

# Δικτυακός προγραμματισμός στον πυρήνα του λειτουργικού συστήματος (Kernel)

Καζακίδης Θεοχάρης

Διπλωματική Εργασία

Επιβλέπων: Παπαπέτρου Ευάγγελος

Ιωάννινα, Ιούλιος 2024



ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  
UNIVERSITY OF IOANNINA



# Ευχαριστίες

Καταρχάς θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου καθώς συντέλεσαν την μεγαλύτερη αρωγή κατά την διάρκεια των πέντε αυτών χρόνων.

Έπειτα θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Παπαπέτρου Ευάγγελο για την άψογη συνεργασία που είχαμε καθώς και για την στήριξη και την καθοδήγηση που μου προσέφερε απλόχερα στην διάρκεια της διπλωματικής εργασίας.

2024

Καζακίδης Θεοχάρης



# Περίληψη στα ελληνικά

Η παρούσα διπλωματική εργασία εστιάζει στην περιγραφή και στην ανάλυση του Netgraph, ενός υποσυστήματος που χρησιμοποιείται για να προσθέσει δικτυακές λειτουργίες στο λειτουργικό συστήματα του FreeBSD. Αρχικά, περιγράφεται ο σκοπός της διπλωματικής και παρουσιάζεται η ιστορική εξέλιξη και χρησιμότητα του Netgraph. Στο δεύτερο κεφάλαιο, αναλύεται η αρχιτεκτονική του Netgraph, οι βασικές του δομές, όπως οι κόμβοι, τα άγκιστρα αλλά και άλλες δομές του Netgraph. Έπειτα γίνεται εμβάθυνση στην λειτουργία των δομών μέσω της τεχνικής ανάλυσης τους. Επιπρόσθετα, δίνονται παραδείγματα διασυνδέσεων μεταξύ των δομών. Το τρίτο κεφάλαιο ασχολείται με την εγκατάσταση και την πρακτική χρήση του Netgraph, παρουσιάζοντας τόσο τις διαδικασίες εγκατάστασης όσο και την εφαρμογή των σχετικών βοηθητικών προγραμμάτων όπως το ngctl. Το τέταρτο κεφάλαιο είναι αφιερωμένο στη δημιουργία ενός κόμβου στο Netgraph, εξηγώντας λεπτομερώς τη διασύνδεση, την κυκλοφορία και τον χειρισμό των πακέτων δεδομένων και μηνυμάτων ελέγχου, καθώς και τις επεμβάσεις στα επίπεδα της στοίβας πρωτοκόλλων TCP/IP. Το πέμπτο και τελευταίο κεφάλαιο αναφέρεται σε ένα παράδειγμα δημιουργίας προσαρμοσμένου κόμβου και συγκεκριμένα στη δημιουργία του κόμβου ng\_triple\_tee, παρουσιάζοντας την ιδέα και την υλοποίηση του.

**Λέξεις Κλειδιά:** Netgraph, FreeBSD, ngctl, ng\_triple\_tee

# Abstract

This thesis focuses on the description and analysis of Netgraph, a subsystem used to add network functionalities to the FreeBSD operating system. Initially, the purpose of the thesis is described, presenting the historical development and usability of Netgraph. The second chapter explains the architecture of Netgraph, its fundamental structures such as nodes, hooks, and other Netgraph components, followed by an in-depth technical analysis of these structures. Examples of connections between the structures are also provided. The third chapter addresses the installation and practical use of Netgraph, presenting both the installation processes and the application of utilities programs like ngctl. The fourth chapter is dedicated to the creation of a node in Netgraph, detailing the interconnection, the flow and handling of data packets and control messages, as well as interventions at the TCP/IP protocol stack levels. The fifth and final chapter refers to an example of custom node creation, specifically the development of the `ng_triple_tee` node, presenting its concept and implementation.

**Keywords:** Netgraph, FreeBSD, ngctl, `ng_triple_tee`

# Πίνακας περιεχομένων

<b>Κεφάλαιο 1. Εισαγωγή.....</b>	<b>1</b>
1.1    Δικτυακές λειτουργίες εσωτερικά στο πυρήνα.....	1
1.2    Τι είναι το Netgraph?.....	1
1.2.1 Ιστορική αναδρομή.....	3
1.3    Χρησιμότητα του Netgraph.....	3
1.4    Αντικείμενο της διπλωματικής .....	5
<b>Κεφάλαιο 2. Αρχιτεκτονική του Netgraph .....</b>	<b>6</b>
2.1    Βασική περιγραφή της αρχιτεκτονικής .....	6
2.2    Ανάλυση των δομών της αρχιτεκτονικής.....	8
2.2.1    Κόμβος.....	8
2.2.2    Άγκιστρο.....	10
2.2.3    Πακέτα δεδομένων .....	11
2.2.4    Μηνύματα ελέγχου.....	11
2.2.5    Μεταδεδομένα.....	12
2.2.6    Διευθυνσιοδότηση.....	12
2.2.7    Τύποι κόμβων.....	13
2.2.7.1    Διασυνδέσεις, συσκευές και υποδοχείς δικτύου.....	16
2.2.7.2    Πρωτόκολλα δικτύου.....	17
2.2.7.3    Μεταγωγή.....	19
2.2.7.4    Έλεγχος και διόρθωση.....	20
2.2.7.5    Λοιπά.....	21
2.3    Παραδείγματα διασυνδέσεων δομών .....	23
2.4    Παρόμοια εργαλεία διαχείρισης του υποσυστήματος δικτύωσης.....	26
<b>Κεφάλαιο 3. Εγκατάσταση και χρήση του Netgraph.....</b>	<b>27</b>
3.1    Εγκατάσταση του Netgraph.....	27
3.1.1    Δυναμική εγκατάσταση.....	28
3.1.2    Στατική εγκατάσταση .....	29

3.2	Χρήση του Netgraph.....	31
3.2.1	Εντολή nghook.....	31
3.2.2	Εντολή ngctl.....	34
3.3	Παραδείγματα χρήσης του Netgraph.....	40
<b>Κεφάλαιο 4.</b>	<b>Δημιουργία κόμβου στο Netgraph .....</b>	<b>46</b>
4.1	Περιγραφή δομής κόμβου .....	46
4.2	Διασύνδεση με σύστημα Netgraph.....	48
4.3	Εγκατάσταση και χρήση νέου κόμβου .....	47
4.4	Αναχαιτηση ροης δεδομένων στα επίπεδα της στοίβας πρωτοκόλλων TCP/IP.....	56
4.4.1	Αναχαιτηση ροης δεδομένων στο επίπεδο συνδέσμου.....	57
4.4.2	Αναχαιτηση ροης δεδομένων στο επίπεδο δικτύου.....	58
4.4.3	Αναχαιτηση ροης δεδομένων στο επίπεδο εφαρμογής.....	59
4.5	Κυκλοφορία και χειρισμός πακέτου ή μηνύματος ελέγχου στο Netgraph.....	61
4.5.1	Χειρισμός πακέτου δεδομένων.....	61
4.5.1.1	Λήψη και ανάγνωση του πακέτου δεδομένων.....	61
4.5.1.3	Τροποποίηση του πακέτου δεδομένων.....	62
4.5.1.4	Δημιουργία του πακέτου δεδομένων .....	64
4.5.1.5	Αποστολή και αποδέσμευση του πακέτου δεδομένων.....	64
4.5.2	Χειρισμός μηνύματος ελέγχου.....	67
4.5.2.1	Λήψη και ανάγνωση του μηνύματος ελέγχου.....	67
4.5.2.2	Τροποποίηση του μηνύματος ελέγχου.....	68
4.5.2.3	Δημιουργία του μηνύματος ελέγχου.....	68
4.5.2.4	Αποστολή και αποδέσμευση του μηνύματος ελέγχου.....	69
<b>Κεφάλαιο 5.</b>	<b>Δημιουργία κόμβου ng_triple_tee .....</b>	<b>73</b>
5.1	Iδέα του κόμβου ng_triple_tee.....	73
5.2	Υλοποίηση του κόμβου ng_triple_tee .....	75
<b>Κεφάλαιο 6.</b>	<b>Βιβλιογραφία και αναφορές.....</b>	<b>94</b>
Παραρτήματα.....	102	
Παράρτημα 1. Επιπλέον παραδείγματα χρήσης του ngctl .....	102	

Παράρτημα 2.	Συναρτήσεις, Μέθοδοι και Κεφαλίδες.....	ix
Παράρτημα 3.	Επεξήγηση σημαντικών τύπων κόμβων.....	120



# Κεφάλαιο 1. Εισαγωγή

## 1.1 Δικτυακές λειτουργίες εσωτερικά στον πυρήνα

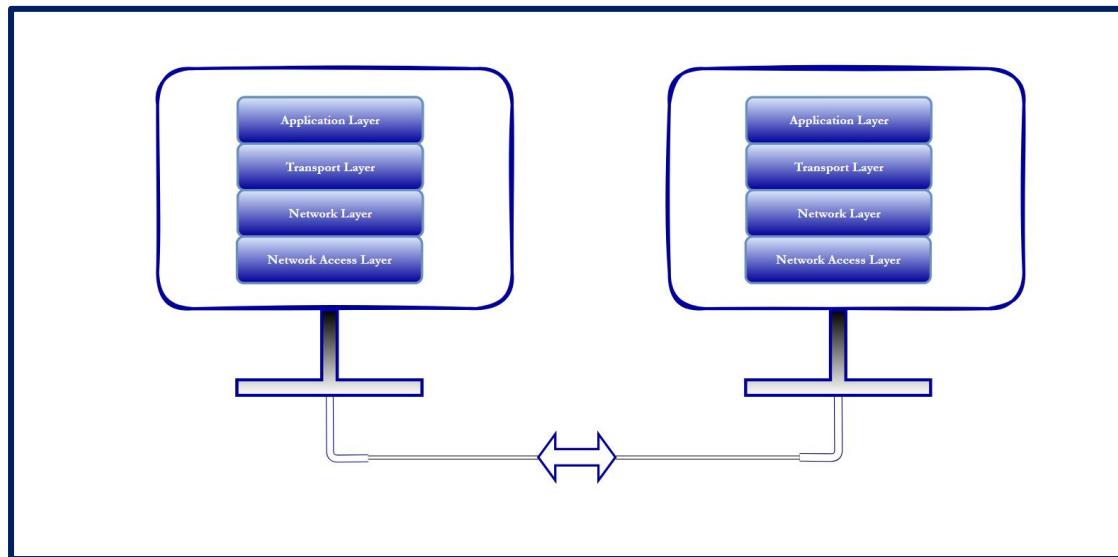
Τα υποσυστήματα δικτύωσης στα περισσότερα λειτουργικά συστήματα είναι μονολιθικά. Επομένως ο χρήστης δεν έχει άμεση πρόσβαση για να παραμετροποιήσει το υποσύστημα δικτύωσης. Ο μόνος τρόπος για να έχει πρόσβαση στο υποσύστημα δικτύωσης είναι με τη χρήση διεπαφών υψηλότερου επιπέδου, όπως τα BSD sockets. Όμως με την χρήση αυτών των διεπαφών ο χρήστης έχει περιορισμένη αλληλεπίδραση με το δίκτυο, καθώς δεν επιτρέπουν την αλλαγή της εσωτερικής λειτουργίας των πρωτοκόλλων δικτύωσης. Το Netgraph προσφέρει μια λύση σε αυτόν τον περιορισμό, αφού επιτρέπει στον χρήστη να επεμβαίνει απευθείας στο υποσύστημα δικτύωσης του λειτουργικού συστήματος FreeBSD. Μέσω του Netgraph, οι χρήστες μπορούν να διαχειρίζονται δυναμικά κόμβους οι οποίοι αντιπροσωπεύουν διάφορα πρωτόκολλα. Αυτό επιτρέπει την τροποποίηση και επεξεργασία των δικτυακών πρωτοκόλλων σε χαμηλό επίπεδο.

## 1.2 Τι είναι το Netgraph?

Αν χρησιμοποιούσαμε μία λέξη για να χαρακτηρίσουμε το Netgraph, αυτή δεν θα ήταν άλλη από τη λέξη “παραμετροποιήσιμο”, αφού ο χρήστης μπορεί να χρησιμοποιήσει ήδη υπάρχοντα πρωτόκολλα και μέσω κατάλληλων συνδέσεων να φτιάχει ένα «νέο» πρωτόκολλο κομμένο και ραμμένο στις απαιτήσεις του. Αυτό επιτυγχάνεται μέσω της παρεμβολής του υποσυστήματος Netgraph μεταξύ των επιπέδων δικτύου και συνδέσμου-μετάδοσης της αρχιτεκτονικής δικτύου TCP/IP[1]. Στην πραγματικότητα, το Netgraph είναι ένα αρθρωτό υποσύστημα το οποίο «ζει» μέσα στο πυρήνα (kernel) του συστήματος FreeBSD και «προσθέτει» δικτυακές λειτουργίες στις ήδη υπάρχων. Αυτό οφείλεται στην συνεργασία οντοτήτων όπως οι κόμβοι (nodes) και τα άγκιστρα (hooks).

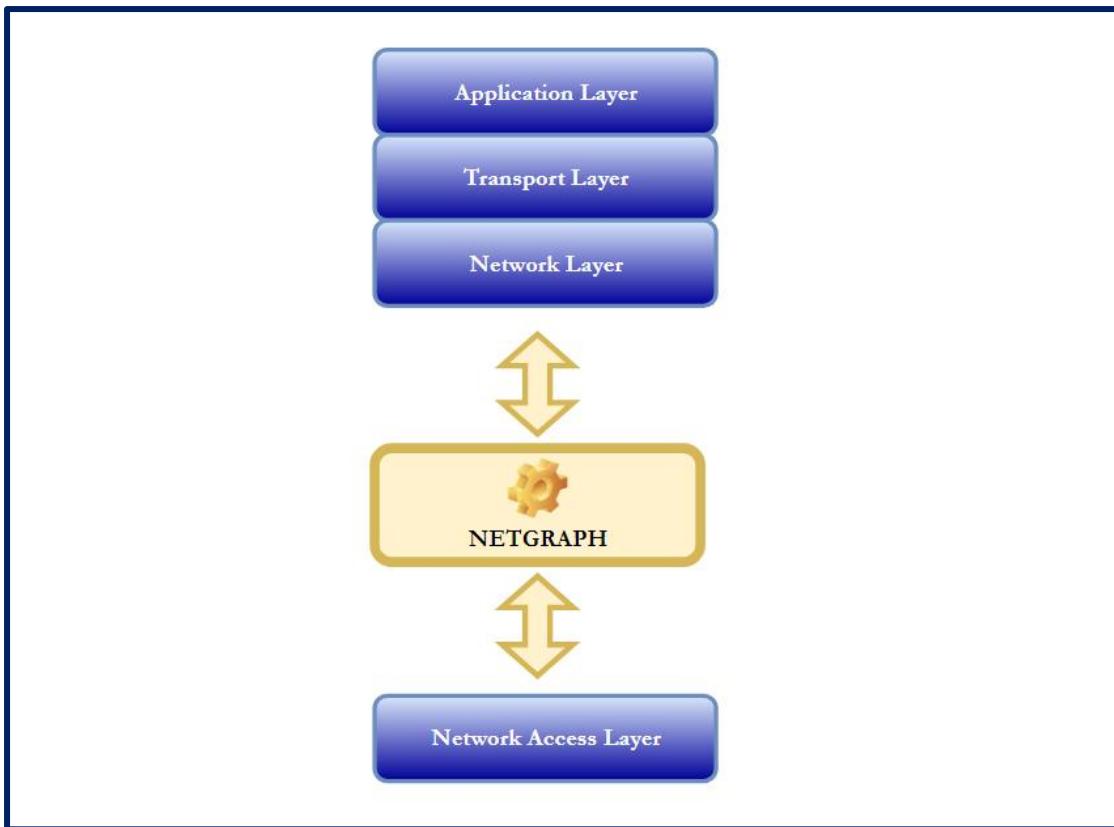
Μέσα σε έναν κόμβο, τα πακέτα δεδομένων υποβάλλονται σε επεξεργασία, τροποποιούνται ή απλώς μεταφέρονται μέσω των συνδεδεμένων αγκίστρων σε άλλους κόμβους.

