύο PCs βρίσκονται στα άκρα της τοπολογίας, ανήκουν στο ίδιο υποδίκτυο και συνδέονται μέσω δύο δρομολογητών, μεταξύ των οποίων δημιουργείται ένα VXLAN Tunnel. Η δομή της τοπολογίας φαίνεται στο παρακάτω σχήμα.



Εικόνα 27: *Σχηματική αναπαράσταση τοπολογίας LAB_VXLAN_PTP*

Οι άμεσα συνδεόμενες διεπαφές των δρομολογητών λαμβάνουν IP διευθύνσεις εντός του ίδιου υποδικτύου, ενώ ο κάθε ένας εκ των δύο αποτελεί την προεπιλεγμένη πύλη του άλλου.

Η VXLAN υλοποίηση γίνεται εντός του εκτελέσιμου αρχείου setup.sh. Οι δύο δρομολογητές λειτουργούν ως VTEPs και εκτελούν τις λειτουργίες encapsulation και decapsulation των VXLAN πακέτων από και προς τα δύο PCs. Εφαρμόζονται ακριβώς τα ίδια βήματα για τους δύο routers με κατάλληλη προσαρμογή των IP. Αρχικά δημιουργείται μια εικονική διεπαφή vxlan100 τύπου VXLAN στην οποία αποδίδεται ID (VNI- VXLAN Network Identifier) 100 και για την οποία ορίζεται ως θύρα προορισμού η 4789 που είναι η default θύρα για το VXLAN. Επιπλέον ορίζεται ως IP πηγής (local IP) των VXLAN πακέτων την IP της eth1 του εκάστοτε router και ως IP προορισμού αντίστοιχα την IP της eth1 του απέναντι router για επικοινωνία. Στη συνέχεια με χρήση της εντολής brctl προστίθεται σε κάθε router από μια linux bridge με όνομα br100, η οποία λειτουργεί σαν ένας μεταγωγέας στρώματος ζεύξης

(Layer 2) και στην κάθε γέφυρα προστίθεται η vxlan100 διεπαφή. Απενεργοποιείται το πρωτόκολλο STP καθώς για την point-to-point σύνδεση δεν είναι απαραίτητο. Ακολουθεί ενεργοποίηση της γέφυρας και της vxlan διεπαφής. Τέλος είναι αναγκαία η προσθήκη στην κάθε γέφυρα της αντίστοιχης διεπαφής eth2 του κάθε router ώστε να επιτραπεί στην γέφυρα br100 να διαχειρίζεται τα πακέτα που στέλνει ο PC και να καταστεί εφικτή η επικοινωνία των δύο PCs διαμέσου του VXLAN tunnel.

Με την εντολή *sudo docker exec clab-lab_vxlan_PtP-router1 bridge fdb show dev vxlan100 | grep dst* εμφανίζεται ο πίνακας προώθησης της γέφυρας στον router1 και στον router2. Όπως παρατηρείται παρακάτω υπάρχουν τρεις εγγραφές στον κάθε πίνακα προώθησης. Και στους δύο η πρώτη εγγραφή δείχνει ότι οι άγνωστες MAC διευθύνσεις θα προωθηθούν στην eth1 διεπαφή του απέναντι router, δηλαδή στο remote VTEP μέσω VXLAN. Η δεύτερη εγγραφή στον router1 αφορά, όπως επιβεβαιώνεται μέσω της *sudo docker exec -it clab- lab_vxlan_PtP-PC2 ip -br link ,* την διεπαφή eth1 του PC2 και δείχνει ότι η MAC του PC2 έγινε γνωστή μέσω του router2 στον οποίο ανήκει η IP 10.1.1.2 και στον οποίο θα προωθηθεί μέσω VXLAN οποιοδήποτε πακέτο με προορισμό αυτή τη διεύθυνση MAC. Η τρίτη εγγραφή αναφέρεται στην linux bridge του router2 και αντιστοιχίζει την MAC της γέφυρας br100 με το VTEP στο οποίο ανήκει η διεύθυνση 10.1.1.2, δηλαδή με τον router2 αφού αυτή είναι η IP της eth1 διεπαφής του, με αποτέλεσμα πλέον ο router1 να γνωρίζει ότι η γέφυρα βρίσκεται πίσω από αυτόν και να προωθηθεί τα πακέτα μέσω του τούνελ VXLAN προς το router2. Αντίστοιχες εγγραφές εντοπίζονται και στον router2 για το PC1 και τον router1.

```
akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-router1 bridge fdb show dev vxlan100 | grep dst
00:00:00:00:00:00 dst 10.1.1.2 self permanent
aa:c1:ab:77:a4:cb dst 10.1.1.2 self
32:11:ec:d8:7b:bd dst 10.1.1.2 self
akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-router2 bridge fdb show dev vxlan100 | grep dst
00:00:00:00:00:00 dst 10.1.1.1 self permanent
62:47:4a:a4:31:65 dst 10.1.1.1 self
aa:c1:ab:be:ab:f8 dst 10.1.1.1 self
akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-PC1 ip -br link
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
eth0@if469       UP             e2:f2:3c:1b:9e:d0 <BROADCAST,MULTICAST,UP,LOWER_UP>
eth1@if475       UP             aa:c1:ab:be:ab:f8 <BROADCAST,MULTICAST,UP,LOWER_UP>
akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-PC2 ip -br link
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
eth0@if468       UP             3e:32:1c:a2:3f:7a <BROADCAST,MULTICAST,UP,LOWER_UP>
eth1@if471       UP             aa:c1:ab:77:a4:cb <BROADCAST,MULTICAST,UP,LOWER_UP>
akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-router1 ifconfig br100
br100     Link encap:Ethernet  HWaddr 62:47:4A:A4:31:65
          inet6 addr: fe80::6047:4aff:fea4:3165/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:33 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2120 (2.0 KiB)  TX bytes:1070 (1.0 KiB)

akoutsoni@clab:~/topologies/containerlab/lab_vxlan_PtP$ sudo docker exec clab-lab_vxlan_PtP-router2 ifconfig br100
br100     Link encap:Ethernet  HWaddr 32:11:EC:D8:7B:BD
          inet6 addr: fe80::3011:ecff:fed8:7bbd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:29 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1744 (1.7 KiB)  TX bytes:946 (946.0 B)
```

Λεπτομέρειες για την VXLAN διεπαφή στους δρομολογητές παρέχονται με την εντολή *sudo docker exec clab-lab_vxlan_PtP-router1 vtysh -c "show int vxlan100"*

Ενδεικτικά για τον router1:

```
Interface vxlan100 is up, line protocol is up
  Link ups:       1    last: 2025/09/02 09:08:21.97
  Link downs:     2    last: 2025/09/02 09:08:21.71
  vrf: default
  index 3 metric 0 mtu 1500 speed 0 txqlen 1000
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  Type: Ethernet
  HWaddr: 62:47:4a:a4:31:65
  inet6 fe80::6047:4aff:fea4:3165/64
  Interface Type Vxlan
  Interface Slave Type Bridge
 VTEP IP: 10.1.1.1  VxLAN Id 100 Access VLAN Id 1
  Mcast Group 10.1.1.2
  Master interface: br100
  protodown: off
```

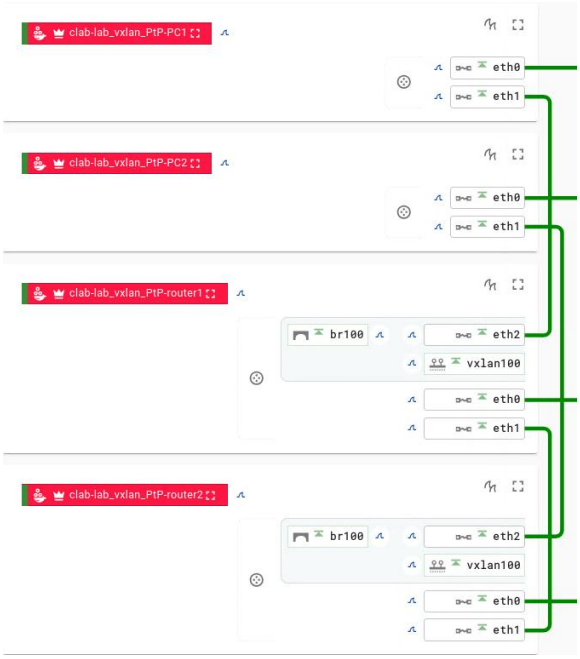
Κάνοντας μια καταγραφή στο Wireshark στην eth1 του router1 μπορούν να μελετηθούν αναλυτικά οι επικεφαλίδες του πακέτου που μεταδίδεται μέσω του VXLAN tunnel. Κάθε πακέτο λαμβάνει αρχικά μια εξωτερική Ethernet επικεφαλίδα με τις MAC διευθύνσεις των VTEP πηγής και προορισμού. Έπειτα μια IP επικεφαλίδα με τις IP διευθύνσεις των VTEP πηγής και προορισμού. Ακολουθεί η UDP επικεφαλίδα με τη θύρα πηγής που προσδιορίζεται δυναμικά και τη θύρα προορισμού που είναι η default για το VXLAN 4789 και ορίστηκε προηγουμένως. Προστίθεται το VXLAN header που περιλαμβάνει το VNI και τέλος ενθυλακώνεται αναλλοίωτο στο UDP πακέτο το αρχικό ICMP πακέτο που περιλαμβάνει τις δικές του επικεφαλίδες Ethernet, IP και ICMP με τις MAC και τις IP των PCs αυτή τη φορά.

```
▸ Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface eth1, id 0
▸ Ethernet II, Src: aa:c1:ab:38:9e:53 (aa:c1:ab:38:9e:53), Dst: aa:c1:ab:59:d6:63 (aa:c1:ab:59:d6:63)
▸ Internet Protocol Version 4, Src: 10.1.1.1, Dst: 10.1.1.2
▸ User Datagram Protocol, Src Port: 59347, Dst Port: 4789
▸ Virtual eXtensible Local Area Network
▸ Ethernet II, Src: aa:c1:ab:be:ab:f8 (aa:c1:ab:be:ab:f8), Dst: aa:c1:ab:77:a4:cb (aa:c1:ab:77:a4:cb)
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.2
▸ Internet Control Message Protocol
```

Κάνοντας καταγραφή στην eth2 του router1 παρατηρείται ότι τα πακέτα που στέλνει ο PC1 λόγω του ping δεν έχουν λάβει ακόμα UDP και VXLAN header ενώ αντίστοιχα τα πακέτα που επιστρέφονται σε αυτόν ως απάντηση στο ping από τον PC2 δεν περιλαμβάνουν πλέον τα UDP και VXLAN headers που είχαν στην eth1 αφού ο router1 λειτουργεί ως VTEP και έχει εκτελέσει το decapsulation ώστε να φτάσει το αρχικό πακέτο αναλλοίωτο στον PC1.

```
▸ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth2, id 0
▸ Ethernet II, Src: aa:c1:ab:77:a4:cb (aa:c1:ab:77:a4:cb), Dst: aa:c1:ab:be:ab:f8 (aa:c1:ab:be:ab:f8)
▸ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.1
▸ Internet Control Message Protocol
```

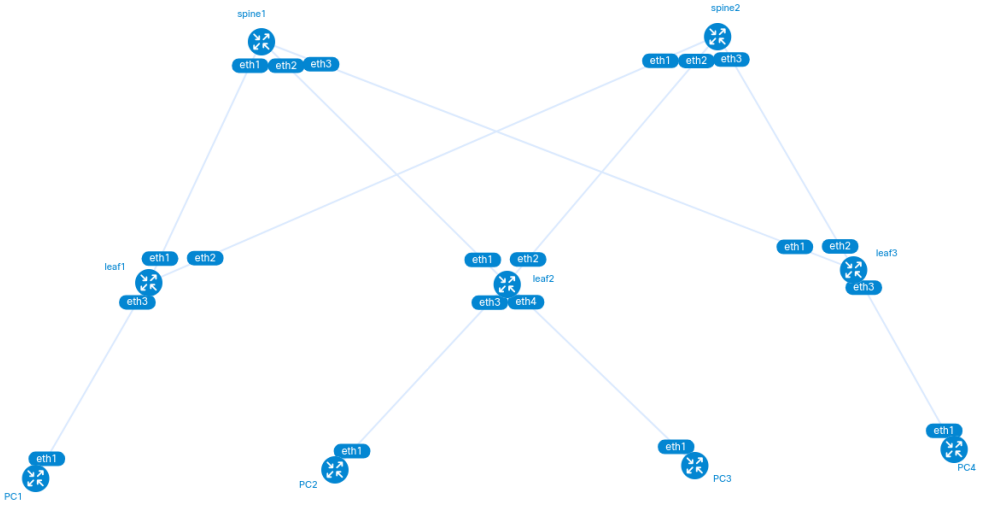
Μέσω του Edgeshark εμφανίζεται ένα αναλυτικό σχεδιάγραμμα το οποίο διασαφηνίζει πως είναι δομημένες οι συνδέσεις μεταξύ των διεπαφών και των γεφυρών:



Εικόνα 28: *Διάγραμμα συνδέσεων στο Edgeshark*

8.9 VXLAN_DC

Στην παρούσα τοπολογία αξιοποιείται η τεχνική VXLAN σε συνδυασμό με EVPN-BGP για τη δημιουργία μιας Spine-Leaf τοπολογίας παρόμοια με αυτή ενός Data Center. Σύμφωνα με το σενάριο που υλοποιείται 4 PCs που ανήκουν ανά δύο σε διαφορετικά υποδίκτυα επικοινωνούν επιτυχώς μεταξύ τους μέσω μιας τοπολογίας με αρχιτεκτονική βασισμένη σε δύο spines και τρία leaves, καθένα από τα οποία συνδέεται και στα δύο spines.
Ακολουθεί σχηματική αναπαράσταση:



Εικόνα 29: *Σχηματική αναπαράσταση τοπολογίας LAB_VXLAN_DC*

Εντός του εκτελέσιμου αρχείου setup.sh αποδίδονται στα PC IP διευθύνσεις και ορίζονται προεπιλεγμένες πύλες μέσω των vxlan διεπαφών των δρομολογητών ο ορισμός και η παραμετροποίηση των οποίων αναλύεται παρακάτω . Τα PC1, PC2 ανήκουν στο 192.168.1.0/24 ενώ τα PC3, PC4 στο 192.168.2.0/24. Στο ίδιο αρχείο περιέχεται και η VXLAN υλοποίηση με τρόπο παρόμοιο με του προηγούμενου παραδείγματος. Σε κάθε leaf δημιουργείται από μια εικονική διεπαφή τύπου VXLAN με όνομα vxlan100 ή vxlan200 ανάλογα με το υποδίκτυο των PCs που θα εξυπηρετεί και αντίστοιχα VNI 100 ή 200. Πιο συγκεκριμένα το leaf1 που συνδέεται με το PC1 το οποίο ανήκει στο 192.168.1.0/24 θα λάβει vxlan100 και VNI 100 ενώ το leaf3 που συνδέεται με το PC4 το οποίο ανήκει στο 192.168.2.0/24 θα λάβει vxlan200 και VNI 200. Το leaf2 που συνδέεται και με το PC2 και με το PC3 τα οποία ανήκουν στα δύο διαφορετικά υποδίκτυα της τοπολογίας θα έχει αντίστοιχα δυο vxlan διεπαφές, μια για το καθένα. Ορίζεται ως local IP η εκάστοτε loopback IP του κάθε leaf και απενεργοποιείται το local learning (nolearning), ώστε να αποφευχθεί η ανεπιθύμητη και περιττή κίνηση που θα προέκυπτε λόγω του MAC flooding με κάθε εμφάνιση μιας άγνωστης MAC. Αντί για αυτή τη μέθοδο η αντιστοίχιση MAC και IP διευθύνσεων γίνεται δυναμικά μέσω EVPN-BGP. Οι απαραίτητες ρυθμίσεις για τη λειτουργία του EVPN-BGP περιλαμβάνονται στα αρχεία παραμετροποίησης των spines, leaves. Τέλος απενεργοποιείται το πρωτόκολλο STP εφόσον δεν υπάρχουν βρόγχοι (loops) και δημιουργείται σε κάθε leaf από μια linux bridge στην οποία προστίθεται η vxlan διεπαφή και η διεπαφή του leaf που συνδέεται με το αντίστοιχο PC.

Όπως προαναφέρθηκε η εκμάθηση και η διανομή των MAC διευθύνσεων μεταξύ των leaves γίνεται μέσω BGP-EVPN και η ρύθμιση του γίνεται ξεχωριστά για κάθε leaf, spine στο bind-mount αρχείο παραμετροποίησης του (leaf1.conf , leaf2.conf, leaf3.conf, spine1.conf, spine2.conf). Σε κάθε leaf και κάθε spine ορίζεται μια loopback IP η οποία θα είναι και το BGP router-id του. Κάθε leaf ανήκει σε ένα ξεχωριστό AS (leaf1 → 65001, leaf2 → 65002, leaf3 → 65003) ενώ και τα δύο spine ανήκουν στο 65000. Ορίζεται ένα peer-group SPINE σε κάθε leaf που περιλαμβάνει τους BGP γείτονες του προς τα spines και ένα peer-group LEAF σε κάθε spine με τους BGP γείτονες του προς τα leaves. Κάθε πλευρά ονομάζει το peer-group με βάση τον ρόλο των απέναντι peers. Τα peer groups κάνουν πιο αποδοτική τη λειτουργία του BGP αφού διαμοιράζουν τις διάφορες ρυθμίσεις και τα BGP updates χωρίς να χρειάζεται να επαναλαμβάνονται για κάθε bgp neighbor. Ο χωρισμός στα peer groups SPINE, LEAF αφορά το eBGP πάνω στις φυσικές διεπαφές μεταξύ των leaves, spines μέσω του οποίου επιτυγχάνεται IP connectivity. Επιπλέον ορίζεται ένα peer group FABRIC στο οποίο ανήκουν και τα leaves και τα spines. Το FABRIC αφορά το iBGP πάνω στις loopback διεπαφές και χρησιμεύει για την ανταλλαγή EVPN routes, δηλαδή την αντιστοίχιση MAC-IP-VNI. Στη διαδικασία αυτή τα spines λειτουργούν ως Route Reflectors δηλαδή κάνουν Reflect τα EVPN routes στα leaves ώστε αυτά να μη χρειάζονται direct BGP μεταξύ τους (ή full-mesh τοπολογία).

Στην παραπάνω τοπολογία είναι δυνατή η επικοινωνία μεταξύ όλων των PCs είτε ανήκουν στο ίδιο υποδίκτυο είτε σε διαφορετικό. Παρατίθενται ενδεικτικά τρία επιτυχή ping από το PC1 προς τα PC2, PC3, PC4 και ένα επιτυχές ping μεταξύ PC2-PC3:

```
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC1 ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.354 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.115 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.124 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.124 ms
^C
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.115/0.179/0.354/0.100 ms
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC1 ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=61 time=0.542 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=61 time=0.176 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=61 time=0.185 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=61 time=0.168 ms
^C
--- 192.168.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.168/0.267/0.542/0.158 ms
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC1 ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=61 time=0.306 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=61 time=0.109 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=61 time=0.114 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=61 time=0.103 ms
^C
--- 192.168.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.103/0.158/0.306/0.085 ms
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC2 ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=61 time=0.433 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=61 time=0.207 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=61 time=0.276 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=61 time=0.213 ms
^C
--- 192.168.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.207/0.282/0.433/0.091 ms
```

Ο τρόπος ωστόσο με τον οποίο επιτυγχάνεται η μετάδοση των πακέτων διαφέρει στις δύο περιπτώσεις. Στην περίπτωση που τα δυο PC ανήκουν στο ίδιο υποδίκτυο τότε θα έχουν και το ίδιο VNI και τα leaves λειτουργούν σαν switches. Μεταξύ των leaves που συνδέονται στα δύο PCs ανταλλάσσονται ARP request μέσω EVPN, γίνονται έτσι γνωστές οι MAC και οι IP των VTEP και έτσι δημιουργείται ένα Layer 2 VXLAN tunnel μέσω του οποίου επιτυγχάνεται η επικοινωνία. Το γεγονός ότι η μεταγωγή γίνεται στο στρώμα ζεύξης δεδομένων (Layer 2) και δεν υπάρχει δρομολόγηση IP έχει ως αποτέλεσμα να μην εμφανίζονται οι ενδιάμεσες IP μέσω της traceroute.

```
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC1 traceroute 192.168.1.2
traceroute to 192.168.1.2 (192.168.1.2), 30 hops max, 46 byte packets
 1  192.168.1.2 (192.168.1.2)  0.011 ms  0.011 ms  0.006 ms
```

Με καταγραφή στο leaf1 (eth1) φαίνεται η σταδιακή ενθυλάκωση στο αρχικό ICMP πακέτο των VXLAN, μιας επικεφαλίδας UDP για την μεταγωγή των πακέτων μέσω του VXLAN Tunnel.

```
▸ Frame 13: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface eth1, id 0
▸ Ethernet II, Src: aa:c1:ab:6e:67:71 (aa:c1:ab:6e:67:71), Dst: aa:c1:ab:2d:fd:ee (aa:c1:ab:2d:fd:ee)
▸ Internet Protocol Version 4, Src: 172.22.22.4, Dst: 172.22.22.3
▸ User Datagram Protocol, Src Port: 40512, Dst Port: 4789
▸ Virtual eXtensible Local Area Network
▸ Ethernet II, Src: aa:c1:ab:d1:df:cd (aa:c1:ab:d1:df:cd), Dst: aa:c1:ab:9b:e0:3f (aa:c1:ab:9b:e0:3f)
▸ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.1
▸ Internet Control Message Protocol
```

Στη συνέχεια με καταγραφή στην eth3 του leaf2 παρατηρείται πως έχει γίνει το decapsulation και το αρχικό πακέτο φτάνει αναλλοίωτο στο PC2.