Παρακάτω απεικονίζονται δύο συνδεδεμένοι υπολογιστές που ακολουθούν την αρχιτεκτονική TCP/IP .



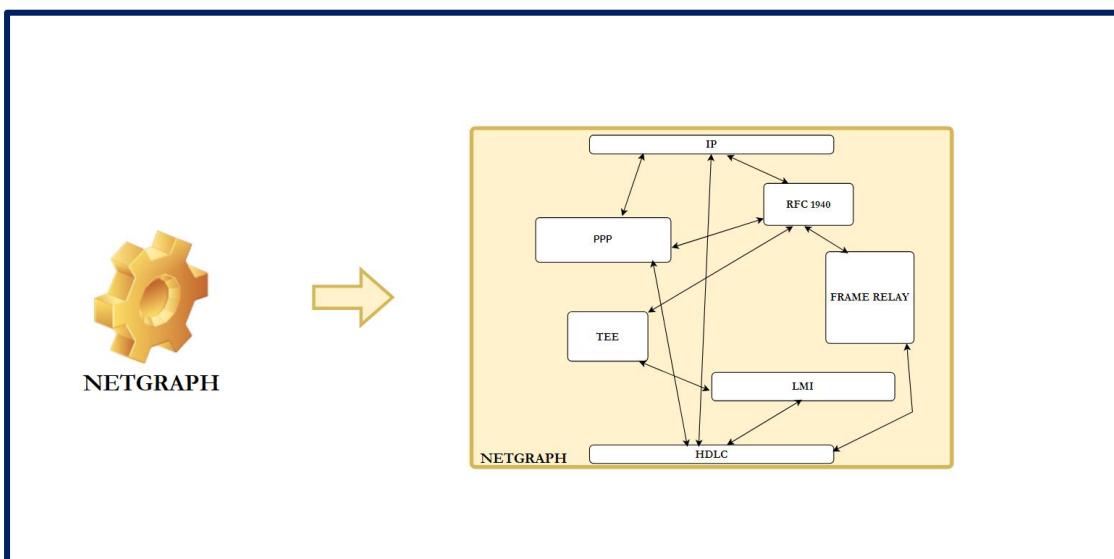
Εικόνα 1.1. Επικοινωνία Η/Υ με μοντέλο TCP/IP

Το Netgraph ενσωματώνεται ουσιαστικά μεταξύ του επιπέδου δικτύου και του επιπέδου συνδέσμου ή πρόσβασης.



Εικόνα 1.2. Αναπαράσταση θέσης του επιπέδου Netgraph στο TCP/IP

Άμα γίνει μεγέθυνση στο «πλαίσιο» Netgraph(κίτρινο πλαίσιο της παραπάνω εικόνας) θα φανεί στο παράδειγμα ότι στο πάνω μέρος συνδέεται με το IP και στο κάτω μέρος με το HDLC. Εσωτερικά του κουτιού προστίθενται επιπλέον πρωτόκολλα που ξεχωριστά το καθένα επιτελεί μια λειτουργία. Έτσι μεταξύ τους φτιάχνουν ένα «custom» πρωτόκολλο όπου η πληροφορία μεταφέρεται από το IP προς τον σύνδεσμο και το αντίστροφο[2].



Εικόνα 1.3. Μεγέθυνση του επιπέδου Netgraph

### **1.2.1 Ιστορική αναδρομή**

Το υποσύστημα Netgraph αναπτύχθηκε το έτος 1996 από τους Julian Elischer και Archie Cobbs για την εταιρεία Whistle InterJet[2-3]. Η αφορμή για την υλοποίηση του Netgraph ήταν οι πολύπλοκες απαιτήσεις ενός TCP/IP δρομολογητή που λειτουργούσε με βάση το FreeBSD και ταυτόχρονα έπρεπε να υποστηρίζει διάφορα πρωτόκολλα και μεθόδους μετάδοσης IP πακέτων μέσω bit-synchronous σειριακών συνδέσεων WAN. Έτσι απαιτούνταν μια λύση με σκοπό την υποστήριξη διαφορετικών και πολύπλοκων πρωτοκόλλων δικτύωσης και μεθόδων ενθυλάκωσης σε ένα σύγχρονο περιβάλλον υψηλής ταχύτητας, όπως οι T1/E1 γραμμές. Το Netgraph δημιουργήθηκε για να ανταποκριθεί σε αυτήν την ανάγκη, παρέχοντας μια ευέλικτη και επεκτάσιμη πλατφόρμα για τη διαχείριση και υποστήριξη διάφορων πρωτοκόλλων και μεθόδων δικτύωσης. Αξίζει να σημειωθεί ότι το Netgraph, στην πρώτη του έκδοση (FreeBSD version 2.2) είχε μόλις 10 μονάδες (modules). Στην έκδοση που μελετάμε (FreeBSD version 14.0), το Netgraph αποτελείται πλέον από 58 μονάδες (modules). Αυτό μπορούμε να το δούμε με την εντολή `grep ng_ | wc -l`.

## **1.3 Χρησιμότητα του Netgraph**

Ο στόχος του Netgraph είναι να συμπληρώσει, και όχι να αντικαταστήσει, την ήδη υπάρχουσα δομή δικτύωσης του πυρήνα(kernel). Με την παρουσία του, δίνει τη δυνατότητα στους μηχανικούς να επωφεληθούν από μια πληθώρα πλεονεκτημάτων[4], ενισχύοντας σημαντικά τις δυνατότητες δικτύωσης και επικοινωνίας του συστήματος. Μερικά από αυτά παρουσιάζονται παρακάτω.

**Ταχύτητα:** Οι διαδικασίες εκτελούνται εντός του ίδιου του πυρήνα του συστήματος, επιτυγχάνοντας έτσι ελάχιστη καθυστέρηση στη μετάδοση δεδομένων μεταξύ των κόμβων.

**Προηγμένη Διαχείριση Δεδομένων:** Προσφέρει τη δυνατότητα ταυτόχρονης διαχείρισης πολλαπλών πρωτοκόλλων, επιτρέποντας την ομαλή μεταφορά πληροφορίας από και προς το επίπεδο IP. Η μετακίνηση δεδομένων μεταξύ των κόμβων μπορεί να γίνεται

άμεσα ή να οργανωθεί σε ουρά και συνδυαστικά με τις μικρές καθυστερήσεις, καθιστά το Netgraph ιδανικό σε περιπτώσεις όπου ο μικρός χρόνος είναι σημαντικός παράγοντας.

**Ευελιξία:** Επιτρέπει τη σύνδεση πολλαπλών κόμβων με έναν κόμβο και την αυθαίρετη διάταξή τους, προσφέροντας στους χρήστες τη δυνατότητα να διαμορφώσουν κόμβους χωρίς την ανάγκη ειδικών βοηθητικών προγραμμάτων(utilities) για κάθε τύπο κόμβου.

**Χρηστικότητα:** Υποστηρίζει τη φόρτωση τύπων κόμβων κατά την εκτέλεση, προσαρμόζοντας τις ανάγκες του δικτύου.

**Βελτιωμένη Διαμόρφωση Δικτύου:** Επιτρέπει τη διαμόρφωση οποιουδήποτε κόμβου και την παραγωγή αιτημάτων διαμόρφωσης από έναν κόμβο προς έναν άλλο, απλοποιώντας τη διαδικασία διατήρησης και ενημέρωσης των διαμορφώσεων δικτύου.

## 1.4 Αντικείμενο της διπλωματικής

Έχει παρατηρηθεί ότι δεν υπάρχει ένας αναλυτικός οδηγός για το Netgraph, αλλά αντίθετα, οι πληροφορίες είναι διάσπαρτες. Ο στόχος μας είναι να συγκεντρώσουμε πληροφορίες για το Netgraph και να παρέχουμε έναν αναλυτικό οδηγό χρήσης. Συγκεκριμένα, θα αναλυθεί η λειτουργία και οι δυνατότητες του Netgraph. Επίσης, θα πραγματοποιηθεί επεξήγηση των βασικών δομών και του τρόπου συνεργασίας αυτών των δομών μεταξύ τους. Θα δοθούν οδηγίες για το πώς μπορεί κάποιος να χρησιμοποιήσει το Netgraph, ποια στοιχεία πρέπει να ληφθούν υπόψη για τη δημιουργία ενός δικού του κόμβου και τέλος, θα παρουσιαστεί ένα παράδειγμα λειτουργικού προσαρμοσμένου κόμβου. Κατ' αυτόν τον τρόπο, οι αναγνώστες θα είναι σε θέση να αποκτήσουν γνώση σχετικά με τη λειτουργία του Netgraph και αυξάνοντας το επίπεδο της κατανόησής τους, να φτάσουν στο σημείο να δημιουργούν το δικό τους κόμβο.

# **Κεφάλαιο 2.**

# **Αρχιτεκτονική του Netgraph**

## **2.1 Βασική περιγραφή της αρχιτεκτονικής**

Για την επίτευξη των λειτουργιών του, το Netgraph χρησιμοποιεί κάποιες βασικές δομές, από τις οποίες οι πιο σημαντικές είναι οι κόμβοι (nodes) και τα άγκιστρα (hooks). Εξίσου σημαντικές είναι και οι δομές που διευκολύνουν την ανταλλαγή μηνυμάτων, καθώς και εκείνες που αφορούν την πακετοποίηση της πληροφορίας, την διευθυνσιοδότηση των κόμβων και γενικότερα οι δομές που θα επεξηγηθούνε παρακάτω[2-5].

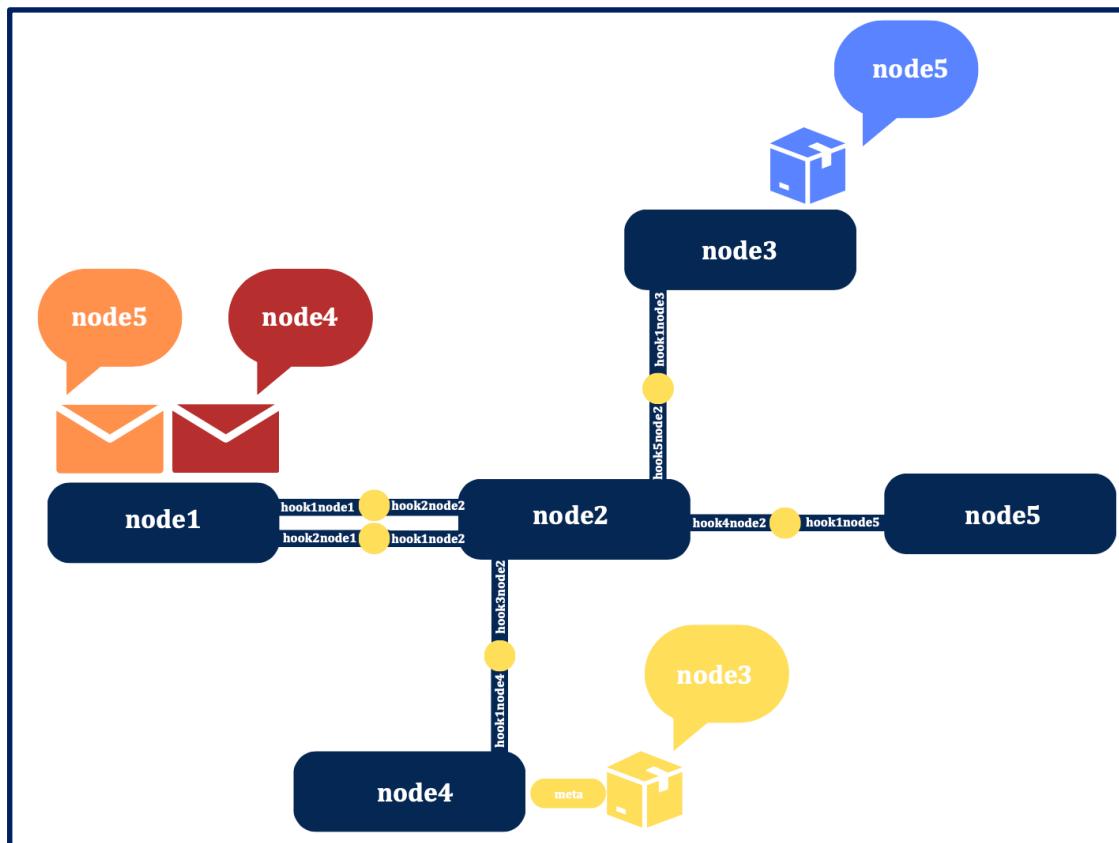
Πίνακας 2.1. Στοιχεία της αρχιτεκτονικής του Netgraph

Κόμβος	Άγκιστρα	Πακέτα δεδομένων	Μηνύματα ελέγχου	Σύστημα διευθυνσιοδότηση	Μεταδεδομένα

Κάθε δομή της αρχιτεκτονικής του Netgraph υλοποιεί ανεξάρτητα μια λειτουργία και όταν συνεργάζονται όλες οι δομές μαζί υλοποιούν ένα δίκτυο. Συγκεκριμένα ο κόμβος είναι η δομή που αναπαριστά συνήθως ένα δικτυακό πρωτόκολλο και εσωτερικά σε αυτόν υλοποιούνται λειτουργίες για την διαχείριση των διερχομένων και εξερχόμενων

πακέτων. Η διέλευση των πακέτων και η σύνδεση του ενός κόμβου με τον άλλο γίνεται μέσω των αγκίστρων. Τα πακέτα δεδομένων περιέχουν πληροφορίες και τις μεταφέρουν από κόμβο σε κόμβο. Παρόμοια λειτουργία έχουν και τα μηνύματα ελέγχου, απλώς η διαφορά τους είναι ότι αυτά μεταφέρουν τα μηνύματα ενός κόμβου σε ένα άλλον. Άμα έχει σταλθεί το πακέτο δεδομένων ή των μήνυμα ελέγχου και χρειαστούν επιπλέον πληροφορίες σχετικά με αυτό, τότε ο κόμβος στέλνει τα μεταδεδομένα που θα βοηθήσουν στην διαχείρισή του. Τελευταίο αλλά εξίσου σημαντικό είναι το σύστημα διευθυνσιοδότησης όπου χάρη σε αυτό ο κάθε κόμβος μπορεί να επιλέξει σε ποιον κόμβο ή άγκιστρο να στείλει τα δεδομένα.