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth3, id 0
▸ Ethernet II, Src: aa:c1:ab:9b:e0:3f (aa:c1:ab:9b:e0:3f), Dst: aa:c1:ab:d1:df:cd (aa:c1:ab:d1:df:cd)
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.2
▸ Internet Control Message Protocol
```

Στην δεύτερη περίπτωση που τα PC ανήκουν σε διαφορετικά υποδίκτυα (έστω από PC1 σε PC3) και άρα έχουν διαφορετικά VNIs θα χρειαστεί να γίνει IP routing από το leaf1 και το VNI 100 προς το leaf3 και το VNI 200 το οποίο επιτυγχάνεται μέσω BGP EVPN στο Layer 3. Εφόσον τώρα συντελείται δρομολόγηση πακέτου στο στρώμα δικτύου φαίνονται μέσω της traceroute όλα τα ενδιάμεσα βήματα (hops).

```
akoutsoni@clab:~/topologies/containerlab/vxlan_dc$ sudo docker exec -it clab-vxlan_dc-PC1 traceroute 192.168.2.1
traceroute to 192.168.2.1 (192.168.2.1), 30 hops max, 46 byte packets
 1  192.168.1.100 (192.168.1.100)  0.009 ms  0.009 ms  0.007 ms
 2  172.22.22.1 (172.22.22.1)  0.006 ms  0.009 ms  0.007 ms
 3  172.22.22.5 (172.22.22.5)  0.006 ms  0.010 ms  0.007 ms
 4  192.168.2.1 (192.168.2.1)  0.007 ms  0.010 ms  0.087 ms
```

Αξίζει να σημειωθεί ότι η επιλογή υλοποίησης της παραπάνω δικτυακής τοπολογίας σε περιβάλλον Containerlab με τη χρήση containers προσφέρει ταχύτητα ανάπτυξης και ξεκάθαρη αναπαράσταση της λειτουργίας του VXLAN και του EVPN-BGP αλλά δεν ευνοεί την εξαγωγή ασφαλών συμπερασμάτων για την απόδοση παρόμοιων πραγματικών δικτύων. Αν στη θέση των containers χρησιμοποιούνταν εικονικά μηχανήματα, τότε τα τελικά συμπεράσματα θα προσέγγιζαν καλύτερα πραγματικά σενάρια απόδοσης. Αυτό συμβαίνει διότι κάθε VM διαθέτει τον δικό του πυρήνα (kernel) και δικές του εικονικές κάρτες δικτύου, οπότε οι συνδέσεις μεταξύ spines και leaves προσομοιώνονται πιο ρεαλιστικά και επιτρέπουν την παρατήρηση φαινομένων όπως καθυστερήσεις και απώλειες πακέτων, τα οποία στο περιβάλλον του Containerlab αλλοιώνονται λόγω του κοινού πυρήνα με το host σύστημα.

8.10 VXLAN_DC_OSPF

Μια εναλλακτική υλοποίηση της Spine-leaf τοπολογίας του προηγούμενου παραδείγματος προκύπτει εάν για την επικοινωνία μεταξύ leaves και spines δεν χρησιμοποιηθεί BGP-EVPN αλλά το πρωτόκολλο OSPF. Το αρχείο YAML με την περιγραφή της τοπολογίας θα είναι ακριβώς το ίδιο με πριν ενώ η υλοποίηση του VXLAN παρουσιάζει κάποιες μικρές διαφορές. Η βασικότερη από αυτές είναι ότι θα πρέπει να ενεργοποιηθεί ξανά το local learning αφού πλέον δεν υπάρχει το BGP-EVPN για να χειριστεί δυναμικά την αντιστοίχιση MAC-IP. Προκειμένου να διαχειριστεί σωστά η κίνηση με άγνωστη MAC προορισμού απαιτούνται και κάποιες ακόμα εντολές (bridge fdb append) ώστε να εκχωρηθούν χειροκίνητα οι IP-MAC εγγραφές στους πίνακες προώθησης των γεφυρών.

Οι στατικές εγγραφές στην FDB των leaves καθορίζουν σε ποιο remote leaf πρέπει να σταλούν τα πακέτα με άγνωστο προορισμό. Συγκεκριμένα στο leaf1 εκχωρείται

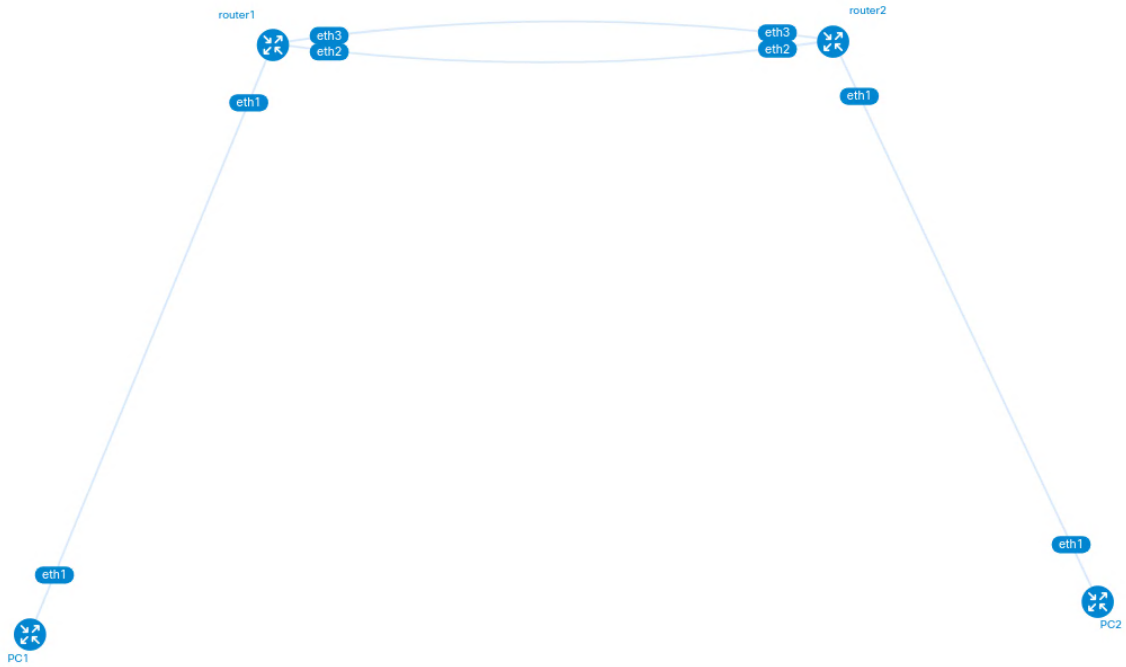
εγγραφή σύμφωνα με την οποία η κίνηση με άγνωστη MAC προορισμού και VNI 100, προωθείται προς το leaf2. Έτσι όλα τα πακέτα από το PC1 μπορούν να φτάνουν στο leaf2 που έχει εγγραφή για το PC2. Στο leaf2 προστίθενται δύο εγγραφές, μια η οποία υποδεικνύει η κίνηση με άγνωστη MAC προορισμού και VNI 100 να προωθείται προς το leaf1 και μια σύμφωνα με την οποία η κίνηση με άγνωστη MAC προορισμού και VNI 200, προωθείται προς το leaf3. Τέλος στον leaf3 εκχωρείται εγγραφή για την αποστολή της κίνησης με άγνωστη MAC προορισμού και VNI 200 στο leaf2.

Μέσω του OSPF διασφαλίζεται ότι κάθε leaf μπορεί να δρομολογήσει κίνηση προς κάθε άλλο leaf. Κάθε leaf και κάθε spine ανακοινώνει την loopback διεύθυνση του και όλα τα υποδίκτυα στα οποία ανήκουν οι διεπαφές του που το συνδέουν με τα υπόλοιπα leaves/spines. Αν κάποια σύνδεση δεν είναι πλέον ενεργή και παρουσιαστεί δυσλειτουργία, οι δρομολογητές ενημερώνουν αυτόματα τους πίνακες διαδρομών τους μέσω των ανακοινώσεων OSPF χωρίς να επηρεαστεί η λειτουργία του VXLAN.

Θα πρέπει να δοθεί ιδιαίτερη προσοχή στο γεγονός ότι εδώ δεν είναι εφικτή η επικοινωνία μεταξύ όλων των τελικών κόμβων όπως στην περίπτωση που χρησιμοποιήθηκε EVPN-BGP. Συγκεκριμένα δεν είναι δυνατή η επικοινωνία μεταξύ PC1 - PC3 και PC1 - PC4. Αιτία αυτού είναι το γεγονός ότι οι κόμβοι σε αυτά τα δύο ζεύγη ανήκουν σε διαφορετικά υποδίκτυα ανήκουν στα οποία αντιστοιχούν και διαφορετικά VXLAN VNIs. Το PC1 ανήκει στο 192.168.1.0/24 με VNI 100, ενώ τα PC3 και PC4 ανήκουν στο 192.168.2.0/24 με VNI 200. Ο μόνος κόμβος που έχει ορατότητα και στα δύο VNIs είναι ο leaf2, αλλά δεδομένου ότι οι εγγραφές της FDB είναι στατικές, δεν υπάρχει μηχανισμός που να προωθεί τα άγνωστα πακέτα από το ένα VNI στο άλλο.
Για να γίνει δυνατή η επικοινωνία μεταξύ PC1 και PC3/PC4 στα πλαίσια του τρέχοντος σεναρίου θα έπρεπε να χρησιμοποιηθεί ο leaf2 να αναλάβει να δρομολογήσει την κίνηση ανάμεσα στα δύο υποδίκτυα σε επίπεδο στρώματος δικτύου.

8.11 ECMP_LAB

Στο παρόν παράδειγμα εξετάζεται η λειτουργία του ECMP. Για το σκοπό αυτό υλοποιείται μια απλή τοπολογία αποτελούμενη από δύο τελικούς κόμβους, η διασύνδεση των οποίων γίνεται μέσω δύο δρομολογητών, οι οποίοι έχουν μεταξύ τους δύο διαφορετικές συνδέσεις. Για τους δρομολογητές χρησιμοποιούνται FRR containers ενώ για του τερματικούς κόμβους (PCs) το pktgen-container, το image που κατασκευάστηκε για τις ανάγκες της παρούσας διπλωματικής και παρουσιάστηκε σε προηγούμενο κεφάλαιο.



Εικόνα 30: *Σχηματική αναπαράσταση τοπολογίας ECMP_LAB*

Για την παραμετροποίηση όλων των κόμβων της τοπολογίας χρησιμοποιείται ένα εκτελέσιμο αρχείο setup.sh. Τα PC1, PC2 ανήκουν σε διαφορετικά υποδίκτυα (10.0.1.0/24 και 10.0.2.0/24 αντίστοιχα). Στο κάθε PC ορίζεται στατική διαδρομή τέτοια ώστε τα πακέτα που προορίζονται για το υποδίκτυο του άλλου PC να στέλνονται στην άμεσα συνδεδεμένη σε αυτόν διεπαφή του εκάστοτε router, η οποία έχει λάβει IP διεύθυνση από το ίδιο υποδίκτυο (10.0.1.0/24 για την eth1 του router1 και 10.0.2.0/24 για την eth1 του router2). Επιπλέον οι eth2 των δρομολογητών λαμβάνουν IP από το υποδίκτυο 192.168.1.0/30 και οι eth3 από το 192.168.2.0/30. Ενεργοποιείται η λειτουργία προώθησης στους δρομολογητές και θέτεται η MTU των eth1 στα PCs σε 1500 για λόγους συμβατότητας με το Netmap το οποίο από προεπιλογή χρησιμοποιεί buffers των 2KB. Επιπλέον στον router1 προστίθεται μια εγγραφή για το υποδίκτυο του PC2 με δύο ίσου κόστους διαδρομών μέσω της eth2 και της eth3. Για την εγγραφή αυτή θα πρέπει η εντολή να συνταχθεί με τον εξής τρόπο ώστε να τοποθετηθούν στον πίνακα διαδρομών και οι δύο διαδρομές:
sudo docker exec -it clab-ecmp_lab-router1 ip route add 10.0.2.0/24 nexthop via 192.168.1.2 dev eth2 nexthop via 192.168.2.2 dev eth3
Επιβεβαιώνεται το αποτέλεσμα:

```
router1:/# ip route show
default via 172.20.20.1 dev eth0
10.0.1.0/24 dev eth1 proto kernel scope link src 10.0.1.254
10.0.2.0/24
        nexthop via 192.168.1.2 dev eth2 weight 1
        nexthop via 192.168.2.2 dev eth3 weight 1
172.20.20.0/24 dev eth0 proto kernel scope link src 172.20.20.8
192.168.1.0/30 dev eth2 proto kernel scope link src 192.168.1.1
192.168.2.0/30 dev eth3 proto kernel scope link src 192.168.2.1
```

Σε περίπτωση που η προσθήκη των δύο διαδρομών μέσω eth2 και eth3 γινόταν χρησιμοποιώντας δύο διαφορετικές εντολές ως εξής:
sudo docker exec -it clab-ecmp_lab-router1 ip route add 10.0.2.0/24 via 192.168.1.2 dev eth2
sudo docker exec -it clab-ecmp_lab-router1 ip route add 10.0.2.0/24 via 192.168.2.2 dev eth3
τότε η δεύτερη εγγραφή αγνοείται αφού είναι ισάξια με την πρώτη και όπως φαίνεται παρακάτω τοποθετείται στον πίνακα διαδρομών μόνο η πρώτη εγγραφή με αποτέλεσμα να μην λειτουργεί το ECMP.

```
router1:/# ip route show
default via 172.20.20.1 dev eth0
10.0.1.0/24 dev eth1 proto kernel scope link src 10.0.1.254
10.0.2.0/24 via 192.168.1.2 dev eth2
172.20.20.0/24 dev eth0 proto kernel scope link src 172.20.20.8
192.168.1.0/30 dev eth2 proto kernel scope link src 192.168.1.1
192.168.2.0/30 dev eth3 proto kernel scope link src 192.168.2.1
```

Ακολουθείται η αντίστοιχη διαδικασία και για τον router2.

Μετά την ανάπτυξη της τοπολογίας και την εκτέλεση του setup.sh ελέγχεται η λειτουργία του ECMP. Με *sudo docker exec -it clab-ecmp_lab-router1 bash* μπορεί να γίνει είσοδος στο command line του router1 και αντίστοιχα του router2 και στη συνέχεια χρησιμοποιείται η εντολή *apk add bmon* για να εγκατασταθεί στους δρομολογητές το bmon (Bandwidth Monitor), ένα εργαλείο παρακολούθησης εύρους ζώνης δικτύου ώστε να είναι εφικτή η παρακολούθηση σε πραγματικό χρόνο της εισερχόμενης και την εξερχόμενης κίνησης που διέρχεται από κάθε διεπαφή αλλά και πληροφορίες για errors, dropped packets, collisions.
Στη συνέχεια εντός των PC1, PC2 χρησιμοποιούνται οι παρακάτω εντολές pkt-gen ώστε να παραχθεί κίνηση στο ένα PC και λήψη της από το άλλο. Η κίνηση αυτή θα περιλαμβάνει πακέτα με πολλαπλές IP πηγής και προορισμού ενεργοποιώντας έτσι το ECMP αφού ένα απλό ping μεταξύ των PCs δεν θα ήταν αρκετό για να αναδείξει τη λειτουργία του ECMP καθώς έχει σταθερές IP και σταθερές θύρες και άρα δημιουργείται μοναδική ροή που μεταδίδεται όλη από το ίδιο πάντα μονοπάτι.

Παραγωγή την κίνησης στο PC1:

pkt-gen -i eth1 -f tx -n 8000000 -l 60 -d 10.0.2.1:2000-10.0.2.254 -D aa:c1:ab:f5:15:11 -s 10.0.1.1:2000-10.0.1.254 -S aa:c1:ab:85:18:34 -w 4 -R 20000

Η παραπάνω εντολή ορίζει ότι ο PC1 βρίσκεται σε λειτουργία μετάδοσης (tx: transmit mode) και παράγει 8 εκατομμύρια πακέτα, μεγέθους 60 bytes το καθένα, τα οποία στέλνονται από τη eth1 διεπαφή του PC1. Η IP πηγής του κάθε πακέτου επιλέγεται τυχαία από το εύρος 10.0.1.1-10.0.1.254 (δηλαδή όλες τις διευθύνσεις εντός του υποδικτύου του PC1) και έχει σταθερή θύρα 2000 ενώ η IP προορισμού επιλέγεται τυχαία από το εύρος 10.0.2.1-10.0.2.254 (δηλαδή όλες τις διευθύνσεις εντός του υποδικτύου του PC2) και έχει επίσης σταθερή θύρα 2000. Προσδιορίζονται επιπλέον οι διευθύνσεις MAC πηγής και προορισμού ώστε να διασφαλιστεί η σωστή

δρομολόγηση στο στρώμα ζεύξης δεδομένων. Ως MAC πηγής ορίζεται η MAC διεύθυνση της eth1 του PC1 ενώ ως MAC προορισμού η MAC της eth1 του router1. Στον ρυθμό αποστολή αποδίδεται η τιμή των 20000 πακέτων το δευτερόλεπτο και εισάγεται καθυστέρηση 4 δευτερολέπτων πριν την έναρξη αποστολής ώστε να εξασφαλιστεί η ετοιμότητα της σύνδεσης.

Λήψη της κίνησης στο PC2:

pkt-gen -i eth1 -f rx

Από την άλλη ο PC2 βρίσκεται σε λειτουργία υποδοχής πακέτων (rx: receive mode) μέσω της διεπαφής eth1 και μετρά πόσα πακέτα λαμβάνονται, με τι ρυθμό, εμφανίζοντας τα σε πραγματικό χρόνο χωρίς να περιορίζει τον αριθμό των πακέτων που λαμβάνονται.

Ακολουθούν στιγμιότυπα από την παραπάνω διαδικασία.

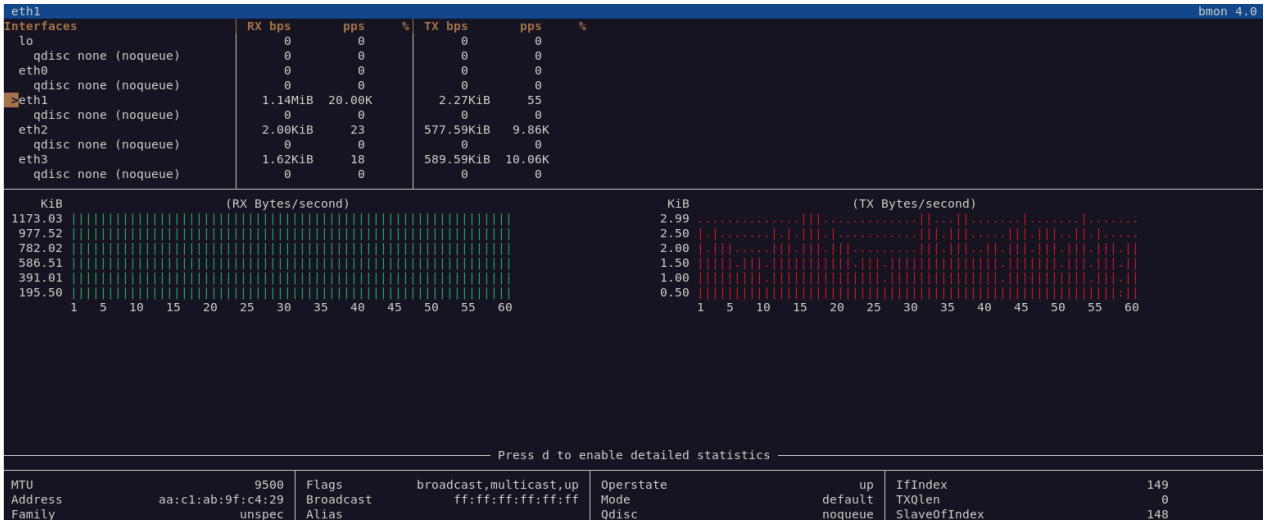
PC1**:**

```
root@PC1:/opt/netmap/LINUX/apps# pkt-gen -i eth1 -f tx -n 8000000 -l 60 -d 10.0.2.1:2000-10.0.2.254 -D aa:c1:ab:9f:c4:29 -s 10.0.1.1:2000-10.0.1.254 -S aa:c1:ab:99:e9:ad -
w 4 -R 20000
939.552665 main [3036] interface is eth1
939.552733 main [3159] using default burst size: 512
939.552787 main [3167] running on 1 cpus (have 8)
939.552828 extract_ip_range [477] range is 10.0.1.1:2000 to 10.0.1.254:2000
939.552839 extract_ip_range [477] range is 10.0.2.1:2000 to 10.0.2.254:2000
Sending on netmap:eth1: 2 queues, 1 threads and 1 cpus.
10.0.1.1:2000-10.0.1.254 -> 10.0.2.1:2000-10.0.2.254 (aa:c1:ab:99:e9:ad -> aa:c1:ab:9f:c4:29)
939.553360 main [3342] --- SPECIAL OPTIONS: copy