Παρακάτω παρουσιάζεται ενδεικτικά μια εικόνα για το πώς συνεργάζονται οι δομές που αναφέρθηκαν



Εικόνα 2.1. Αναπαράσταση συνεργασίας δομών του Netgraph μεταξύ τους

## 2.2 Ανάλυση των δομών της αρχιτεκτονικής

Στην παραπάνω ενότητα αναφέρθηκαν με μια πιο γενική έννοια οι δομές που απαρτίζουν το υποσύστημα Netgraph, ώστε να γίνει μια αρχική κατανόηση της κάθε δομής. Παρακάτω θα γίνει εμβάθυνση τόσο στην λειτουργία όσο και σε τεχνικά χαρακτηριστικά κάθε δομής αλλά και στον τρόπο που μπορούν όλες αυτές οι δομές να συνεργαστούν μεταξύ τους[2-5].

### 2.2.1 Κόμβος

Η πιο θεμελιώδης οντότητα στο Netgraph είναι ο κόμβος, ο οποίος βρίσκεται στον πυρήνα του συστήματος και είναι υπεύθυνος να υλοποιεί εσωτερικά ορισμένες προκαθορισμένες μεθόδους. Χάρη σε αυτές τις μεθόδους ο κόμβος επιτυγχάνει την ομαλή αλληλεπίδρασή του με τους υπόλοιπους κόμβους του δικτύου. Κατά την δημιουργία ενός κόμβου ορίζεται και ο τύπος του(type). Ο τύπος του κόμβου ουσιαστικά είναι μια μοναδική λέξη σε ASCII κώδικα, που καθορίζει την λειτουργία που επιτελεί ο κόμβος. Για παράδειγμα ο κόμβος με τύπο tee σημαίνει ότι δημιουργείται ένας κόμβος που θα επιτελεί λειτουργίες tee κόμβου. Σε προγραμματιστικό επίπεδο οι τύποι των κόμβων αποτελούν τις κλάσεις ενώ οι ίδιοι οι κόμβοι λειτουργούν ως περιπτώσεις (instances) των αντίστοιχων κλάσεων τους. Να επισημανθεί ότι όλοι οι τύποι κόμβων είναι υποκλάσεις του γενικού(generic) τύπου κόμβων και έτσι έχουν κληρονομήσει μερικές λειτουργίες, δηλαδή πιο απλοϊκά είναι σαν κάθε κόμβος που δημιουργείτε να έχει μια βάση που δίνετε η δυνατότητα να χτίσει ο χρήστης πάνω σε αυτήν. Από εκεί και πέρα κάθε κόμβος ανάλογα τα επίπεδο λειτουργικότητας που θέλει να στοχεύσει προσθέτει και τροποποιεί λειτουργίες πάνω σε αυτήν την βάση. Τέλος για την διαφοροποίησή των κοινών τύπων κόμβων μεταξύ τους αλλά και για την διευθυνσιοδότηση τους που θα αναφερθούμε παρακάτω, κάθε κόμβος έχει έναν μοναδικό αριθμό ID. Ο αριθμός αυτός εκφράζεται ως δεκαεξαδική τιμή 32 bit και μπορεί να χρησιμοποιηθεί για την αναφορά στο κόμβο όταν δεν του έχει εκχωρηθεί όνομα ASCII. Στους κόμβους μπορεί να εκχωρηθεί ένα παγκόσμιο(global) μοναδικό όνομα ASCII, το οποίο μπορεί να χρησιμοποιηθεί για την αναφορά στον κόμβο(Το όνομα δεν πρέπει να περιέχει τους χαρακτήρες “.” ή “:”).

To Netgraph διαθέτει ως δυνατό στοιχείο του την ταχύτητα. Για να συντελεστεί αυτό συμβάλουν πολλά στοιχεία και πολλές λειτουργίες, μια εκ των οποίων είναι τεχνική της πολλαπλής κλήσης συναρτήσεων στο εσωτερικό των κόμβων. Για παράδειγμα, τα πακέτα δεδομένων και μηνύματα ελέγχου παραδίδονται με κλήσεις συναρτήσεων και όχι με ουρές και γραμματοκιβώτια. Συγκεκριμένα στο εσωτερικό των κόμβων υπάρχουν διάφοροι μέθοδοι που καθορίζουν την λειτουργικότητα του κόμβου. Εσωτερικά αυτών των μεθόδων γίνεται η κλήση συναρτήσεων. Αν ο κόμβος tee επιθυμεί να στείλει ένα mbuf στον γειτονικό κόμβο hole, καλεί την συνάρτηση παράδοσης δεδομένων του δικτύου. Η συνάρτηση αυτή με τη σειρά της εντοπίζει τον κόμβο hole και καλεί τη μέθοδο λήψη δεδομένων του hole. Εκτός από την κλήση συναρτήσεων μέσα στις μεθόδους γίνεται και η επεξεργασία των δεδομένων άμα χρειαστεί.

Υπάρχουν μέθοδοι που κάθε κόμβος πρέπει να υλοποιεί, όπως και συναρτήσεις μέσα σε αυτές τις μεθόδους και βρίσκονται όλες στο Παράρτημα 2. Οι πιο σημαντικές μέθοδοι που υλοποιούνται μέσα στους κόμβους είναι οι παρακάτω

Πίνακας 2.2. Οι πιο σημαντικές μέθοδοι των κόμβων	
int constructor(node_p *node);	Δημιουργία νέου κόμβου
int newhook(node_p node, hook_p hook, const char *name);	Δημιουργία νέου αγκίστρου
int rcvmsg(node_p node, struct ng_mesg *msg,const char *retaddr, struct ng_mesg **resp);	Λήψη και χειρισμός ενός μηνύματος ελέγχου
int connect(hook_p hook)	Επιτυχής σύνδεση δύο αγκίστρων
int shutdown(node_p node);	Τερματισμός λειτουργίας ενός κόμβου

Πάραντα υπάρχουν εξαιρέσεις, δεν είναι εφικτή σε όλες τις περιπτώσεις η κλήση των συναρτήσεων και για αυτό κάθε κόμβος διαθέτει μια ουρά εισόδου που διαχειρίζεται τη ροή των δεδομένων που εισέρχονται σε αυτόν. Οι λειτουργίες που τροποποιούν την κατάσταση του κόμβου ονομάζονται εγγραφείς(writers) και είναι σημαντικό η μια λειτουργία να μην επηρεάζει την άλλη λειτουργία και κατά συνέπεια την κατάσταση του κόμβου. Για να αποφευχθεί αυτό η ουρά εισόδου εφαρμόζει την τεχνική αναγνώστη/συγγραφέα κάνοντας αμοιβαίο αποκλεισμό ώστε να συνεχιστεί η ομαλή λειτουργία του κόμβου. Προκαθορισμένα οι κόμβοι ορίζουν όλα τα αιτήματα λειτουργίας

ως εγγραφείς . Ακόμη έχουν την δυνατότητα να καθορίζουν ότι τα αιτήματα που προέρχονται από συγκεκριμένα άγκιστρα να θεωρούνται ως εγγραφείς. Επιπλέον τα δεδομένα που μεταφέρονται μέσω συγκεκριμένων αγκίστρων να μπορούν να τίθενται στην ουρά αντί να παραδίδονται απευθείας για να υπάρχει καλύτερη διαχείριση όταν αυτή απαιτείτε. Τέλος κάθε κόμβος θεωρεί προκαθορισμένα ότι όλα τα μηνυμάτων ελέγχου θεωρούνται ως εγγραφείς εκτός εάν έχουν καθοριστεί ρητά ως αναγνώστες(readers) στον ορισμό τους, όπου αναγνώστες είναι οι λειτουργίες που κάνουν διαχείριση δεδομένων.

Ένας κόμβος μπορεί να έχει κρυφή αλληλεπίδραση με άλλα στοιχεία του πυρήνα εκτός του υποσυστήματος Netgraph, όπως το υλικό της συσκευής, οι στοίβες πρωτοκόλλων του πυρήνα κλπ. Με αυτό επιτυγχάνει την συνεργασία διαφορετικών στοιχείων κάνοντας πιο γρήγορες τις διαδικασίες.

## 2.2.2 Άγκιστρα

Όσο χρήσιμος και αν είναι ένας κόμβος, «πνοή» στον γράφο δίνει το άγκιστρο(hook), καθώς μέσω αυτού γίνεται η μεταφορά δεδομένων εισόδου και εξόδου από κόμβο σε κόμβο. Τα άγκιστρα λειτουργούν ως σημεία σύνδεσης των κόμβων για την επικοινωνία μεταξύ τους, αποτελώντας ουσιαστικά μια «αμφίδρομη ακμή» μεταξύ των κόμβων. Κάθε άγκιστρο έχει σχεδιαστεί για να μεταδίδει ή να λαμβάνει δεδομένα από και προς άλλους κόμβους, επιτυγχάνοντας έτσι τη ροή δεδομένων. Συγκεκριμένα για να συνδεθούν δύο κόμβοι μεταξύ τους απαιτείται ένα άγκιστρο από τον καθένα κόμβο ώστε να δημιουργηθεί ένα ζεύγος(pair) αγκίστρων και να ξεκινήσει μετέπειτα η επικοινωνία τους. Γενικότερα ένα κόμβος μπορεί να έχει όσα άγκιστρα χρειάζεται και μπορεί να αποδίδει όποια σημασία θέλει σε ένα άγκιστρο. Όμως ένας κόμβος με συγκεκριμένο τύπο(type) έχει προκαθορισμένο αριθμό αγκίστρων, με συγκεκριμένη λειτουργία και όνομα. Υπάρχουν λεπτομέρειες που χρήζουν προσοχή σχετικά με τα άγκιστρα όπως ότι ένα άγκιστρο έχει ένα όνομα ASCII(Το όνομα δεν πρέπει να περιέχει τους χαρακτήρες “.” ή “：“), το οποίο είναι μοναδικό μεταξύ όλων των αγκίστρων του ίδιου κόμβου. Επιπλέον ένα άγκιστρο συνδέεται μόνο με κάποιο άλλο άγκιστρο και έτσι δημιουργείται σύνδεση. Στην περίπτωση που αποσυνδεθεί ή καταστραφεί το ένα άγκιστρο τότε καταστρέφεται και το άλλο άγκιστρο που ήταν συνδεδεμένο μαζί του. Επιπλέον ένα άγκιστρο μπορεί να βρεθεί σε μια κατάσταση(state) όπου τα εισερχόμενα πακέτα να βρίσκονται πάντα σε

ουρά αναμονής από το σύστημα ουράς εισόδου, αντί να παραδίδονται απευθείας. Αυτό μπορεί να χρησιμοποιηθεί όταν τα δεδομένα αποστέλλονται από έναν χειριστή διακοπών και η επεξεργασία πρέπει να είναι γρήγορη, ώστε να μην μπλοκάρει άλλες διακοπές. Ακόμη μπορεί να παρέχει συναρτήσεις προτεραιότητας τόσο λήψης δεδομένων όσο και λήψης μηνυμάτων. Αυτές θα πρέπει να χρησιμοποιούνται για δεδομένα και μηνύματα που λαμβάνονται μέσω του συγκεκριμένου άγκιστρου, αντί των γενικών μεθόδων σε όλο τον κόμβο. Ένας κόμβος μπορεί να αποφασίσει να αποδώσει ειδική σημασία σε ορισμένα άγκιστρα. Για παράδειγμα με τη ονοματοδοσία ενός αγκίστρου, δηλαδή με το όνομα debug ο κόμβος θα αρχίσει να στέλνει πληροφορίες εντοπισμού σφαλμάτων σε αυτό το άγκιστρο, μειώνοντας έτσι την υλοποίηση και άλλων μεθόδων μέσα στο κόμβο.

Έτσι τα άγκιστρα μεταφέρουν δεδομένα από κόμβο σε κόμβο. Πιο συγκεκριμένα αυτά τα δεδομένα χωρίζονται σε πακέτα δεδομένων, μηνύματα ελέγχου και μεταδεδομένα.

### 2.2.3 Πακέτα δεδομένων

Τα πακέτα δεδομένων (data packets) ουσιαστικά είναι τα δεδομένα που ρέουν μέσα στο δίκτυο του Netgraph και μετακινούνται από κόμβο σε κόμβο μέσω των αγκίστρων. Έτσι ανάλογα με τη λειτουργία ή τη δομή του κόμβου, τα πακέτα δεδομένων μετασχηματίζονται ώστε να μπορούν να διαβαστούν από αυτόν. Όταν ένας κόμβος λαμβάνει ένα πακέτο δεδομένων, το επεξεργάζεται και στη συνέχεια, συνήθως, το προωθεί σε έναν άλλο κόμβο. Η επεξεργασία μπορεί να είναι απλή όπως η προσθήκη ή αφαίρεση επικεφαλίδων, αλλά μπορεί επίσης να είναι περίπλοκη ή να απαιτεί συνεργασία με άλλα μέρη του συστήματος. Πιο αναλυτικά τα πακέτα δεδομένων μεταφέρονται μέσα σε αλυσίδες mbuf μέσω των αγκίστρων που συνδέουν τους κόμβους. Έτσι κάθε κόμβος επεξεργάζεται τα εισερχόμενα δεδομένα μέσω των αγκίστρων που διαθέτει ανάλογα με τις εκάστοτε προτεραιότητες και λειτουργίες του. Το πρώτο mbuf σε μια αλυσίδα πρέπει να έχει ενεργή την σημαία M\_PKTHDR.

### 2.2.4 Μηνύματα ελέγχου

Αυτή η κατηγορία έχει δύο ειδών μηνύματα ελέγχου, τα γενικά(generic) μηνύματα ελέγχου και τα μηνύματα ελέγχου για συγκεκριμένο τύπο(type-specific). Τα μηνύματα ελέγχου έχουν κοινή μορφή κεφαλίδας(common header), δεδομένα συγκεκριμένου

τύπου(αν υπάρχουν) και δυαδικές δομές. Ουσιαστικά τα μηνύματα ελέγχου είναι δυαδικές δομές για λόγους αποδοτικότητας αλλά οι κόμβοι έχουν την δυνατότητα μετατροπής των δεδομένών του συγκεκριμένο τύπου μεταξύ δυαδικής και ASCII μορφής, για λόγους ελέγχου και αποσφαλμάτωσης. Κάθε μήνυμα ελέγχου περιέχει μια τιμή 32 bit, που ονομάζεται typecookie, δείχνοντας τον τύπο του μηνύματος. Κάθε τύπος μηνύματος ορίζει ένα μοναδικό typecookie για τα μηνύματα που κατανοεί αλλά ένας κόμβος μπορεί να επιλέξει να αναγνωρίζει παραπάνω από ένα τύπο μηνύματος.