939.553366 main [3369] Sending 66 packets every  0.003300000 s
939.553446 start_threads [2695] Wait 4 secs for phy reset
943.553554 start_threads [2697] Ready...
943.553740 sender_body [1702] start, fd 3 main_fd 3
944.554735 main_thread [2781] 0.000 pps (0.000 pkts 0.000 bps in 1001038 usec) 0.00 avg_batch 0 min_space
945.000012 sender_body [1760] frags 1 frag_size 60
945.555792 main_thread [2781] 11.076 Kpps (11.088 Kpkts 5.317 Mbps in 1001056 usec) 66.00 avg_batch 99999 min_space
946.556359 main_thread [2781] 19.987 Kpps (19.998 Kpkts 9.594 Mbps in 1000568 usec) 66.00 avg_batch 99999 min_space
947.557440 main_thread [2781] 19.976 Kpps (19.998 Kpkts 9.589 Mbps in 1001080 usec) 66.00 avg_batch 99999 min_space
948.557575 main_thread [2781] 19.995 Kpps (19.998 Kpkts 9.598 Mbps in 1000136 usec) 66.00 avg_batch 99999 min_space
949.558639 main_thread [2781] 20.043 Kpps (20.064 Kpkts 9.620 Mbps in 1001063 usec) 66.00 avg_batch 99999 min_space
950.559279 main_thread [2781] 19.985 Kpps (19.998 Kpkts 9.593 Mbps in 1000640 usec) 66.00 avg_batch 99999 min_space
951.559909 main_thread [2781] 19.985 Kpps (19.998 Kpkts 9.593 Mbps in 1000631 usec) 66.00 avg_batch 99999 min_space
952.561130 main_thread [2781] 20.040 Kpps (20.064 Kpkts 9.619 Mbps in 1001221 usec) 66.00 avg_batch 99999 min_space
953.561553 main_thread [2781] 19.990 Kpps (19.998 Kpkts 9.595 Mbps in 1000422 usec) 66.00 avg_batch 99999 min_space
954.562625 main_thread [2781] 19.977 Kpps (19.998 Kpkts 9.589 Mbps in 1001073 usec) 66.00 avg_batch 99999 min_space
955.563697 main_thread [2781] 19.977 Kpps (19.998 Kpkts 9.589 Mbps in 1001072 usec) 66.00 avg_batch 99999 min_space
956.564762 main_thread [2781] 20.043 Kpps (20.064 Kpkts 9.620 Mbps in 1001064 usec) 66.00 avg_batch 99999 min_space
957.565827 main_thread [2781] 19.977 Kpps (19.998 Kpkts 9.589 Mbps in 1001066 usec) 66.00 avg_batch 99999 min_space
958.566899 main_thread [2781] 19.977 Kpps (19.998 Kpkts 9.589 Mbps in 1001071 usec) 66.00 avg_batch 99999 min_space
```

PC2:

```
root@PC2:/opt/netmap/LINUX/apps# pkt-gen -i eth1 -f rx
940.750240 main [3036] interface is eth1
940.750316 main [3159] using default burst size: 512
940.750367 main [3167] running on 1 cpus (have 8)
940.750588 extract_ip_range [477] range is 10.0.0.1:1234 to 10.0.0.1:1234
940.750600 extract_ip_range [477] range is 10.1.0.1:1234 to 10.1.0.1:1234
Receiving from netmap:eth1: 2 queues, 1 threads and 1 cpus.
940.751167 start_threads [2695] Wait 2 secs for phy reset
942.751269 start_threads [2697] Ready...
942.751496 receiver_body [1939] reading from netmap:eth1 fd 3 main_fd 3
943.752457 main_thread [2781] 0.000 pps (0.000 pkts 0.000 bps in 1001054 usec) 0.00 avg_batch 0 min_space
943.752565 receiver_body [1954] waiting for initial packets, poll returns 0 0
944.753550 main_thread [2781] 0.000 pps (0.000 pkts 0.000 bps in 1001093 usec) 0.00 avg_batch 99999 min_space
944.753647 receiver_body [1954] waiting for initial packets, poll returns 0 0
945.754614 main_thread [2781] 311.000 pps (311.000 pkts 140.778 Kbps in 1001064 usec) 2.36 avg_batch 1018 min_space
946.755396 main_thread [2781] 136.000 pps (136.000 pkts 45.660 Kbps in 1000782 usec) 1.33 avg_batch 1021 min_space
947.756493 main_thread [2781] 212.000 pps (212.000 pkts 71.154 Kbps in 1001097 usec) 1.77 avg_batch 1018 min_space
948.757554 main_thread [2781] 490.000 pps (491.000 pkts 201.195 Kbps in 1001061 usec) 2.48 avg_batch 1016 min_space
949.758625 main_thread [2781] 233.000 pps (233.000 pkts 78.204 Kbps in 1001070 usec) 1.79 avg_batch 1018 min_space
950.759679 main_thread [2781] 235.000 pps (235.000 pkts 78.877 Kbps in 1001055 usec) 2.03 avg_batch 1018 min_space
951.760762 main_thread [2781] 485.000 pps (486.000 pkts 199.512 Kbps in 1001083 usec) 2.72 avg_batch 992 min_space
952.761855 main_thread [2781] 239.000 pps (239.000 pkts 80.216 Kbps in 1001093 usec) 1.80 avg_batch 1018 min_space
953.762926 main_thread [2781] 233.000 pps (233.000 pkts 78.204 Kbps in 1001071 usec) 1.82 avg_batch 1018 min_space
954.763991 main_thread [2781] 234.000 pps (234.000 pkts 78.540 Kbps in 1001064 usec) 1.97 avg_batch 1018 min_space
955.765054 main_thread [2781] 240.000 pps (240.000 pkts 80.554 Kbps in 1001064 usec) 1.79 avg_batch 1018 min_space
```

Με την εντολή bmon εντός των δρομολογητών παρακολουθείται σε πραγματικό χρόνο η κίνηση μεταξύ PC1 – PC2 και επιβεβαιώνεται η λειτουργία του ECMP. Όπως φαίνεται παρακάτω στον router1, η κίνηση που στέλνει ο PC1 λαμβάνεται ολόκληρη από την eth1 (1.14MiB (bps) – 20.00K (pps)) και έπειτα μοιράζεται ισόποσα στις eth2, eth3 (577.59+589.59KiB (bps) – 9.86+10.06K (pps))



Εικόνα 31: *Παρακολούθηση κίνησης μέσω του bmon*

9. Κατανάλωση πόρων

Στο κεφάλαιο αυτό μελετάται η επίδραση της λειτουργίας του Containerlab στο λειτουργικό σύστημα ως προς την κατανάλωση πόρων. Στόχος είναι η αξιολόγηση των δυνατοτήτων κλιμάκωσης και ευρείας χρήσης του Containerlab από πολλαπλούς χρήστες ταυτόχρονα και με διαφορετικά σενάρια χρήσης.

9.1 Στόχος και Μεθοδολογία

Στα πλαίσια της παρούσας μελέτης, αρχικά δημιουργήθηκε ένας κύριος χρήστης, στον οποίο αναπτύχθηκαν και δοκιμάστηκαν όλες οι τοπολογίες του προηγούμενου κεφαλαίου (11 στο σύνολο). Στη συνέχεια, προκειμένου να ελεγχθεί η συμπεριφορά του συστήματος υπό συνθήκες αυξημένου φόρτου, δημιουργήθηκαν τρεις επιπλέον χρήστες. Σε κάθε έναν από αυτούς αντιγράφηκαν σταδιακά οι ίδιες τοπολογίες, ώστε να προσομοιωθεί ένα σενάριο ταυτόχρονης αξιοποίησης ενός φυσικού εξυπηρετητή από πολλαπλούς χρήστες με παράλληλη εκτέλεση διεργασιών στο Containerlab. Η συνθήκη αυτή εμφανίζεται κατά κόρον σε πραγματικές ερευνητικές, εργαστηριακές ή εκπαιδευτικές εγκαταστάσεις, όπου πολλοί ερευνητές ή φοιτητές μοιράζονται την ίδια υποδομή. Συνεπώς η αξιολόγηση του περιβάλλοντος του Containerlab στα πλαίσια μιας τέτοιας κατάστασης κρίνεται απαραίτητη για την απόκτηση μιας ολοκληρωμένης εικόνας για την χρηστικότητα και τις δυνατότητες του.

Η διερεύνηση που ακολουθεί στοχεύει μέσω της παρατήρησης στατιστικών, της καταγραφής μετρήσεων και της ανάλυσης αυτών των αποτελεσμάτων να απαντήσει σε ερωτήματα σχετικά με τη μεταβολή της κατανάλωσης CPU, RAM συναρτήσει του αριθμού ενεργών τοπολογιών και χρηστών, την χρήση αποθηκευτικού χώρου και την ενδεχόμενη διαφοροποίηση στην κατανάλωση πόρων ανάλογα με το αν ο χρήστης είναι ενεργός ή σε αδράνεια και άρα η λειτουργία του Containerlab στο παρασκήνιο.

Για σαφήνεια και πιο ολοκληρωμένη ερμηνεία των αποτελεσμάτων παρατίθεται ένας πίνακας με τις προδιαγραφές του συστήματος στο οποίο έγιναν τα πειράματα.

Operation System	Ubuntu 22.04.5 LTS
CPU Architecture	x86_64
CPUs	8
CPU Clock Speed	2.40 GHz
Cache	L1: 512 KiB , L2: 2 MiB, L3: 120 MiB
RAM	15.6 GiB ≈ 16.8 GB
Swap Memory	4 GiB ≈ 4.3 GB
Disk Storage	160 GB (2 GB /boot partition, 100 GB root LVM partition)

Εικόνα 32: *Προδιαγραφές συστήματος*

9.2 Υλοποίηση πολλαπλών σεναρίων χρήσης

Σενάριο 1: Ένας ενεργός χρήστης – 3 τοπολογίες

Εικόνα 33: *Στιγμιότυπο System Monitor για το Σενάριο 1: Ένας ενεργός χρήστης - 3 τοπολογίες*

Σενάριο 2: Ένας ενεργός χρήστης – 6 τοπολογίες

Εικόνα 34: *Στιγμιότυπο System Monitor για το Σενάριο 2: Ένας ενεργός χρήστης - 6 τοπολογίες*

Σενάριο 3: Ένας ενεργός χρήστης – 11 τοπολογίες

Εικόνα 35: *Στιγμιότυπο System Monitor για το Σενάριο 3: Ένας ενεργός χρήστης - 11 τοπολογίες*

Σενάριο 4: Δύο ενεργοί χρήστες – 11 τοπολογίες ο ένας και καμία ο άλλος

Εικόνα 36: *Στιγμιότυπο System Monitor για το Σενάριο 4: Δύο ενεργοί χρήστες - 11 τοπολογίες ο ένας και καμία ο άλλος*

Σενάριο 5: Δύο ενεργοί χρήστες – 11 τοπολογίες ο καθένας

Εικόνα 37: *Στιγμιότυπο System Monitor για το Σενάριο 5: Δύο ενεργοί χρήστες – 11 τοπολογίες ο καθένας*

Σενάριο 6: Ένας ενεργός χρήστης και ένας αδρανής – 11 τοπολογίες ο καθένας

Εικόνα 38: *Στιγμιότυπο System Monitor για το Σενάριο 6: Ένας ενεργός χρήστης και ένας αδρανής – 11 τοπολογίες ο καθένας*

Σενάριο 7: Τρεις ενεργοί χρήστες – 11 τοπολογίες ο καθένας

(πριν την προσθήκη των τοπολογιών στον 3ο χρήστη):

(μετά):

Εικόνα 39: *Στιγμιότυπο System Monitor για το Σενάριο 7: Τρεις ενεργοί χρήστες – 11 τοπολογίες ο καθένας*

Σενάριο 8: Δύο ενεργοί χρήστες και ένας αδρανής – 11 τοπολογίες ο καθένας

Εικόνα 40: *Στιγμιότυπο System Monitor για το Σενάριο 8: Δύο ενεργοί χρήστες και ένας αδρανής – 11 τοπολογίες ο καθένας*

Σενάριο 9: Τέσσερεις ενεργοί χρήστες – 11 τοπολογίες ο καθένας

(πριν την προσθήκη των τοπολογιών)

(μετά)

Εικόνα 41: *Στιγμιότυπο System Monitor για το Σενάριο 9: Τέσσερεις ενεργοί χρήστες – 11 τοπολογίες ο καθένας*

Σενάριο 10: Τρεις ενεργοί χρήστες και ένας αδρανής – 11 τοπολογίες ο καθένας

Εικόνα 42: *Στιγμιότυπο System Monitor για το Σενάριο 10: Τρεις ενεργοί χρήστες και ένας αδρανής – 11 τοπολογίες ο καθένας*

Τα αριθμητικά δεδομένα των παραπάνω εικόνων συνοψίζονται στον παρακάτω πίνακα.

Σενάριο	RAM (%)	Cache (%)	Swap (%)	CPU1 (%)	CPU2 (%)	CPU3 (%)	CPU4 (%)	CPU5 (%)	CPU6 (%)	CPU7 (%)	CPU8 (%)
1	13.9	11.1	0.1	5.1	11.8	15	8	11.8	3	4	4
2	15.8	11.1	0.1	4	13.1	27.5	4	12.7	4	5	3
3	20.4	11.2	0.1	2.8	22.8	3.7	9.3	4.6	2.8	22.4	2.8
4	24.5	11.2	0.1	4.6	27.8	18.5	7.3	4.6	3.7	3.7	10.2
5	32.1	11.2	0.1	4.6	18	6.7	4.6	2.7	2	46.1	4
6	27.8	11.1	0.1	7.8	5.2	8.5	7.9	45.8	2.6	2.6	2.6
7	44.2	9.4	2.8	3.2	10.5	2.3	8.1	3.2	2.3	3.1	63.3
8	39.7	10	1.1	3.2	5	62	9.2	4.1	4.6	6.9	3.2
9	56.3	7.5	6.5	15.4	46.9	18.4	3.1	5.3	4.1	9.5	5
10	51.4	7.5	6.3	10	2.4	9.1	3.3	14.6	57.4	1.8	1.2

Εικόνα 43: *Συγκεντρωτικά αριθμητικά δεδομένα από το System Monitor για όλα τα σενάρια*

9.3 Ανάλυση Αποτελεσμάτων

Με βάση τα παραπάνω αποτελέσματα η μνήμη RAM αποδεικνύεται να είναι ο πιο ευαίσθητος πόρος με τη μεγαλύτερη μεταβολή κατά την κλιμάκωση χρηστών και τοπολογιών. Παρατηρείται ότι στα αρχικά σενάρια στα οποία υπάρχει μόνος ένας χρήστης και λίγες τοπολογίες η χρήση της είναι ιδιαίτερα χαμηλή, αλλά αυξάνεται γρήγορα καθώς προστίθενται περισσότερες τοπολογίες ή χρήστες, ξεπερνώντας τελικά στην ταυτόχρονη εργασία 4 χρηστών στο Containerlab ακόμα και το 55% δέσμευσης της συνολικής διαθέσιμης. Η αύξηση της είναι σχεδόν γραμμική. Σημειώνεται αύξηση κατά 4-5% κάθε φορά που ένας νέος χρήστης κάνει είσοδο στο σύστημα (log in) και παραμένει ενεργός και αύξηση κατά 6-8% με την προσθήκη των 11 τοπολογιών σε έναν επιπλέον χρήστη. Σε περίπτωση αδράνειας ενός χρήστη (log out) αποδεσμεύεται αυτό το 4-5% που προκύπτει από την είσοδο ενός νέου χρήστη αλλά ο φόρτος που επιφέρουν οι τοπολογίες (6-8%) παραμένει, γεγονός αναμενόμενο αφού τα containers τους συνεχίζουν να είναι ενεργά. Συμπεραίνοντας, η μνήμη RAM αποτελεί έναν αξιόλογο περιοριστικό παράγοντα σε ενδεχόμενη επιθυμητή μεγάλη αύξηση χρηστών και τοπολογιών που πιέζει μεν το σύστημα αλλά όχι σε απαγορευτικό βαθμό. Παρόλο που στην παράλληλη εργασία 4 χρηστών καταναλώνεται πάνω από τη μισή RAM, το σύστημα ανταποκρίνεται ικανοποιητικά και τα containers λειτουργούν χωρίς πρόβλημα, ενώ τα αριθμητικά δεδομένα υποδεικνύουν ότι θα μπορούσε να αντέξει ακόμα περίπου 2 χρήστες με παρόμοια λειτουργία. Συνεπώς η συνολική απόδοση του συστήματος αναφορικά με την κατανάλωση μνήμης RAM αξιολογείται ως αρκετά καλή και υψηλή.

Η συμπεριφορά της Cache ακολουθεί μια αντίστροφη πορεία. Όσο η συνολική επιβάρυνση του συστήματος με επιπλέον χρήστες και τοπολογίες παραμένει σχετικά χαμηλή η Cache διατηρείται σε υψηλά επίπεδα κοντά στο 11%. Με την αύξηση του φόρτου και την κατανάλωση περισσότερης RAM το σύστημα ανακατανέμει τους διαθέσιμους πόρους και μειώνει το ποσοστό της Cache, φτάνοντας μέχρι και το 7.5% στο σενάριο με τους 4 χρήστες. Στόχος αυτής της διαδικασίας είναι η απελευθέρωση μνήμης για την κάλυψη των αναγκών των containers, γεγονός που υποδηλώνει την πίεση που υφίσταται το σύστημα αλλά συμβάλλει σημαντικά στην αρμονική λειτουργία των containers.

Η χρήση της μνήμης Swap είναι ανάλογη της χρήσης της RAM. Στα αρχικά σενάρια που το σύστημα δεν υφίσταται ουσιώδη πίεση η Swap παραμένει πρακτικά μηδενική υποδεικνύοντας ότι η RAM επαρκεί, ενώ με την αύξηση του φορτίου ξεκινά να χρησιμοποιείται σταδιακά φτάνοντας μέχρι και το 6.5% χρήσης. Πρακτικά αυτό σημαίνει ότι το σύστημα αναγκάζεται να μεταφέρει δεδομένα, συνήθως όσα δεν χρησιμοποιούνται ενεργά σε μια δεδομένη χρονική στιγμή, στο δίσκο προκειμένου να συνεχίσει να εξυπηρετεί τα containers.

Σχετικά με τη χρήση της CPU σημειώνεται αρχικά ότι δεν υπάρχει ισομερής κατανομή σε όλες τις CPUs και ότι ο καταμερισμός αλλάζει διαρκώς οπότε τα παραπάνω στιγμιότυπα περιέχουν ενδεικτικές τιμές αλλά αντικατοπτρίζουν ικανοποιητικά την συνολική χρήση της CPU η οποία είναι ανά σενάριο σχετικά σταθερή. Η κατανάλωση CPU αυξάνεται σε γενικές γραμμές όσο αυξάνονται και οι χρήστες και ο αριθμός των τοπολογιών και εμφανίζει απότομες μεταβολές (spikes στα γραφήματα) τις στιγμές που μια νέα τοπολογία γίνεται deploy ή ένας νέος χρήστης κάνει log in. Η ανομοιόμορφη κατανομή του φορτίου στις 8 CPUs μπορεί να οδηγήσει σε τοπική συμφόρηση και περιορισμό της συνολικής απόδοσης.