Επίσης υπάρχουν τα μηνύματα απάντησης που στέλνονται στην αντίθετη κατεύθυνση από τα μηνύματα ελέγχου. Συγκεκριμένα με το μήνυμα ελέγχου περιέχεται και μια διεύθυνση επιστροφής. Γενικά το βασικό σύστημα ορίζει ένα σύνολο τυποποιημένων μηνυμάτων για σκοπούς ελέγχου ροής και διαχείρισης συνδέσεων, τα οποία συνήθως μεταδίδονται ως μηνύματα απάντησης.

## 2.2.5 Μεταδεδομένα

Τα μεταδεδομένα αποτελούν δεδομένα που έρχονται μαζί με τα πακέτο δεδομένων ή τα μηνύματα ελέγχου. Αυτά παρέχουν πληροφορίες που βοηθούν στο χειρισμό αυτών των δομών στο δίκτυο. Οι τύποι κόμβων μπορούν να ορίσουν τα δικά τους μεταδεδομένα που αφορούν τον τύπο τους και για αυτό το Netgraph ορίζει μια δομή struct ng\_meta. Τα μεταδεδομένα αντιμετωπίζονται ως αδιαφανείς πληροφορίες από το βασικό σύστημα Netgraph. Επιπλέον αν θέλει κάποιος να περάσει μεταδεδομένα μέσω την αλυσίδας mbuf τότε θα πρέπει χρησιμοποιήσει το mbuf\_tags πλαίσιο.

Όμως ο κόμβος θα πρέπει να υποδείξει την διαδρομή που οφείλουν τα παραπάνω δεδομένα να ακολουθήσουν για να φτάσουν στο κόμβο παραλήπτη. Για τον λόγο έχει δημιουργηθεί ένα σύστημα διευθυνσιοδότησης διευκολύνοντας τον χρήστη να ορίσει την διαδρομή.

## 2.2.6 Διευθυνσιοδότηση

Η διαδικασία της διευθυνσιοδότησης έχει σχεδιαστεί με σκοπό να παρέχει μια απλή και γρήγορη λειτουργία στο χρήστη. Συγκεκριμένα ένας κόμβος ή ένα εξωτερικό

στοιχείο(component) το μόνο που χρειάζεται να γνωρίζει είναι το όνομα του κόμβου που θέλει να απευθυνθεί.

Οι διευθύνσεις μπορούν να είναι είτε απόλυτες είτε σχετικές. Μια απόλυτη(absolute) διεύθυνση ξεκινά με το όνομα ή τον αναγνωριστικό αριθμό (ID) ενός κόμβου, ακολουθούμενο από άνω τελεία, και στη συνέχεια από μια σειρά αγκίστρων διαχωρισμένων με τελείες. Αυτή η διαδικασία απευθύνεται στον κόμβο παραλήπτη, ξεκινώντας την διαδρομή από τον ονομαζόμενο κόμβο και των μεταξύ τους αγκίστρων.

Από την άλλη πλευρά, μια σχετική(relative) διεύθυνση περιλαμβάνει μόνο τη σειρά των ονομάτων των αγκίστρων, με τη διαδρομή να ξεκινά υποθετικά από τον τοπικό κόμβο. Αυτό επιτρέπει την έναρξη της πλοήγηση σε ένα δίκτυο από ένα σχετικό σημείο εκκίνησης, χωρίς την ανάγκη αναφοράς σε ένα συγκεκριμένο όνομα ή ID κόμβου.

Υπάρχουν τρεις τρόποι για να διευθυνσιοδοτηθεί ένα μήνυμα ελέγχου. Αν υπάρχει μια ακολουθία αγκίστρων που συνδέουν τους δύο κόμβους, το μήνυμα μπορεί να δρομολογηθεί βάσει πηγής(source-routed). Αυτό γίνεται μέσω της καθορισμένης ακολουθίας ονομάτων αγκίστρων σε ASCII ως τη διεύθυνση προορισμού για το μήνυμα δηλαδή ως σχετική διεύθυνση(relative). Αν ο προορισμός βρίσκεται δίπλα στην πηγή, τότε ο κόμβος πηγής μπορεί απλώς να καθορίσει το άγκιστρο μέσω του οποίου θα πρέπει να σταλεί το μήνυμα. Διαφορετικά, χρησιμοποιείται η παγκόσμια διεύθυνση σε ASCII του κόμβου παραλήπτη ως η διεύθυνση προορισμού για το μήνυμα. Οι δύο τύποι διευθύνσεων σε ASCII μπορούν να συνδυαστούν, καθορίζοντας έναν αρχικό κόμβο και μια ακολουθία αγκίστρων.

## 2.2.7 Τύποι κόμβων

Το Netgraph έχει ήδη κάποιος ενσωματωμένους κόμβους στο σύστημα που καθένας τους έχει δημιουργηθεί για να επιτελεί μια συγκεκριμένη λειτουργία. Μπορούμε να τα ομαδοποιήσουμε ενδεικτικά ως εξής και παρακάτω να δούμε την λειτουργία τους[6].

- 1) Διασυνδέσεις, συσκευές και υποδοχές δικτύου (Network interfaces, devices and sockets)
- 2) Πρωτόκολλα δικτύου(Network protocols)
- 3) Μεταγωγή (Switching)

4) Έλεγχος και διόρθωση(Testing and Debugging)

5) Λοιπά(Other)

Με την εντολή grep ng\_ μπορεί να δούμε όλους του κόμβους του συστήματος, οι οποίοι παρουσιάζονται στον παρακάτω πίνακα.

Πίνακας 2.3. Όλοι οι τύποι κόμβων στο Netgraph.

ng_ether	ng_hub
ng_gif	ng_one2many
ng_tty	ng_tee
ng_iface	ng_split
ng_eiface	ng_etf
ng_device	ng_hole
ng_socket	ng_echo
ng_ksocket	ng_ether_echo
ng_btsocket	ng_ether
ng_cisco	ng_async
ng_frame_relay	ng_atmllc
ng_gif_demux	ng_bluetooth
ng_l2tp	ng_bpf
ng_lmi	ng_car
ng_ppp	ng_ccatm
ng_mppc	ng_checksum
ng_pppoe	ng_hci
ng_pptpgre	ng_ip_input
ng_sppp	ng_ipfw
ng_vlan	ng_macfilter
ng_tcpmss	ng_nat
ng_sscfu	ng_pipe
ng_sscop	ng_source
ng_rfc1490	ng_tag
ng_netflow	ng_ubt
ng_l2cap	ng_UI
ng_deflate	ng_pred1
ng_bridge	

### 2.2.7.1 Διασυνδέσεις, συσκευές και υποδοχές δικτύου

Η υποενότητα με όνομα διασυνδέσεις, συσκευές και υποδοχές δικτύου περιλαμβάνει κόμβους που είναι υπεύθυνοι για την αντιστοίχιση λογισμικού με υλικό, βοηθώντας στη διαχείριση του δικτύου. Επίσης, μέσα από τους κόμβους αυτή της κατηγορίας ο χρήστης μπορεί να κάνει εξομοιώσεις του δικτύου, να παρακολουθήσει την κυκλοφορία του δικτύου και να κάνει γρήγορη και εύκολη εισαγωγή νέων συσκευών στο δίκτυο. Παρακάτω παρουσιάζονται με μια συνοπτική παρουσίαση οι περιεχόμενοι κόμβοι αυτής της κατηγορίας.

Πίνακας 2.4. Όλοι οι τύποι κόμβων της 1<sup>ης</sup> κατηγορίας

Πίνακας 2.4. Όλοι οι τύποι κόμβων της 1 <sup>ης</sup> κατηγορίας	
ng_ether	Ο κόμβος ether επιτρέπει τον χειρισμό και την παρακολούθηση της κυκλοφορίας Ethernet που διέρχεται από τη διασύνδεση(Ethernet interface με το υποσύστημα Netgraph)[7].
ng_gif	Ο κόμβος gif είναι υπεύθυνος για να αλληλοεπιδράνε διασυνδέσεις gif με τον υποσύστημα Netgraph, δηλαδή επιτρέπει τη μεταφορά δεδομένων από το ένα δίκτυο στο άλλο για IPv4 και IPv6[8].
ng_tty	Ο κόμβος tty είναι μια διασύνδεση(interface) μεταξύ των σειριακών συσκευών και του υποσυστήματος Netgraph όπου μπορεί να γίνει ανταλλαγή δεδομένων από και προς σειριακές συσκευών και Netgraph[9].
ng_iface	Ο κόμβος iface είναι ταυτόχρονα ένας κόμβος Netgraph και μια εικονική δικτυακή διασύνδεση (interface) [10] .
ng_eiface	Ο κόμβος eiface υλοποιεί μια εικονική διασύνδεση (interface) Ethernet και όταν δημιουργείται ένας κόμβος eiface, εμφανίζεται μια διασύνδεση (interface ngeth0, ngeth1 κ.λπ.) που είναι προσβάσιμη μέσω της εντολής ifconfig[11].
ng_device	Ο κόμβος device είναι ταυτόχρονα μια διασύνδεση(interface) του συστήματος μιας συσκευής και ένας κόμβος Netgraph[12].

ng_socket	Ο κόμβος socket επιτρέπει στα προγράμματα του χρήστη να συμμετέχουν στο σύστημα Netgraph. Κάθε κόμβος είναι ταυτόχρονα ένας κόμβος Netgraph και ένα ζεύγος υποδοχών της οικογένειας PF_NETGRAPH[13].
ng_ksocket	Ο κόμβος ksocket είναι ο αντίστροφος του ng_socket. Κάθε κόμβος είναι ταυτόχρονα ένας κόμβος και μια υποδοχή που περιέχεται πλήρως στον πυρήνα(kernel)[14].
ng_btsocket	Ο κόμβος btsocket χρησιμοποιείται για τη υλοποίηση τριών τύπων κόμβων Netgraph όπου κάθε τύπος με τη σειρά του υλοποιεί ένα πρωτόκολλο στον τομέα PF_BLUETOOTH[15].

#### 2.2.7.2 Πρωτόκολλα δικτύου

Η υποενότητα με όνομα πρωτόκολλα δικτύου περιλαμβάνει κόμβους που είναι υπεύθυνοι να υλοποιούν τα γνωστά πρωτόκολλα δικτύου. Δηλαδή οι κόμβοι οι οποίοι περιέχονται σε αυτήν την κατηγορία είναι σχεδιασμένοι για να υλοποιούν λειτουργίες που σχετίζονται είτε με την επεξεργασία είτε με τη διαχείριση πρωτοκόλλων δικτύου. Παρακάτω παρουσιάζονται με μια συνοπτική παρουσίαση οι περιεχόμενοι κόμβοι αυτής της κατηγορίας.

Πίνακας 2.5.. Όλοι οι τύποι κόμβων της 2 <sup>ης</sup> κατηγορίας	
ng_cisco	Ο κόμβος cisco είναι υπεύθυνος τόσο για την ενθυλάκωση πακέτων όσο και για την αντίστροφη ενθυλάκωση πακέτων χρησιμοποιώντας το πρωτόκολλο Cisco HDLC [16] .
ng_frame_relay	Ο κόμβος frame_relay είναι υπεύθυνος τόσο για την ενθυλάκωση πακέτων όσο και για την αντίστροφη ενθυλάκωση πακέτων χρησιμοποιώντας το πρωτόκολλο Frame_Relay[17]
ng_gif_demux	Ο κόμβος gif_demux είναι υπεύθυνος για την αποπολύπλεξη των πακέτων εξόδου του κόμβου ng_gif που αναλύσαμε στην παραπάνω κατηγορία[18].
ng_l2tp	Ο κόμβος l2tp είναι υπεύθυνος για την ενθυλάκωση του L2TP(Layer 2 Tunneling Protocol) πρωτοκόλλου. Συγκεκριμένα τοποθετεί την

	κεφαλίδα L2TP στα εξερχόμενα πακέτα αλλά και επίσης αφαιρεί την κεφαλίδα στα εισερχόμενα πακέτα[19]
ng_lmi	Ο κόμβος lmi είναι υπεύθυνος για την ενθυλάκωση πακέτων χρησιμοποιώντας το πρωτόκολλο Frame_Relay LMI(Local Management Interface )[20].
ng_ppp	Ο κόμβος ppp είναι υπεύθυνος ώστε να εκτελεί την πολυπλεξία για το πρωτόκολλο PPP(Point-to-Point Protocol) και χειρίζεται μόνο τα πακέτα αυτά που περιέχουν δεδομένα, τα υπόλοιπα πακέτα όπως πακέτα ελέγχου προωθούνται σε άλλη μονάδα[21].
ng_mppc	Ο κόμβος mppc είναι υπεύθυνος ώστε να υλοποιεί τα υπο-πρωτόκολλα του PPP πρωτόκολλου τα οποία είναι τα MPPC (Microsoft Point-to-Point Compression) και MPPE(Microsoft Point-to-Point Encryption)[22].
ng_pppoe	Ο κόμβος pppoe είναι υπεύθυνος ώστε να υλοποίει το πρωτόκολλο PPPoE (Point-to-Point over Ethernet) δηλαδή ενθυλακώνει PPP πλαίσια μέσα σε Ethernet πλαίσια και το αντίστροφο[23].
ng_pptpgr	Ο κόμβος pptpgr είναι υπεύθυνος για την υλοποίηση του πρωτοκόλλου GRE (Generic Routing Encapsulation) στο πλαίσιο του πρωτοκόλλου PPTP, παρέχοντας δυνατότητες ενθυλάκωσης, διαχείρισης ακολουθιών και επιβεβαίωσης πακέτων. Επιπλέον, υποστηρίζει έναν προσαρμοσμένο μηχανισμό κυλιόμενου παραθύρου για τη διαχείριση των διαλειμμάτων(timeouts)[24].
ng_sppp	Ο κόμβος sppp χρησιμοποιείται ως διασύνδεση(interface) για σύγχρονες γραμμές, προσφέροντας μια εναλλακτική στον πυρήνα για την υλοποίηση του PPP σε αντίθεση με τις λύσεις net/mpd + ng_ppp και ng_cisco. Υποστηρίζει τα πρωτόκολλα PPP και Cisco HDLC, επιτρέποντας απλούστερη διαμόρφωση σε σχέση με το net/mpd + ng_ppp [25].
ng_vlan	Ο κόμβος vlan είναι υπεύθυνος ώστε να πολυπλέκει πλαίσια στα οποία περιέχεται η «ετικέτα» VLAN(Virtual Local Area Network) σύμφωνα με το πρότυπο IEEE 802.1Q μεταξύ των διαφορετικών αγκίστρων τα οποία είναι τα downstream, nomatch και γενικότερα άλλα προσαρμοσμένα άγκιστρα(custom hooks)[26].

ng_tcpmss	Ο κόμβος tcpmss χρησιμοποιείται για να μεταβάλλει την επιλογή του μέγιστου μεγέθους τμήματος των πακέτων TCP[27].
ng_sscfu	Ο κόμβος sscfu χρησιμοποιείται για να υλοποιεί τη σύσταση Q.2130 της ITU-T. Αυτή η σύσταση καθορίζει τη λειτουργία συντονισμού της συγκεκριμένης υπηρεσίας στο UNI[28].
ng_sscop	Ο κόμβος sscop χρησιμοποιείται για να υλοποιεί το πρότυπο Q.2110 της ITU-T. Συγκεκριμένα περιγράφει το λεγόμενο Service Specific Connection Oriented Protocol (SSCOP) που χρησιμοποιείται για τη μεταφορά μηνυμάτων σηματοδοσίας μέσω των ιδιωτικών και δημόσιων UNIs και του δημόσιου NNI[29].
ng_rfc1490	Ο κόμβος rfc1490 στο Netgraph χρησιμεύει στην εκτέλεση των λειτουργιών ενθυλάκωσης, αντίστροφης ενθυλάκωσης, και πολυπλεξίας πρωτοκόλλου, σύμφωνα με τις προδιαγραφές του RFC 1490 [30].
ng_netflow	Ο κόμβος netflow χρησιμοποιείται για την υλοποίηση του πρωτοκόλλου εξαγωγής δεδομένων NetFlow της Cisco[31].
ng_l2cap	Ο κόμβος l2cap χρησιμοποιείται για να υλοποιεί το πρωτόκολλο Bluetooth Logical Link Control and Adaptation Protocol σύμφωνα με το κεφάλαιο D του βιβλίου προδιαγραφών Bluetooth v1.1[32].
ng_deflate	Ο κόμβος deflate χρησιμοποιείται για την υλοποίηση των υπο-πρωτοκόλλων Deflate του Compression Control Protocol (CCP) [33].
ng_pred1	Ο κόμβος pred1 χρησιμοποιείται για να υλοποιεί τα υπο-πρωτόκολλα Predictor-1 του πρωτοκόλλου ελέγχου συμπίεσης (CCP)[34].

### 2.2.7.3 Μεταγωγή

Η υποενότητα με όνομα μεταγωγή περιλαμβάνει κόμβους που συμβάλλουν στην ομαλή μεταφορά δεδομένων μέσω της δημιουργίας αντιγράφων ή της δρομολόγησης σε πολλαπλούς παραλήπτες. Ακόμη γίνεται μετάδοση αυτής της κίνησης σε όλες τις συνδεδεμένες συσκευές με την δυνατότητα φιλτραρίσματος της κίνησης. Παρακάτω παρουσιάζονται με μια συνοπτική παρουσίαση οι περιεχόμενοι κόμβοι αυτής της κατηγορίας.

Πίνακας 2.6. Όλοι οι τύποι κόμβων της 3<sup>ης</sup> κατηγορίας

ng_bridge	Ο κόμβος bridge είναι υπεύθυνος για την σύνδεση ενός ή περισσοτέρων Ethernet συνδέσεις(links) όπου κάθε σύνδεση είναι ουσιαστικά ένα άγκιστρο στο οποίο μεταφέρονται ακατέργαστα πλαίσια Ethernet[35].
ng_hub	Ο κόμβος hub είναι υπεύθυνος για την απλή μετάδοση πακέτων δεδομένων από μία συσκευή σε όλες τις άλλες συσκευές στο δίκτυο[36].
ng_one2many	Ο κόμβος one2many υλοποιεί ένα μηχανισμό ο οποίος δρομολογεί πακέτα σε one-to-many συνδέσμους και αντίστροφα σε many-to-one[37].
ng_tee	Ο κόμβος tee δημιουργεί ένα αντίγραφο όλων των δεδομένων που περνούν από αυτό σε οποιαδήποτε κατεύθυνση (right ή left)[38]
ng_split	Ο κόμβος split χρησιμοποιείται για να διαμοιράσει μια ροή πακέτων σε δύο ροές πακέτων μονής κατεύθυνσης[39].
ng_etf	Ο κόμβος etf χρησιμοποιείται για την πολυπλεξία και φιλτράρισμα των δεδομένων μεταξύ των άγκιστρων με βάση το κάθε ethertype που βρίσκεται στην κεφαλίδα Ethernet[40].

#### 2.2.7.4 Έλεγχος και διόρθωση

Η υποενότητα με όνομα έλεγχος και διόρθωση περιλαμβάνει κόμβους που συμβάλλουν στην ομαλή λειτουργία του δικτύου καθώς ο σκοπός τους είναι να ελέγχουν και να διορθώνουν λειτουργίες και διαδικασίες . Παρακάτω παρουσιάζονται με μια συνοπτική παρουσίαση οι περιεχόμενοι κόμβοι αυτής της κατηγορίας.

Πίνακας 2.7. Όλοι οι τύποι κόμβων της 4<sup>ης</sup> κατηγορίας

ng_hole	Ο κόμβος hole χρησιμοποιείται για έλεγχο και διόρθωση απορρίπτοντας οποιαδήποτε δεδομένα ή μηνύματα ελέγχου λαμβάνει[41].
ng_echo	Ο κόμβος echo χρησιμοποιείται για έλεγχο και διόρθωση ανακλώντας πίσω στο αποστολέα οποιαδήποτε δεδομένα ή μηνύματα ελέγχου λαμβάνει[42].

ng_ether_echo	Ο κόμβος ether_echo χρησιμοποιείται για έλεγχο και διόρθωση ανακλώντας όλα τα μηνύματα δεδομένων και ελέγχου πίσω στον αποστολέα[43].
---------------	---

### 2.2.7.5 Λοιπά

Η υποενότητα με όνομα λοιπά περιλαμβάνει κόμβους που δεν ταιριάζουν με τις υπόλοιπες κατηγορίες. Παρακάτω παρουσιάζονται με μια συνοπτική παρουσίαση οι περιεχόμενοι κόμβοι αυτής της κατηγορίας.

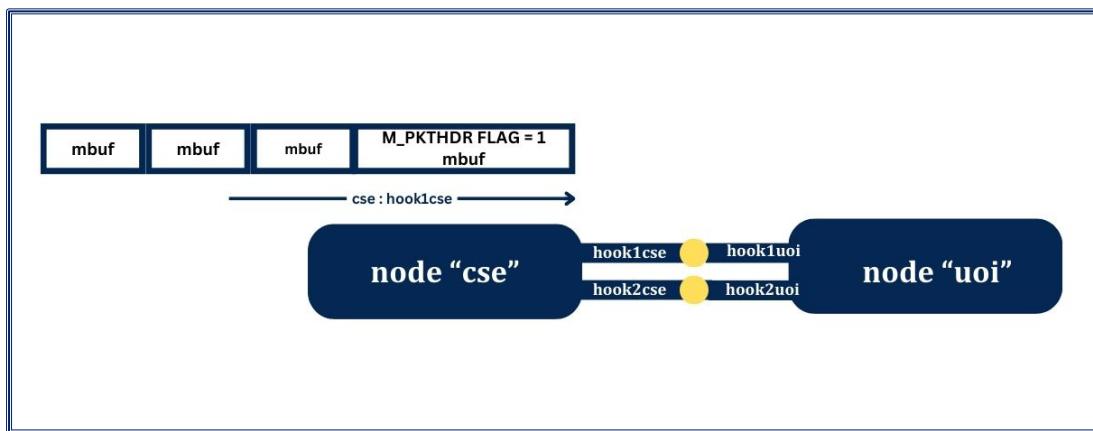
Πίνακας 2.8. Όλοι οι τύποι κόμβων της 5 <sup>ης</sup> κατηγορίας	
ng_async	Ο κόμβος async χρησιμοποιείται για τη μετατροπή των σύγχρονων(synchronous) πλαισίων και των ασύγχρονων(asynchronous) πλαισίων, όπως ορίζεται για το πρωτόκολλο PPP στο RFC 1662[44].
ng_atmllc	Ο κόμβος atmllc χρησιμοποιείται για να μεταφράσει τα πλαίσια προς και από την ενθυλάκωση ATM LLC, όπως ορίζεται από το RFC 1483[45].
ng_bluetooth	Ο κόμβος bluetooth χρησιμοποιείται ως κάτοχος(placeholder) για παγκόσμιες μεταβλητές Bluetooth και όλες οι μεταβλητές μπορούν να εξεταστούν και να αλλάξουν μέσω του sysctl[46].
ng_bpf	Ο κόμβος bpf χρησιμοποιείται για τη εφαρμογή φύλτρων Berkeley Packet Filter σε δεδομένα που ταξιδεύουν μέσω ενός δικτύου Netgraph[47].
ng_car	Ο κόμβος car χρησιμοποιείται για να περιορίζει την κυκλοφορία που ρέει μέσω αυτού χρησιμοποιώντας ένα ενιαίο δείκτη τριών χρωμάτων[48].
ng_ccatm	Ο κόμβος ccatm χρησιμοποιείται για να υλοποιεί το API που καθορίζεται από το ATM Forum για την πρόσβαση σε υπηρεσίες ATM [49].
ng_checksum	Ο κόμβος checksum χρησιμοποιείται για την ανίχνευση σφαλμάτων που μπορεί να έχουν εισαχθεί κατά τη μετάδοση ή την αποθήκευση. Ακόμη χρησιμοποιείται για την επαλήθευση της ακεραιότητας των δεδομένων[50].

ng_hci	Ο κόμβος hci χρησιμοποιείται για την υλοποίηση του επίπεδο Bluetooth Host Controller Interface (HCI) σύμφωνα με το βιβλίο προδιαγραφών Bluetooth v1.1[51]
ng_ip_input	Ο κόμβος ip_input χρησιμοποιείται ως σημείο εισόδου για όλα τα εισερχόμενα IP πακέτα, τα οποία διαχειρίζεται προωθώντας τα στην ουρά του υποσυστήματος επεξεργασίας IP για περαιτέρω ανάλυση και επεξεργασία.[52]
ng_ipfw	Ο κόμβος ipfw χρησιμοποιείται για να υλοποιεί διασύνδεση μεταξύ των υποσυστημάτων ipfw(IP firewall) και Netgraph. Μόλις φορτωθεί το ipfw στον πυρήνα, δημιουργείται ένας μόνο κόμβος και δεν μπορούν να δημιουργηθούν άλλοι κόμβοι ipfw[53].
ng_macfilter	Ο κόμβος macfilter χρησιμοποιείται για να την δρομολόγηση πακέτων ethernet μέσω διαφορετικών αγκίστρων με βάση τη διεύθυνση MAC του αποστολέα[54].
ng_nat	Ο κόμβος nat χρησιμοποιείται για την μετάφραση διευθύνσεων δικτύου (NAT) των πακέτων IPv4 που περνούν από αυτόν και χρησιμοποιεί τη μηχανή libalias για την αλλοίωση πακέτων[55].
ng_pipe	Ο κόμβος rpipe χρησιμοποιείται στη χειραγώγηση της δικτυακής κυκλοφορίας μέσω της προσομοίωσης εύρους ζώνης, καθυστερήσεων και τυχαίων απωλειών πακέτων, παρέχοντας έναν τρόπο για την εξομοίωση διαφόρων δικτυακών συνθηκών. [56].
ng_source	Ο κόμβος source χρησιμοποιείται ως πηγή πακέτων σύμφωνα με τις παραμέτρους που έχουν οριστεί με τη χρήση μηνυμάτων ελέγχου και πακέτων εισόδου. Χρησιμοποιείται κυρίως για δοκιμές και συγκριτική αξιολόγηση[58].
ng_tag	Ο κόμβος tag χρησιμοποιεί ετικέτες πακέτων mbuf για να εξεταστούν, να αφαιρεθούν ή να εφαρμοστούν σε δεδομένα που ταξιδεύουν μέσω ενός δικτύου Netgraph[59].
ng_ubt	Ο κόμβος ubt είναι ταυτόχρονα ένας μόνιμος τύπος κόμβου Netgraph και ένας οδηγός για συσκευές Bluetooth USB[60].
ng_UI	Ο κόμβος UI χρησιμοποιείται ώστε να προωθεί πακέτα με 0x03 ως πρώτο byte ή να προσθέτει το 0x03 byte σε καινούργια πακέτα[61].

## 2.3 Παραδείγματα διασύνδεσης δομών

Παρακάτω παρατίθενται απεικονίσεις λειτουργίας του συστήματος και συνεργασίας όλων των δομών που επεξηγήθηκαν στην προηγούμενη ενότητα.

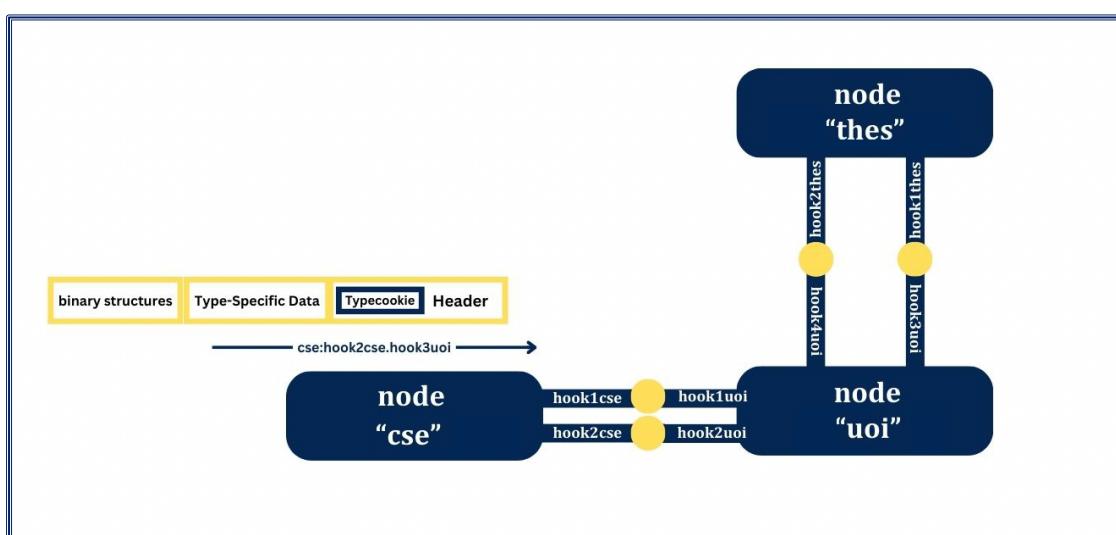
Στην παρακάτω εικόνα εμφανίζεται ένα δίκτυο στο όποιο γίνεται μεταφορά δεδομένων από τον έναν κόμβο σε έναν άλλον. Αρχικά υπάρχουν δύο κόμβοι με ονόματα cse και uoi και ο καθένας έχει δυο άγκιστρα. Ο cse έχει τα άγκιστρα hook1cse, hook2cse και ο uoi έχει τα άγκιστρα hook1uoi, hook2uoi. Όπως αναφέρθηκε παραπάνω ένα άγκιστρο συνδέεται μόνο με αλλά άγκιστρα και τότε δημιουργείται ζεύγος αγκίστρων. Έτσι για να συνδεθούν οι κόμβοι παρατηρούνται οι συνδέσεις (με κίτρινο χρώμα) hook1cse.hook1uoi και hook2cse.hook2uoi. Αφού έγινε σύνδεση μεταξύ των κόμβων ο κόμβος cse θέλει να στείλει ένα πακέτο δεδομένων στο κόμβο uoi και αυτό γίνεται αντιληπτό από την διεύθυνσιοδότηση του πακέτου που είναι cse.hook1cse. Ο χρήστης αντιλαμβάνεται ότι πρόκειται για πακέτο δεδομένων καθώς εμπεριέχεται μια αλυσίδα από mbuf δομές που μεταφέρουν την πληροφορία και το σημαντικότερο στο πρώτο πακέτο της αλυσίδας παρατηρείται ότι η σημαία M\_PKTHDR FLAG = 1 δηλαδή είναι ενεργή.