Συνολικά, η αξιολόγηση των παραπάνω αποτελεσμάτων οδηγεί στο συμπέρασμα ότι το Containerlab αποτελεί ένα ιδιαίτερα αποδοτικό και ευέλικτο εργαλείο, ικανό να διαχειριστεί πολλαπλές και σύνθετες τοπολογίες χωρίς να επιβαρύνει υπερβολικά το σύστημα με καταναλώνοντας απαγορευτικά μεγάλο μέρος των διαθέσιμων πόρων. Επιπλέον το σύστημα ανταποκρίνεται ομαλά σε σενάρια με ταυτόχρονη χρήση από πολλούς χρήστες, διατηρώντας σταθερή τη λειτουργία του και χωρίς να εμφανίζονται κρίσιμα σημεία κατάρρευσης.

Η συνολική εικόνα αναδεικνύει το Containerlab ως μια αξιόπιστη λύση για την ανάπτυξη και αξιολόγηση δικτυακών πειραμάτων σε εικονικό περιβάλλον, προσφέροντας υψηλή αποδοτικότητα και κλιμακωσιμότητα που το καθιστούν κατάλληλο για ερευνητική και εκπαιδευτική χρήση.

10. Σύνοψη και μελλοντικές επεκτάσεις

10.1 Σύνοψη

Η παρούσα διπλωματική εργασία είχε ως αντικείμενο τη μελέτη και την αξιολόγηση του Containerlab ως περιβάλλοντος εξομοίωσης δικτυακών τοπολογιών με τη χρήση τεχνολογιών ελαφράς εικονικοποίησης. Μέσα από τη θεωρητική ανάλυση, την πρακτική υλοποίηση και την αποτίμηση της κατανάλωσης πόρων, αναδείχθηκαν πολλαπλές διαστάσεις της λειτουργίας του, τα πλεονεκτήματα αλλά και οι προκλήσεις που συνοδεύουν τη χρήση του.

Τα κύρια συμπεράσματα που εξήχθησαν είναι τα εξής:

Αποδοτικότητα

Η συνολική αποτίμηση του Containerlab μετά τα πολλαπλά σενάρια χρήσης που δοκιμάστηκαν είναι ιδιαίτερη θετική καθώς παρατηρείται αρκετά χαμηλή κατανάλωση πόρων αναλογικά με το φόρτο εργασίας που του προστέθηκε. Οι μικροί χρόνοι εκκίνησης, η χαμηλή χρήση μνήμης και η περιορισμένη κατανάλωση CPU το καθιστούν ένα ιδιαίτερα αποδοτικό εργαλεία, κατάλληλο για ακαδημαϊκή χρήση, η οποία απαιτεί συχνά την ταυτόχρονη εργασία πολλών φοιτητών σε κοινόχρηστους πόρους.

Ευελιξία σχεδιασμού και παραμετροποίησης

Μέσω των αρχείων YAML, ο σχεδιασμός και η εξατομίκευση του εξομοιώμενου δικτύου καθίσταται απλός και εύχρηστος επιτρέποντας στο χρήστη να υλοποιεί πολύπλοκα σενάρια αλλά και να καταστρέφει τις τοπολογίες του και να τις επαναδημιουργεί με ευκολία. Το γεγονός αυτό ενισχύει με ουσιαστικό τρόπο τον πειραματισμό και τον έλεγχο πραγματικών σεναρίων σε ένα ευέλικτο περιβάλλον.

Υποστήριξη πολλαπλών δικτυακών τεχνολογιών

Η πειραματική μελέτη απέδειξε ότι το Containerlab υποστηρίζει πληθώρα δικτυακών πρωτοκόλλων και τεχνολογιών διευρύνοντας έτσι το πεδίο εφαρμογής του, το οποίο έχει έκταση από απλές τοπολογίες στα πλαίσια εισαγωγικής εκπαιδευτικής διαδικασίας, μέχρι εξειδικευμένες ερευνητικές εφαρμογές.

Συμβατότητα με άλλα εργαλεία

Η δυνατότητα ενσωμάτωσης διαφόρων εργαλείων και προγραμμάτων διευκολύνει την παρακολούθηση και ανάλυση της κίνησης για διαγνωστικούς σκοπούς και δίνει στους χρήστες την επιλογή να επεκτείνουν τις πειραματικές δοκιμές τους και να συνδυάσουν τεχνικές και λειτουργίες.

Περιορισμοί και Προκλήσεις

Η χρήση containers βασίζεται στον διαμοιρασμό του πυρήνα του λειτουργικού συστήματος, γεγονός που θέτει ζητήματα ασφαλείας και σταθερότητας, καθώς σε περίπτωση κάποιας αδυναμίας σε επίπεδο πυρήνα καθίστανται ευάλωτα όλα τα containers.

Μια από τις προκλήσεις που αντιμετωπίζει το Containerlab είναι η εξάρτηση του σε μεγάλο βαθμό σε άλλες εξωτερικές ήδη υπάρχουσες τεχνολογίες, όπως το Docker και οι εκάστοτε πάροχοι των image (FRRouting, Nokia, Arista) με αποτέλεσμα να υπάρχει μεγάλη ανάγκη συντήρησης και συνεχούς ενημέρωσης ώστε να διατηρείται η συμβατότητα.

Τέλος ο πρακτικός περιορισμός χρήσης του μόνο σε Linux-based συστήματα μειώνει σημαντικά το διαθέσιμο κοινό χρηστών ή απαιτεί την ύπαρξη εργαλείων όπως το WSL ή εικονικά μηχανήματα προκειμένου να λειτουργήσει σε άλλα συστήματα (Windows, macOS) με άμεσο αποτέλεσμα την κατανάλωση επιπλέον πόρων και την εξασθένιση της απλότητας και ευελιξίας της χρήσης του.

10.2 Μελλοντικές Επεκτάσεις

Ακολουθούν κάποιες μελλοντικές επεκτάσεις, οι οποίες θα μπορούσαν να διευρύνουν την μελέτη γύρω από το Containerlab και να αναδείξουν νέες πτυχές του.

Πειραματική σύγκριση με άλλες πλατφόρμες

Προκειμένου να αναδειχθούν σαφέστερα και εκτενέστερα τα οφέλη και οι περιορισμοί του Containerlab θα μπορούσε να διεξαχθεί μια πειραματική σύγκριση της συμπεριφοράς του υπό τις ίδιες συνθήκες με αυτή άλλων πλατφορμών που εξυπηρετούν σε γενικές γραμμές τον ίδιο σκοπό αλλά λειτουργούν διαφορετικά, όπως το Kubernetes, το GNS3 και το EVE-NG.

Μελέτη απόδοσης σε μεγάλη κλίμακα

Η απόδοση του Containerlab εξετάστηκε μεν στα πλαίσια της παρούσας διπλωματικής σε πολλαπλά σενάρια αλλά δεν έφτασε σε υψηλά επίπεδα κλιμάκωσης και επεκτασιμότητας. Μελλοντική έρευνα εστιασμένη στην παρατήρηση της κατανάλωσης πόρων και τις ανταπόκρισης του συστήματος σε συνθήκες πολύ υψηλού φόρτου με πολύ μεγάλο αριθμό πολύπλοκων τοπολογιών και αυξημένο αριθμό χρηστών θα οδηγούσε σε μια σαφέστερη αξιολόγηση των ορίων του Containerlab.

Διερεύνηση μηχανισμών ασφαλείας

Όπως προαναφέρθηκε η χρήση containers επιφέρει ζητήματα ασφαλείας. Μια εκτενής εξέταση των μηχανισμών που διαθέτει το Containerlab για πρόληψη και αντιμετώπιση τέτοιων θεμάτων θα αποτελούσε μια χρήσιμη επέκταση, η οποία θα προσέφερε όχι μόνο τη γνώση στο χρήστη αλλά θα μπορούσε και να συνεισφέρει στην διόρθωση ενδεχόμενων προβλημάτων, αφού το Containerlab αποτελεί ένα λογισμικό ανοιχτού κώδικα.

Βιβλιογραφία

[1]“Containerlab: A free and simple way to build your complex virtual test lab | Nokia.com,” Nokia.com, 2024. https://www.nokia.com/blog/containerlab-a-free-and-simple-way-to-build-your-complex-virtual-test-lab/ (accessed Sep. 04, 2025).

[2]“Powering the new NetOps era with Containerlab | Nokia.com,” Nokia.com, 2022. https://www.nokia.com/blog/powering-the-new-netops-era-with-containerlab/ (accessed Sep. 04, 2025).

[3]K. Suo, Y. Zhao, W. Chen, and J. Rao, “An Analysis and Empirical Study of Container Networks,” IEEE Xplore, Apr. 01, 2018. https://ieeexplore.ieee.org/abstract/document/8485865/ (accessed Sep. 25, 2022).

[4]“Making containers work on-prem and on AWS with Cisco,” Cisco, Jan. 2025. https://www.cisco.com/site/us/en/learn/topics/computing/what-are-containers.html

[5]Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, “A Comparative Study of Containers and Virtual Machines in Big Data Environment,” arXiv (Cornell University), Jul. 2018, doi: https://doi.org/10.48550/arxiv.1807.01842.

[6]S. Deochake, S. Maheshwari, R. De, and A. Grover, “Comparative Study of Virtual Machines and Containers for DevOps Developers,” arXiv (Cornell University), Aug. 2018, doi: https://doi.org/10.48550/arxiv.1808.08192.

[7]P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and Virtual Machines at Scale,” Proceedings of the 17th International Middleware Conference, Nov. 2016, doi: https://doi.org/10.1145/2988336.2988337.

[8]“namespaces(7) - Linux manual page,” man7.org. https://man7.org/linux/man-pages/man7/namespaces.7.html

[9]D. Thomas, “Optimizing Data Center Resource Management: A Comparative Study of Virtual Machine and Container Orchestration Tools,” Jul. 2025, doi: https://doi.org/10.31224/4940.

[10]Docker, “What is a Container?,” Docker. https://www.docker.com/resources/what-container/

[11]IBM Cloud Team, “Containers Versus Virtual Machines (VMs): What’s The Difference? | IBM,” www.ibm.com, Apr. 25, 2024. https://www.ibm.com/think/topics/containers-vs-vms

[12]“cgroups(7) - Linux manual page,” man7.org. https://man7.org/linux/man-pages/man7/cgroups.7.html

[13]“Control Groups — The Linux Kernel documentation,” Kernel.org, 2025. https://www.kernel.org/doc/html/v5.9/admin-guide/cgroup-v1/cgroups.html (accessed Sep. 04, 2025).

[14]Liu Xingtao, Guo Yantao, W. Wei, Zhou Sanyou, and L. Jiliang, “Network virtualization by using software-defined networking controller based Docker,” 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, pp. 1112–1115, May 2016, doi: https://doi.org/10.1109/itnec.2016.7560537.

[15]V. P. Reynoso-Sánchez, Betzabet García-Mendoza, C. R. Jaimez-González, and W. A. Luna-Ramírez, “Converting WOX Objects to YAML Documents,” 2021 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 1565–1571, Dec. 2023, doi: https://doi.org/10.1109/csci62032.2023.00258.

[16]Docker, “What is Docker?,” Docker Documentation, 2024. https://docs.docker.com/get-started/docker-overview/

[17]IBM, “Docker,” Ibm.com, Jun. 06, 2024. https://www.ibm.com/think/topics/docker

[18]D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” 2014. Available: https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf

[19]“What is an image?,” Docker Documentation, 2024. https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/

[20]“Volumes,” Docker Documentation, 2024. https://docs.docker.com/engine/storage/volumes/

[21]“Bind mounts,” Docker Documentation, 2024. https://docs.docker.com/engine/storage/bind-mounts/

[22]“What is Docker Compose?,” Docker Documentation, 2024. https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-docker-compose/

[23]“docker,” Docker Documentation, Feb. 09, 2024. https://docs.docker.com/reference/cli/docker/

[24]“Network drivers,” Docker Documentation, 2024. https://docs.docker.com/engine/network/drivers/

[25]R. Dodin, “containerlab,” containerlab.dev. https://containerlab.dev/

[26]“FRRouting,” frrouting.org. https://frrouting.org/

[27]L. Rizzo, “netmap: a novel framework for fast packet I/O.” Accessed: Sep. 04, 2025. [Online]. Available: https://www.usenix.org/system/files/conference/atc12/atc12-final186.pdf

[28]Hasan Redzovic, Zoran Cica, and Aleksandra Smiljanic, “Traffic Modelling Challenges on 10 Gbit/s Links Using Netmap Platform,” 2022 30th Telecommunications Forum (TELFOR), pp. 1–4, Nov. 2018, doi: https://doi.org/10.1109/telfor.2018.8612011.

[29]“GitHub - wbitt/Network-MultiTool: Multi-arch multitool for container network troubleshooting. (Formerly Praqma/Network-Multitool),” GitHub, 2021. https://github.com/wbitt/Network-MultiTool (accessed Sep. 04, 2025).

[30]Git, “Git,” Git-scm.com, 2024. https://git-scm.com/

[31]“libpcap in Launchpad,” Launchpad, May 04, 2005. https://launchpad.net/libpcap (accessed Sep. 04, 2025).

[32]A. Jain, “Demystifying Development: A Guide to ‘build-essential’ in Ubuntu for Seamless Software Compilation,” Medium, Jan. 12, 2024. https://medium.com/@adwalkz/demystifying-development-a-guide-to-build-essential-in-ubuntu-for-seamless-software-compilation-b590b5a298bb