Εικόνα 2.2. Αναπαράσταση δικτύου με αποστολή πακέτου δεδομένου

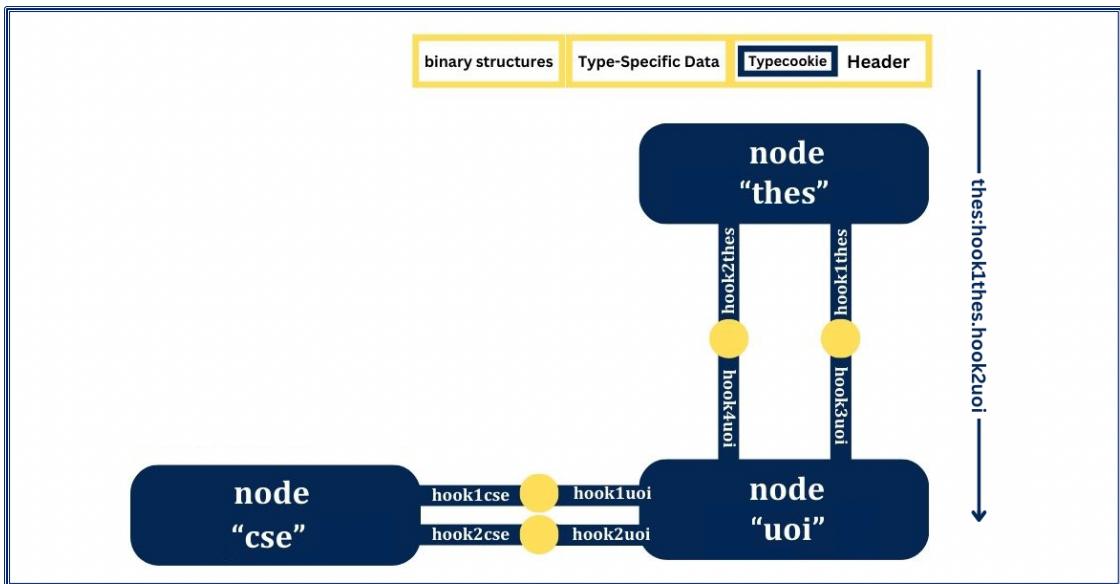
Συνεχίζοντας στο ίδιη υπάρχων δίκτυο θα προστεθεί ένας ακόμα κόμβος με το όνομα thes. Επομένως θα υπάρχουν και νούργιες συνδέσεις κόμβων μέσω των άγκιστρων τους. Συγκεκριμένα παρατηρούνται οι συνδέσεις (με κίτρινο χρώμα) hook4uoi.hook2.thes και hook3uoi.hook1.thes. Τώρα ο κόμβος cse θέλει να στείλει ένα μήνυμα ελέγχου στο κόμβο thes και αυτό γίνεται αντιληπτό μέσω της διεύθυνσιοδότησης του μηνύματος

cse:hook2cse.hook3uoi. Δηλαδή το μήνυμα θα ακολουθήσει την διαδρομή από το cse και θα εξέλθει από το άγκιστρο hook2cse που είναι συνδεμένο με το άγκιστρο hook2uoi και έτσι θα εισέλθει στο κόμβο uoi. Έπειτα το μήνυμα έχει οδηγία να εξέλθει από το κόμβο uoi μέσω του hook3uoi και να εισέλθει στο κόμβο thes μέσω του hook1thes. Να σημειωθεί ότι κόμβος uoi δεν τροποποιεί καθόλου το μήνυμα απλά το προωθεί στο thes κόμβο. Επιπλέον ο χρήστης αντιλαμβάνεται ότι πρόκειται για μήνυμα ελέγχου καθώς παρατηρεί ότι η δομή του απαρτίζεται ξεκινώντας με την κεφαλίδα, ακολουθούν τα δεδομένα συγκεκριμένου τύπου και τέλος υπάρχουν οι δυαδικές δομές του μηνύματος.



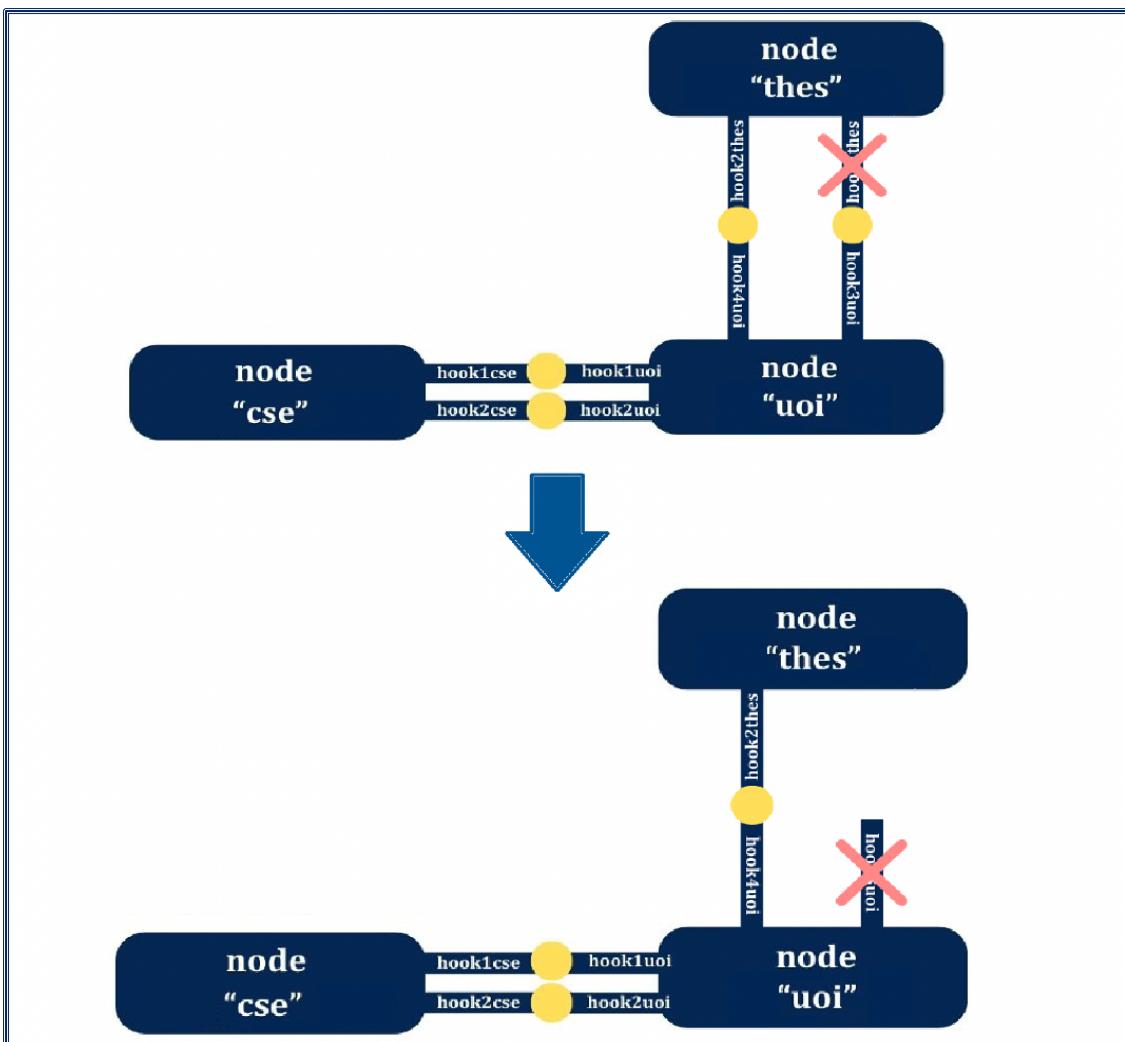
Εικόνα 2.3. Αναπαράσταση δικτύου με αποστολή μηνύματος ελέγχου

Στην παρακάτω εικόνα εμφανίζεται ένα μήνυμα απάντησης από το κόμβο thes στο κόμβο cse. Μπορεί να παρατηρήσει κάποιος ότι το μήνυμα απάντησης έχει την δομή του μηνύματος ελέγχου, αφού πρακτικά το μήνυμα απάντησης είναι ένα μήνυμα ελέγχου με αντίθετη κατεύθυνση.



Εικόνα 2.4. Αναπαράσταση δικτύου με αποστολή μηνύματος απάντησης

Τέλος απεικονίζεται να σπάει το ένα hook από την σύνδεση του κόμβου **thes** με το **uoi**, επομένως σπάει και η άλλη μεριά της σύνδεσης αυτόματα και μένει το **hook3uoi** μόνο του, αφού η σύνδεση (κίτρινη βούλα) έσπασε.



Εικόνα 2.5. Αναπαράσταση δικτύου με καταστροφή αγκίστρου

## 2.4 Παρόμοια εργαλεία διαχείρισης του υποσυστήματος δικτύωσης

Υπάρχουν και άλλα εργαλεία που προσφέρουν παρόμοια λειτουργικότητα με το Netgraph και παρακάτω παρατίθενται ενδεικτικά κάποια από αυτά. Συγκεκριμένα τα εργαλεία αυτά επιτρέπουν την τροποποίηση του υποσυστήματος δικτύωσης, βοηθώντας στην καλύτερη διαχείριση του δικτύου. Συγκεκριμένα τα εργαλεία είναι τα iproute2, Open vSwitch και eBPF.

Το iproute2 είναι ένα σύνολο εργαλείων για τη διαχείριση των δικτύων στο Linux. Περιλαμβάνει το ipl για διαχείριση διευθύνσεων και δρομολογίων, το tc για έλεγχο της

κυκλοφορίας και άλλες λειτουργίες δικτύου. Το iproute2 επιτρέπει τη διαμόρφωση IP διευθύνσεων και δρομολογίων, την εφαρμογή traffic shaping και quality of service (QoS) και την παρακολούθηση και διαχείριση της σύνδεσης δικτύου. Είναι ιδανικό για προχωρημένη διαχείριση δικτύων στο Linux, προσφέροντας δυνατότητες στο χρήστη.

Το Open vSwitch είναι ένα πολυλειτουργικό εικονικό μεταγωγέα(virtual switch) σχεδιασμένο για προχωρημένη διαχείριση δικτύου σε εικονικά περιβάλλοντα. Επιτρέπει τη διαχείριση εικονικών δικτύων, την υλοποίηση προχωρημένων πολιτικών δικτύου και ACLs, και τη διασύνδεση με εργαλεία νέφους(cloud) όπως το OpenStack.

Το eBPF είναι μια τεχνολογία στο Linux που επιτρέπει την εκτέλεση δοκιμαστικών προγραμμάτων (sandboxed) στον πυρήνα του λειτουργικού συστήματος. Με το eBPF μπορεί ο χρήστης να παρακολουθεί και να αναλύει το δίκτυο σε πραγματικό χρόνο, να εφαρμόζει πολιτικές ασφάλειας και τοίχο προστασίας (firewall). Το eBPF προσφέρει πολύ χαμηλό (latency) και υψηλή απόδοση, καθιστώντας το ιδανικό για προχωρημένες ανάγκες διαχείρισης δικτύου.

## Κεφάλαιο 3. Εγκατάσταση και χρήση του Netgraph

### 3.1 Εγκατάσταση του Netgraph

Το Netgraph όπως είπαμε και παραπάνω είναι σχεδιασμένο ως μέρος του πυρήνα του FreeBSD και συνήθως είναι ενσωματωμένο στις περισσότερες εγκαταστάσεις του FreeBSD.

Επομένως πρώτη δουλεία είναι να γίνει έλεγχος αν το σύστημα FreeBSD είναι ενημερωμένο. Για να γίνει ο έλεγχος προϋπόθεση θεωρείτε η είσοδος στο σύστημα FreeBSD ως διαχειριστής (root). Έπειτα ο έλεγχος γίνεται με την με την εντολή pkg install update στο UNIX shell του συστήματος(προαιρετικά).

```
root@:/ # pkg install update
Updating FreeBSD repository catalogue...
Fetching packagesite.pkg: 100%    7 MiB   3.6MB/s  00:02
Processing entries: 100%
FreeBSD repository update completed. 34086 packages processed.
All repositories are up to date.
```

Εικόνα 3.1. Εντολή ενημέρωσης του FreeBSD

Για να φορτώσει κάνεις το Netgraph υπάρχουν δυο τρόποι, ο δυναμικός και ο στατικός. Ο δυναμικός χρησιμοποιεί την ήδη εγκατεστημένη βιβλιοθήκη kld ,έχοντας την δυνατότητα ενημέρωσης και διόρθωσης λογισμικού χωρίς την ανάγκη επανεκκίνησης του συστήματος. Από την άλλη με τον στατικό απαιτείται μεταγλώττιση του πυρήνα (compile στο kernel) και επεξεργασία του αρχείου που περιέχει όλες τις ρυθμίσεις του πυρήνα. Έτσι επιτυγχάνουμε καλύτερη απόδοση κατά την εκκίνηση του συστήματος, αφού όλοι οι απαραίτητοι κώδικες βρίσκονται ήδη ενσωματωμένοι μέσα στο εκτελέσιμο αρχείο.

### 3.1.1 Δυναμική εγκατάσταση

Για την φόρτωση του Netgraph ως μονάδα(modules) του πυρήνα(kernel) στο σύστημα, χρησιμοποιείται η εντολή kldload netgraph.ko[62]. Η επιτυχής φόρτωση της μονάδας επαληθεύεται με τη χρήση της εντολής kldstat. Σε περίπτωση που παρουσιαστεί έλλειψη αντίστοιχων δικαιωμάτων ενώ η εκτέλεση των παραπάνω εντολών πραγματοποιείται από χρήστη με δικαιώματα διαχειριστή (root) τότε, θα πρέπει να προηγηθεί η χρήση της εντολής sudo στην αρχή της εντολής, για την εξασφάλιση των απαιτούμενων δικαιωμάτων εκτέλεσης.