[33]“kmod package in Ubuntu,” Launchpad. https://launchpad.net/ubuntu/+source/kmod

[34]“kmod(8) - Linux manual page,” Man7.org, 2025. https://man7.org/linux/man-pages/man8/kmod.8.html (accessed Sep. 04, 2025).

[35]“networking:iproute2 [Wiki],” Linuxfoundation.org, 2023. https://wiki.linuxfoundation.org/networking/iproute2

[36]“iputils package in Ubuntu,” Launchpad. https://launchpad.net/ubuntu/+source/iputils

[37]“networking:net-tools [Wiki],” Linuxfoundation.org, 2021. https://wiki.linuxfoundation.org/networking/net-tools

[38]“ethtool(8) - Linux manual page,” Man7.org, 2019. https://man7.org/linux/man-pages/man8/ethtool.8.html (accessed Sep. 04, 2025).

[39]R. Chandran, “Tips and tricks for optimizing container builds,” Opensource.com, 2020. https://opensource.com/article/20/5/optimize-container-builds (accessed Sep. 04, 2025).

[40]Kang Xi, Yulei Liu, and H. J. Chao, “Enabling flow-based routing control in data center networks using Probe and ECMP,” 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Apr. 2011, doi: https://doi.org/10.1109/infcomw.2011.5928885.

[41]Junaid Israr, Mouhcine Guennoun, and H. T. Mouftah, “Credible BGP Extensions to BGP for Secure Networking,” pp. 212–216, Sep. 2009, doi: https://doi.org/10.1109/icsnc.2009.74.

[42]S. Bakkali, H. Benaboud, and M. Ben Mamoun, “Security problems in BGP: An overview,” IEEE Xplore, Apr. 01, 2013. https://ieeexplore.ieee.org/document/6595458

[43]Y. Rekhter, T. Li, and S. Hares, “A Border Gateway Protocol 4 (BGP-4),” Rfc-editor.org, 2006, doi: https://doi.org/10.17487/RFC4271.

[44]Wikipedia Contributors, “Path-vector routing protocol,” Wikipedia, Jul. 25, 2025. https://en.wikipedia.org/wiki/Path-vector_routing_protocol

[45]CISCO, “BGP Best Path Selection Algorithm,” Cisco, Jul. 11, 2023. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[46]IBM, “AIX,” Ibm.com, Aug. 27, 2024. https://www.ibm.com/docs/en/aix/7.1.0?topic=protocol-autonomous-systems

[47]B. Wang, J. Zhang, W. Chen, and Y. Guo, “A Fast Reroute Mechanism for RIP Protocol,” Pacific-Asia Conference on Circuits, Communications and Systems, pp. 74–77, May 2009, doi: https://doi.org/10.1109/paccs.2009.17.

[48]M. Munas and Kuruvikulam Chandrasekaran Arun, “Performance Evaluation of Distance Vector (RIP) and Link-State (OSPF) Routing Protocols,” Nov. 2023, doi: https://doi.org/10.1109/iciics59993.2023.10421146.

[49]T. Y. Khang and K. C. Arun, “Performance Evaluation of Wireless Routing Protocols: RIP vs OSPF,” 2021 14th International Conference on Developments in eSystems Engineering (DeSE), vol. 5, pp. 541–545, Dec. 2021, doi: https://doi.org/10.1109/dese54285.2021.9719366.

[50]M. Perez and A. Santiago, “Process of Migration of Routing Protocols in a LAN: RIP to OSPF,” pp. 1–5, Sep. 2006, doi: https://doi.org/10.1109/iceee.2006.251919.

[51]M. Athira, L. Abrahami, and R. G. Sangeetha, “Study on network performance of interior gateway protocols — RIP, EIGRP and OSPF,” IEEE Xplore, Mar. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8067958

[52]M. Jayakumar, N. Ramya Shanthi Rekha, and B. Bharathi, “A comparative study on RIP and OSPF protocols,” IEEE Xplore, Mar. 01, 2015. https://ieeexplore.ieee.org/document/7193275

[53]CISCO, “What Is Administrative Distance?,” Cisco, Aug. 2015. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/15986-admin-distance.html

[54]CISCO, “OSPF Neighbor States,” Cisco, Dec. 13, 2022. https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/13685-13.html

[55]“HPE Aruba Networking,” HPE Aruba Networking, 2019. https://arubanetworking.hpe.com/techdocs/AOS-S/16.10/MRG/WB/content/common (accessed Sep. 04, 2025).

[56]J. Moy, “OSPF Version 2,” OSPF version 2, Apr. 1998, doi: https://doi.org/10.17487/rfc2328.

[57]CISCO, “How Does Load Balancing Work?,” Cisco. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html

[58]Wikipedia Contributors, “Equal-cost multi-path routing,” Wikipedia, Aug. 29, 2024. https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing

[59]C. Hopps, “RFC 2992: Analysis of an Equal-Cost Multi-Path Algorithm,” IETF Datatracker, 2025. https://datatracker.ietf.org/doc/html/rfc2992 (accessed Sep. 04, 2025).

[60]CISCO, “ECMP Load Balancing.” Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mp_l3_vpns/configuration/xe-3s/asr903/17-1-1/b-mpls-l3-vpns-xe-17-1-asr900/b-mpls-l3-vpns-xe-17-1-asr900_chapter_0101.pdf

[61]“ECMP Load-Balancing Algorithms,” Paloaltonetworks.com, 2025. https://docs.paloaltonetworks.com/pan-os/11-0/pan-os-networking-admin/ecmp/ecmp-load-balancing-algorithms (accessed Sep. 04, 2025).

[62]IBM, “IBM i,” Ibm.com, Apr. 08, 2025. https://www.ibm.com/docs/en/i/7.6.0?topic=routing-open-shortest-path-first%20%20%20%20%0d (accessed Sep. 04, 2025).

[63]Shaoqiang Wang, DongSheng Xu, and ShiLiang Yan, “Analysis and application of Wireshark in TCP/IP protocol teaching,” IEEE Xplore, Apr. 01, 2010. https://ieeexplore.ieee.org/abstract/document/5496372

[64]P. Goyal and A. Goyal, “Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark,” IEEE Xplore, Sep. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8319360

[65]CISCO, “What Is an Ethernet Switch?,” Cisco, Jun. 2025. https://www.cisco.com/site/us/en/learn/topics/networking/what-is-an-ethernet-switch.html

[66]IBM, “z/VM,” Ibm.com, Jul. 18, 2025. https://www.ibm.com/docs/en/zvm/7.4.0?topic=options-virtual-switch

[67]IBM, “IBM Power10,” Ibm.com, Dec. 12, 2024. https://www.ibm.com/docs/en/power10?topic=pnc-virtual-network-bridges (accessed Sep. 04, 2025).

[68]“networking:bridge [Wiki],” Linuxfoundation.org, 2021. https://wiki.linuxfoundation.org/networking/bridge (accessed Sep. 04, 2025).

[69]CISCO, “Different Types of Network Switches,” Cisco, Mar. 2025. https://www.cisco.com/site/us/en/learn/topics/small-business/what-are-the-different-types-of-network-switches.html

[70]Z. Xiyang and C. Chuanqing, “Research on VLAN Technology in L3 Switch,” IEEE Xplore, Nov. 01, 2009. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5369283 (accessed Apr. 19, 2022).

[71]H.-M. Tseng, H.-L. Lee, J.-W. Hu, T.-L. Liu, J.-G. Chang, and W.-C. Huang, “Network Virtualization with Cloud Virtual Switch,” IEEE Xplore, Dec. 01, 2011. https://ieeexplore.ieee.org/abstract/document/6121394 (accessed Jun. 18, 2023).

[72]D. A. SHAJI. GEORGE and A. S. HOVAN. GEORGE, “A Brief Overview of VXLAN EVPN,” Zenodo (CERN European Organization for Nuclear Research), Jul. 2021, doi: https://doi.org/10.5281/zenodo.7027361.

[73]M. Elmadani and Salem Omar Sati, “Data Center Lab Using VxLAN Data Plane and BGP-EVPN Control Plane,” pp. 354–358, Oct. 2023, doi: https://doi.org/10.1109/icdabi60145.2023.10629438.

[74]Cristian Hernandez Benet, Kyoomars Alizadeh Noghani, A. Kassler, Ognjen Dobrijevic, and P. Jestin, “Policy-based routing and load balancing for EVPN-based data center interconnections,” Nov. 2017, doi: https://doi.org/10.1109/nfv-sdn.2017.8169841.

[75]T. Singh, V. Jain, and G. S. Babu, “VXLAN and EVPN for data center network transformation,” IEEE Xplore, Jul. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8203947 (accessed May 25, 2023).

[76]A.-E. Rădoi and C.-I. Rincu, “Integration of Data Center Network Technologies VxLAN, BGP, EVPN,” IEEE Xplore, Jun. 01, 2022. https://ieeexplore.ieee.org/abstract/document/9817218/ (accessed Jan. 14, 2023).

[77]Claudiu Trăistaru, “VXLAN - A practical approach to cloud computing scalability,” pp. 1–4, Sep. 2023, doi: https://doi.org/10.1109/roedunet60162.2023.10274934.

[78]Pravallika Vasireddy, R. K. Rangan, and V Kaviya, “Troubleshooting Tool for VxLAN Bridging,” 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), pp. 1–7, Oct. 2022, doi: https://doi.org/10.1109/gcat55367.2022.9972002.

[79]J. E. Vaca P. and G. D. Salazar-Chacon., “VXLAN-IPSec Dual-Overlay as a Security Technique in Virtualized Datacenter Environments,” 2020 IEEE ANDESCON, pp. 1–6, Oct. 2020, doi: https://doi.org/10.1109/andescon50619.2020.9272160.

[80]J. Kinoshita, K. Maeda, H. Yabusaki, K. Akune, and N. Komoda, “Realization of VXLAN Gateway-Based Data Center Network Virtualization,” IEEE Xplore, Jul. 01, 2016. https://ieeexplore.ieee.org/document/7557737 (accessed Sep. 20, 2023).

[81]CISCO, “Cisco Learning Network,” learningnetwork.cisco.com. https://learningnetwork.cisco.com/s/blogs/a0D3i000005YebJEAS/introduction-to-vxlan

[82]M. Mahalingam et al., “RFC 7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks,” IETF Datatracker, 2025. https://datatracker.ietf.org/doc/html/rfc7348%23page-10 (accessed Sep. 04, 2025).

[83]CISCO, “BGP EVPN VXLAN Overview The Evolution of BGP EVPN VXLAN.” Accessed: Sep. 04, 2025. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst9600/software/release/17-8/configuration_guide/vxlan/b_178_bgp_evpn_vxlan_9600_cg/bgp_evpn_vxlan_overview.pdf

[84]CISCO, “Configure VXLAN Flood and Learn with Multicast Core,” Cisco, Jun. 2018. https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/nx-os-software/200262-Configure-VxLAN-Flood-And-Learn-Using-Mu.html (accessed Sep. 04, 2025).

[85]J. Drake et al., “RFC 7432: BGP MPLS-Based Ethernet VPN,” IETF Datatracker, 2015. https://datatracker.ietf.org/doc/html/rfc7432 (accessed Sep. 04, 2025).

[86]R. Sharpe, E. Warnicke, and U. Lamping, “Wireshark user’s guide,” Wireshark.org, 2019. https://www.wireshark.org/docs/wsug_html_chunked/index.html

[87]G. S. Malkin, “RIP Version 2,” IETF, Nov. 01, 1998. https://datatracker.ietf.org/doc/html/rfc2453

[88]“Edgeshark,” Siemens.io, 2025. https://edgeshark.siemens.io/#/ (accessed Sep. 04, 2025).

[89]“siemens/edgeshark,” GitHub, Feb. 14, 2024. https://github.com/siemens/edgeshark

[90]“brctl(8) - Linux manual page,” Man7.org, 2020. https://man7.org/linux/man-pages/man8/brctl.8.html (accessed Sep. 04, 2025).

[91]“Linux on IBM Systems,” Ibm.com, Jul. 16, 2025. https://www.ibm.com/docs/en/linux-on-systems?topic=choices-using-linux-bridge (accessed Sep. 04, 2025).

[92]B. Pfaff et al., “The Design and Implementation of Open vSwitch The Design and Implementation of Open vSwitch,” 2015. Available: https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-pfaff.pdf

[93]“Getting Started — Open vSwitch 3.6.90 documentation,” Openvswitch.org, 2023. https://docs.openvswitch.org/en/latest/intro/ (accessed Sep. 04, 2025).

[94] Anna Koutsoni, “GitHub - annakts/Containerlab-Topologies: Network topologies implemented in Containerlab for academic research and experimentation.,” GitHub, 2025. https://github.com/annakts/Containerlab-Topologies.git (accessed Sep. 28, 2025).