Id	Refs	Address	Size	Name
1	9	0xffffffff80200000	1f3e2d0	kernel
2	1	0xffffffff8213f000	5b78	ng_ether.ko
3	2	0xffffffff82145000	16d70	netgraph.ko
4	1	0xffffffff82518000	3218	intpm.ko
5	1	0xffffffff8251c000	2180	smbus.ko

Εικόνα 3.2. Εντολή αναπαράστασης ενεργών modules στο kernel

Για την προβολή των διαθέσιμων μονάδων (modules) που έχουν ενσωματωθεί στον πυρήνα του συστήματος, μπορείτε να εκτελέσετε την εντολή ls /boot/kernel/ με τύπο ng\_module.

### 3.1.2 Στατική εγκατάσταση

Στην περίπτωση όμως που η επιλογή της δυναμικής εγκατάσταση δεν είναι επιθυμητή, μπορείτε να επιλέξετε την στατική εγκατάσταση. Η διαδικασία αυτή περιλαμβάνει την δημιουργία και προσθήκη των απαραίτητων επιλογών στο προσαρμοσμένο αρχείου πυρήνα(custom kernel file) και την μετέπειτα διαδικασία μεταγλώττισης(compile) του, προκειμένου να ενσωματωθεί η λειτουργία Netgraph. Αρχικά, είναι απαραίτητο να πραγματοποιηθεί η πλοήγηση εντός του Unix shell προς τον κατάλογο διαμόρφωσης(conf). Για να επιτευχθεί αυτό, πρέπει να εκτελεστεί η ακόλουθη εντολή cd /usr/src/sys/amd64/conf.

Στη συνέχεια, προκειμένου να διατηρηθεί ατόφιο το αρχικό αρχείο ρυθμίσεων του πυρήνα, δηλαδή το αρχείο GENERIC, απαιτείται η δημιουργία ενός αντιγράφου του. Η δημιουργία αντιγράφου επιτυγχάνεται μέσω της εντολής cp GENERIC yourKernel, όπου yourKernel αποτελεί την ονομασία που επιθυμείτε να δώσετε στο νέο αρχείο ρυθμίσεων. Με αυτόν τον τρόπο, εξασφαλίζεται ότι οι αρχικές ρυθμίσεις παραμένουν ανέπαφες. [63-64].

```

/usr/src # cd sys
/usr/src/sys # ls
  dev           Makefile      netsmb      security
  dts           mips          nfs          sys
  fs            modules       nfsclient   taken
  gdb           net           nfsserver   tests
  geom          net80211    nlm          tools
  gnu           netgraph     ofed         ufs
  i386          netinet      opencrypto  vm
  isa            netinet6    powerpc     x86
  kern          netipsec    README.md   xdr
  kgssapi       netlink      riscv        xen
  libkern       netpfifl    rpc
/usr/src/sys # cd amd64
/usr/src/sys/amd64 # ls
  cloudbabi64  include      Makefile      vmm
  conf          linux        pci
  ia32          linux32     sgx
/usr/src/sys/amd64 # cd conf
/usr/src/sys/amd64/conf # ls
  GENERIC-KCSAM LINT        LINT-NOIP    testKernel
  GENERIC-MMCCAM LINT-NOINET MINIMAL
  GENERIC,hints LINT-NOINET6 NOTES
/usr/src/sys/amd64/conf #

```

Εικόνα 3.3. Εύρεση αρχείου GENERIC

```

[ (escape) menu ^y search prompt ^k delete line  ^p prev li
^o ascii code  ^x search      ^l undelete line ^n next li
^u end of file ^a begin of line ^w delete word  ^b back 1 char
^t top of text ^e end of line  ^r restore word  ^f forward char
^c command     ^d delete char  ^j undelete char  E
=====line 29 col 30 lines from top 29 =====
* An exhaustive list of options and more detailed explanations of
* device lines is also present in the ./../conf/NOTES and NOTES
* If you are in doubt as to the purpose or necessity of a line, c
* in NOTES.
*
* $FreeBSD$


cpu          HAMMER
ident        GENERIC

makeoptions DEBUG=-g          # Build kernel with gdb(1)
makeoptions WITH_CTF=1        # Run ctfconvert(1) for D

options      NETGRAPH          #<-----||<<-----
options      NETGRAPH_ETHER
options      SCHED_ULE         # ULE scheduler
options      NUMA              # Non-Uniform Memory Arch

```

Εικόνα 3.4. Επεξεργασία custom kernel(testKernel) αρχείου

Στο συγκεκριμένο παράδειγμα το αντιγραμμένο αρχείο πυρήνα ονομάζεται testKernel. Το άνοιγμα του αρχείου επιτυγχάνεται με την χρήση της εντολής ee testKernel. Ωστόσο κάποιες αξιόλογες εναλλακτικές για πρόσβαση στο αρχείο αποτελούν οι εντολές vi ή nano. Κατά την επεξεργασία του αρχείου, είναι απαραίτητο να προστεθεί η εντολή options NETGRAPH στο εν λόγω αρχείο. Αυτή η εντολή ενεργοποιεί τις συγκεκριμένες λειτουργίες του Netgraph στον πυρήνα. Ακόμη μπορεί να υπάρξει προσθήκη επιπρόσθετων εντολών, όπως η options NETGRAPH\_ETHER που ενεργοποιεί τις λειτουργίες του ether στο Netgraph.

Για την ολοκλήρωση της διαδικασίας απαραίτητη καθίσταται η εντολή make buildkernel KERNCONF=testKernel που είναι υπεύθυνη για την παραγωγή του μεταγλωττισμένου πυρήνα (compiled kernel) και των αρχείων που απαιτούνται. Επιπλέον μετά την επιτυχή μεταγλώττιση χρειάζεται η εντολή make installkernel KERNCONF=testKernel ώστε να εγκαταστήσει στη σωστή τοποθεσία τον μεταγλωττισμένο πυρήνα. Τέλος για την χρήση του προσαρμοσμένου πλέον πυρήνα χρειάζεται η επανεκκίνηση του συστήματος.

## 3.2 Χρήση του Netgraph

Υπάρχουν δύο βοηθητικά προγράμματα (utilities) γραμμής εντολών για την αλληλεπίδραση του χρήστη με το Netgraph, το nghook και το ngctl. Το nghook είναι απλό, η δουλεία του είναι να συνδέεται με οποιοδήποτε μη συνδεδεμένο άγκιστρο οποιουδήποτε υπάρχοντος κόμβου και να σας επιτρέπει να μεταδίδετε και να λαμβάνετε πακέτα δεδομένων μέσω της τυπικής εισόδου και της τυπικής εξόδου. Το ngctl είναι ένα πιο περίπλοκο πρόγραμμα που σας επιτρέπει να κάνετε πληθώρα πράγματων στο Netgraph μέσω της γραμμής εντολών. Τέλος υπάρχει η δυνατότητα χρήσης του ngctl τόσο σε λειτουργία συνόλου (batch) δηλαδή την εισαγωγή των εντολών ως σενάριο (script) αλλά και σαν διαδραστική λειτουργία, δηλαδή συνεχή εισαγωγή εντολών από τον χρήστη.

### 3.2.1 Εντολή nghook

Υπάρχουν δύο τρόποι χρήσης τις εντολής ώστε να προσφέρει ευελιξία στο τι θέλει να πετύχει ο χρήστης. Οι δύο τρόποι παρατίθενται παρακάτω και η κύρια διαφορά τους είναι η παρουσία ή απουσία του ορίσματος -e. [2-65]

**nghook [-adlnSs][-m msg] path [hookname]**

**nghook -e [-n] [-m msg] path hookname program [args ...]**

Όταν παρέχεται η επιλογή -e, το τρίτο όρισμα αναλύεται ως η διαδρομή(path) προς ένα πρόγραμμα, το οποίο στη συνέχεια εκτελείται χρησιμοποιώντας τα υπόλοιπα ορίσματα ως τα δικά του. Πριν από την εικίνηση του προγράμματος, αποστέλλονται προς τον κόμβο μηνύματα Netgraph που έχουν καθοριστεί μέσω της επιλογής -m(msg). Η εκτέλεση του προγράμματος πραγματοποιείται με την τυπική(standard) είσοδο και έξοδο συνδεδεμένη με τον συγκεκριμένο άγκιστρο (hookname), εκτός εάν η τυπική είσοδος έχει κλείσει λόγω της επιλογής -n.

Εάν απουσιάζει η επιλογή -e, οποιαδήποτε δεδομένα εισάγονται μέσω της τυπικής εισόδου αποστέλλονται απευθείας στον κόμβο και όλα τα δεδομένα που λαμβάνονται από τον κόμβο μεταφέρονται στην τυπική έξοδο. Μηνύματα που ορίζονται με την επιλογή -m στέλνονται προς τον κόμβο προτού ξεκινήσει οποιαδήποτε επανάληψη. Το εργαλείο nghook τερματίζει τη λειτουργία του όταν ανιχνεύει το σήμα EOF στην τυπική είσοδο σε αυτή την περίπτωση.

Για την διευκόλυνση της ανάλυσης της εντολής θα γίνει διαχωρίσμος σε τμήματα ανάλογα με την θέση τους ορίσματος που βρίσκονται.

**[-adlnSs]**

Μπορεί να χρησιμοποιηθεί ένα από τα γράμματα που περιέχονται μέσα στο [-adlnSs] ανάλογα με την χρήση που ζητάμε :

Πίνακας 3.1. Επεξήγηση επιλογών χρήσης

<b>-a</b>	<b>-d</b>	<b>-l</b>	<b>-n</b>	<b>-s</b>	<b>-s</b>
Μετατρέπει στην έξοδο τα επεξήγηση	Αυξάνει την επεξήγηση	Εκτός από το να γράφει τα	Χρησιμοποιείται σαν διακόπτης ώστε το nghook	Χρησιμοποιεί ως περιγραφέα	Χρησιμοποιεί ως περιγραφέα

δεδομένα από δυαδική μορφή σε ASCII	των λαθών κατά το debug.	δεδομένα που λαμβάνει στην έξοδο τα επιστρέφει και πίσω στο άγκιστρο που τα έλαβε.	να μην διαβάζει από την standard είσοδο αλλά αποκλειστικά από τον συνδεδεμένο κόμβο. Η σταματάει με την λήψη κάποιου σήματος.	αρχείου (file descriptor) το 1 (stdout) αντί του 0(stdin) που είναι default τιμή.	αρχείου (file descriptor) το 0(stdin) αντί του 1(stdout) που είναι default τιμή
-------------------------------------	--------------------------	--	---	---	---

`[-m msg]`

Στέλνει το μήνυμα ελέγχου που δίνει ο χρήστης σε ASCII μορφή στο κόμβο και μπορεί να χρησιμοποιηθεί πολλές φορές δηλαδή κάθε φορά να δίνετε διαφορετικό μήνυμα.

`[hookname]`

Ουσιαστικά η `nghook` δημιουργεί έναν κόμβο τύπου `socket` (`ng_socket`) και το συνδέει με το άγκιστρο με όνομα «`hookname`» του κόμβου που βρίσκεται στο `path`. Εάν το `hookname` παραλείπεται, θεωρείται ότι είναι "debug".

`path`

To `path` που βρίσκεται ο κόμβος και εισάγεται από το χρήστη σαν όρισμα μέσω του command line.

`program[args ...]`

Το όνομα του `program` το οποίο εισάγεται από το χρήστη σαν όρισμα μέσω του command line.

Παρακάτω παρατίθεται ένα απλό παράδειγμα χρήσης του `nghook`

Αν ο πυρήνα(kernel) σας έχει κάνει μεταγλώττιση (compile ) με τις options NETGRAPH και έχετε μια διασύνδεση(interface) Ethernet fpx0, αυτή η εντολή θα ανακατευθύνει όλα τα πακέτα που λαμβάνει η κάρτα Ethernet και θα τα μεταφέρει στην τυπική(standard) έξοδο σε μορφή hex/ASCII.

+ **nghook -a fpx0: divert**

### 3.2.2 Εντολή ngctl

Το ngctl είναι η «γέφυρα» μεταξύ user space και kernel, μέσω αυτού αλληλοεπιδρά ο χρήστης και γίνονται η πλειοψηφία τόσο των απλών όσο και των περίπλοκων διαδικασιών και λειτουργιών του Netgraph. [2-66]

Επίσης το ngctl είναι σχεδιασμένο να δημιουργεί έναν νέο κόμβο τύπου socket στο Netgraph, ο οποίος μπορεί να χρησιμοποιηθεί για την εκτέλεση διαφόρων εντολών Netgraph. Παρακάτω αναλύετε το συντακτικό και η χρήση της κάθε εντολής.

**ngctl [-d] [-f filename][-n nodename] [command ...]**

Εάν δεν οριστεί η επιλογή -f και δεν δοθεί καμία εντολή από τις παρακάτω μέσω της γραμμής εντολών τότε το ngctl θα εισέλθει σε διαδραστική λειτουργία.

-d

Αυξάνει την επεξήγηση των λαθών κατά το debug.

-f filename

Διαβάζει εντολές από το αρχείο που δίνεται . Η παύλα '-' σημαίνει standard έξοδο ενώ παράλληλα κενές γραμμές ή γραμμές με '#' αγνοούνται.

n nodename

Ονομάζει το νέο κόμβο με το όνομα που δίνει ο χρήστης στο nodename. Στην default λειτουργία το όνομα είναι ngctlAAA όπου AAA το processID.

## [command]

Η χρήση των εντολών command μπορεί να γίνει με συνδυασμό των παραπάνω επιλογών και την χρήσης κάποιας εντολής [command]. Πιο σύνηθες σενάριο όμως είναι η είσοδος του χρήστη στο μενού ngctl με την πληκτρολόγηση της εντολής ngctl στο command line, η εμφάνιση του παρακάτω μενού εντολών και η μετέπειτα χρήση της εκάστοτε εντολής. Έτσι επιτυγχάνεται η αμεσότητα και η διευκόλυνση του χρήστη στη αλληλεπίδραση του με το Netgraph αφού εισέρχεται πλέον σε διαδραστική λειτουργία.

Πίνακας 3.2. Αναπαράσταση εντολών ngctl	
<b>config</b>	get or set configuration of node at <path>
<b>connect</b>	Connects hook <peer hook> of the node at <relpath> to <hook>
<b>debug</b>	Get/set debugging verbosity level
<b>dot</b>	Produce a Graphviz (.dot) of the entire Netgraph.
<b>help</b>	how command summary or get more help on a specific command
<b>list</b>	Show information about all nodes
<b>mkpeer</b>	Create and connect a new node to the node at "path"
<b>msg</b>	Send a Netgraph control message to the node at "path"
<b>name</b>	Assign name <name> to the node at <path>
<b>read</b>	Read and execute commands from a file
<b>rmhook</b>	Disconnect hook "hook" of the node at "path"
<b>show</b>	Show information about the node at <path>
<b>shutdown</b>	Shut down the node at <path>
<b>status</b>	Get human readable status information from the node at <path>
<b>types</b>	Show information about all installed node types
<b>write</b>	Send a data packet down the hook named by "hook".
<b>quit</b>	Exit program

---

**config** → Μέσω αυτής της εντολής μπορεί ο χρήστης να πάρει ή να δώσει στοιχεία διαμόρφωσης (configuration) όπως ρυθμίσεις/παραμέτρους από ή προς στον κόμβο που είναι στο <path> της ngctl.

---

**usage : config <path> [arguments]**

---

**connect** → Μέσω αυτής της εντολής εγκαθιδρύεται μια σύνδεση μεταξύ κόμβων που διασυνδέονται με τα κατάλληλα άγκιστρα. Συγκεκριμένα δημιουργεί μια σύνδεση μεταξύ των <path> και <relpath> χρησιμοποιώντας σαν άγκιστρα τα <hook> και <peerhook> αντίστοιχα. Άμα το relpath δεν είναι συγκεκριμένο καθορίζεται από το path. Άμα το path παραλείπεται τότε «θεωρείτε» το κενό path.

**usage : connect [path] <relpath> <hook> <peerhook>**

Αντί για την λέξη connect μπορεί να χρησιμοποιηθεί η λέξη join η οποία κάνει ακριβώς τα ίδια πράγματα.

---

**debug** → Μέσω αυτής της εντολής ο χρήστης μπορεί είτε να δει (get) είτε να αλλάξει(set) μέσω του -d το μέγεθος της επεξήγησης που δίνει το σύστημα για το debug.

**usage : debug [arguments]**

Αν το [arguments] είναι +/- τότε μεγαλώνει το επίπεδο αποσφαλμάτωσης(debug level),αλλιώς μπορεί να γίνει και με την εντολή παρακάτω

```
root@:~ # ngctl debug
Current debug level is 0
root@:~ # ngctl -d debug
Current debug level is 1
```

Εικόνα 3.5 . Εναλλαγή του επιπέδου debug

---

**dot** → Μέσω αυτής της εντολής δημιουργείται ένα γράφημα [Graphviz](#) (.dot) για ολόκληρο το δίκτυο Netgraph που έχουμε εκείνη την στιγμή .

**usage : dot [-c] [outputfile]**

Αν δεν δίνεται κάποιο συγκεκριμένο outputfile τότε εννοείται το stdout. Όταν χρησιμοποιείται η επιλογή -c τότε δημιουργεί ένα γράφημα χωρίς τα ονόματα των ακμών το οποίο είναι πιο περιεκτικό.

Αντί για την λέξη dot μπορεί να χρησιμοποιηθούν οι λέξεις graphviz ή confdot οι οποίες κάνουν ακριβώς τα ίδια πράγματα.

---

**help** → Μέσω αυτής της εντολής παρέχεται βοήθεια στο χρήστη σχετικά με κάποια συγκεκριμένη εντολή που αντιμετωπίζει πρόβλημα και δίνεται μια μικρή περιγραφή της λειτουργίας της.

**usage : help [command]**

**list** → Μέσω αυτής της εντολής παρέχονται πληροφορίες για τον κάθε κόμβο που βρίσκεται εκείνη την στιγμή στο σύστημα του Netgraph.

**usage : list [-ln]**

Η επιλογή -l παρέχει αναλυτικές πληροφορίες τόσο για την έξοδο όσο και για το άγκιστρο της εξόδου ,ενώ η επιλογή -n περιορίζει την λίστα αναπαράστασης των κόμβων, δείχνοντας μόνο εκείνους που έχουν ανάθεση παγκόσμιου (global)ονόματος.

Αντί για την λέξη list μπορεί να χρησιμοποιηθεί η λέξη ls η οποία κάνει ακριβώς τα ίδια πράγματα.

**mkpeer** → Μέσω αυτής της εντολής δημιουργείται και συνδέεται ένας νέος κόμβος στο κόμβο που βρίσκεται στο path.

**usage : mkpeer [path] <type><hook><peerhook>**

Συγκεκριμένα η εντολή mkpeer φτιάχνει ένα νέο κόμβο τύπου type και τον συνδέει στο path που βρίσκεται ο προϋπάρχω κόμβος ,άμα δεν δίνετε το path τότε «θεωρείτε» το κενό. Τα άγκιστρα που χρησιμοποιούνται είναι τόσο το άγκιστρο<hook> που συνδέεται στο προϋπάρχω κόμβο όσο και το άγκιστρο <peerhook> που συνδέεται στο νέο κόμβο.

**msg** → Μέσω αυτής της εντολής στέλνεται ένα μήνυμα ελέγχου στο κόμβο που βρίσκεται στο path.

**usage : msg path command [args]**

Συγκεκριμένα η εντολή msg κατασκευάζει ένα μήνυμα ελέγχου το οποίο είναι συνδυασμός τόσο από ASCII ορίσματα (αν υπάρχουνε) και το όνομα της εντολής που εκτελεί ο κόμβος στον οποίο στέλνετε το μήνυμα. Ο κόμβος μετατρέπει το ASCII σε δυαδικά(binary) και στέλνει το αποτέλεσμα.

Αντί για την λέξη msg μπορεί να χρησιμοποιηθεί η λέξη cmd η οποία κάνει ακριβώς τα ίδια πράγματα.

---

**name** → Μέσω αυτής της εντολής μπορεί ο χρήστης να αναθέσει ένα όνομα<name> της επιλογής του στο κόμβο που βρίσκεται στην τοποθεσία <path>.

**usage : name <path> <name>**

---

**read** → Μέσω αυτής της εντολής μπορεί να διαβάζονται και εκτελούνται εντολές από το αρχείο που δίνουμε στην εντολή.

**usage : read <filename>**

Αντί για την λέξη read μπορεί να χρησιμοποιηθούν οι λέξεις source ή το ‘κενό’ οι οποίες κάνουν ακριβώς τα ίδια πράγματα.

---

**rmhook** → Μέσω αυτής της εντολής μπορεί ο χρήστης να αποσυνδέσει το άγκιστρο του οποίου το όνομα δίνεται στο <hook> του κόμβου που βρίσκεται στην τοποθεσία <path>.

**usage : rmhook [path] <hook>**

Συγκεκριμένα η εντολή rmhook αναγκάζει το κόμβο στη τοποθεσία <path>(άμα δεν δίνεται τότε εννοείται το ‘κενό’) να αποσυνδεθεί από το άγκιστρο με όνομα <hook>, εφόσον ήταν συνδεδεμένος .

Αντί για την λέξη rmhook μπορεί να χρησιμοποιηθεί η λέξη disconnect η οποία κάνει ακριβώς τα ίδια πράγματα.

---

**show** → Μέσω αυτής της εντολής δίνονται πληροφορίες σχετικά με τον κόμβο στη τοποθεσία <path>.

### **usage : show [-n]<path>**

Αν δοθεί η επιλογή -n τότε παραλείπονται οι πληροφορίες σχετικά με τα άγκιστρα του κόμβου που ζητάμε να δούμε.

Αντί για την λέξη show μπορεί να χρησιμοποιηθούν οι λέξεις inquire ή info οι οποίες κάνουν ακριβώς τα ίδια πράγματα.

---

**shutdown** → Μέσω αυτής της εντολής μπορεί ο χρήστης να τερματίσει ένα κόμβο που βρίσκεται στην τοποθεσία <path>.

### **usage : shutdown <path>**

Αντί για την λέξη shutdown μπορεί να χρησιμοποιηθούν οι λέξεις kill ή rmnode οι οποίες κάνουν ακριβώς τα ίδια πράγματα.

---

**status** → Μέσω αυτής της εντολής μπορεί ο χρήστης να πάρει δεδομένα σχετικά με τον κόμβο που βρίσκεται στη τοποθεσία <path> σε ASCII μορφή ώστε να μπορεί να τα διαβάσει.

### **usage : status <path>**

---

**types** → Μέσω αυτής της εντολής μπορεί ο χρήστης να πάρει δεδομένα σχετικά με τους εγκατεστημένους κόμβους που βρίσκονται στο Netgraph.

### **usage : types**

---

**status** → Μέσω αυτής της εντολής μπορεί ο χρήστης να πάρει δεδομένα σχετικά με τον κόμβο που βρίσκεται στη τοποθεσία <path> σε ASCII μορφή ώστε να μπορεί να τα διαβάσει.

### **usage : status <path>**

---

**write** → Μέσω αυτής της εντολής μπορεί ο χρήστης να στείλει δεδομένα στο άγκιστρο που ονομάζεται <hook>. Τα δεδομένα αυτά μπορεί να περιέχονται σε ένα αρχείο <file>

ή μπορεί να περιγράφονται απευθείας στη γραμμή εντολών παρέχοντας μια ακολουθία bytes <byte ...>.

**usage : write hook <-f file | byte ...>**

Αντί για την λέξη write μπορεί να χρησιμοποιηθεί το γράμμα w το οποίο κάνει ακριβώς το ίδιο πράγμα.

---

**quit** → Μέσω αυτής της εντολής μπορεί ο χρήστης να εξέλθει από το ngctl.

**usage : exit**

Αντί για την λέξη quit μπορεί να χρησιμοποιηθεί η λέξη exit η οποία κάνει ακριβώς τα ίδια πράγματα.

---

### 3.3 Παραδείγματα χρήσης του Netgraph

Στην ενότητα αυτή θα δείξουμε απλά παραδείγματα σχετικά με την χρήση του utility ngctl και των εντολών του αλληλεπιδρώντας με τους κόμβους και τα άγκιστρα.[2]

Θα γίνει περιγραφή βημάτων για τη δημιουργία, την ονοματοδοσία, τη σύνδεση και την παρακολούθηση κόμβων Netgraph αλλά και την συλλογή στατιστικών μέσω του ngctl. Αρχικά με την εντολή show θα εμφανιστεί ο κόμβος ngctl1652, που δημιουργείται κατά την εκκίνηση του εργαλείου. Η διαδικασία θα συνεχιστεί με τη δημιουργία ενός κόμβου τύπου tee, ο οποίος θα συνδεθεί με τον κόμβο ngctl1652 μέσω ενός προσαρμοσμένου άγκιστρου. Στη συνέχεια, θα γίνει ορισμός ονόματος στον κόμβο tee και θα γίνει σύνδεση ενός κόμβου Cisco HDLC, κάνοντας ανταλλαγή δεδομένων μέσα από το δίκτυο Netgraph. Τέλος μέσω της αποστολής ενός μηνύματος θα γίνει η συλλογή στατιστικών για το κόμβο tee και τερματισμός του προγράμματος

Κατά την εκκίνηση του utility ngctl[2] δημιουργείται by default ένας κόμβος τύπου socket, ο οποίος χρησιμοποιείται για την αλληλεπίδραση με άλλους κόμβους του συστήματος, μέσω της εντολής show . Παρατηρείτε ότι ο κόμβος αυτός έχει όνομα

ngctl1652(το ngctl έδωσε τυχαία αυτό το όνομα στο κόμβο) και ID 45 με μηδενικές συνδέσεις αφού num αγκιστρων είναι μηδέν.

### Πίνακας 3.3. Οδηγός εντολών ngctl μέρος 1

```
+ show .
Name: ngctl652      Type: socket      ID: 00000045  Num hooks: 0
```

Για τη δημιουργία ενός νέου κόμβου και τη σύνδεσή του με έναν ήδη υπάρχοντα κόμβο, χρησιμοποιείται η εντολή mkpeer. Σε αυτή την περίπτωση, δημιουργείται ένας κόμβος τύπου tee, ο οποίος συνδέεται με έναν υπάρχοντα κόμβο μέσω του αγκίστρου myhook. Η σύνδεση πραγματοποιείται χρησιμοποιώντας το right άγκιστρο του νέου κόμβου τύπου tee.

### Πίνακας 3.4. Οδηγός εντολών ngctl μέρος 2

```
+ help mkpeer
Usage:  mkpeer [path] <type> <hook> <peerhook>
Summary: Create and connect a new node to the node at "path"
Description:
The mkpeer command atomically creates a new node of type "type"
and connects it to the node at "path". The hooks used for the
connection are "hook" on the original node and "peerhook" on
the new node. If "path" is omitted then "." is assumed.
+ mkpeer . tee myhook right
+ show .
Name: ngctl652      Type: socket      ID: 00000045  Num hooks: 1
Local hook    Peer name    Peer type    Peer ID    Peer hook
-----        -----        -----        -----
myhook        <unnamed>     tee         00000046    right
```

Ακόμη παρατηρείται ότι ο κόμβος δεν διαθέτει όνομα. Ωστόσο, θα μπορούσε πάντα να γίνεται αναφορά σε αυτόν χρησιμοποιώντας τη σχετική διεύθυνση ..myhook. Παρομοίως, θα μπορούσε πάντα να γίνεται αναφορά σε αυτόν με την απόλυτη διεύθυνση (46).

### Πίνακας 3.5. Οδηγός εντολών ngctl μέρος 3

```
+ show .:myhook
Name: <unnamed>      Type: tee          ID: 00000046  Num hooks: 1
Local hook    Peer name    Peer type   Peer ID    Peer hook
-----        -----        -----       -----
right         ngctl652     socket      00000045     myhook
```

Έπειτα, ο κόμβος που βρίσκεται στη σχετική διεύθυνση .:myhook ονομάζεται κανονικά ως mytee και επαληθεύεται ότι το όνομα έχει δοθεί μέσω της εντολής show .

### Πίνακας 3.6. Οδηγός εντολών ngctl μέρος 4

```
+ name .:myhook mytee
+ show mytee:
Name: mytee      Type: tee          ID: 00000046  Num hooks: 1
Local hook    Peer name    Peer type   Peer ID    Peer hook
-----        -----        -----       -----
right         ngctl652     socket      00000045     myhook
```

Συνδέεται ένας κόμβος τύπου Cisco HDLC στην άλλη πλευρά του κόμβου tee, χρησιμοποιώντας το left άγκιστρο του κόμβου tee και επαληθεύεται μέσω της εντολής show αν έχει πραγματοποιηθεί η σύνδεση. Παρατηρείται ότι ο κόμβος Cisco συνδέεται μέσω του downstream αγκίστρου του στο left άγκιστρο του mytee. Ο κόμβος Cisco παράγει περιοδικά πακέτα keep-alive κάθε 10 δευτερόλεπτα. Αυτά τα πακέτα διασχίζουν τον κόμβο tee (από αριστερά προς δεξιά) και καταλήγουν στο myhook.

### Πίνακας 3.7. Οδηγός εντολών ngctl μέρος 5

```
+ mkpeer mytee: cisco left downstream
+ show mytee:
Name: mytee      Type: tee          ID: 00000046  Num hooks: 2
Local hook    Peer name    Peer type   Peer ID    Peer hook
-----        -----        -----       -----
left          <unnamed>    cisco      00000047     downstream
```

```

right      ngctl652      socket      00000045      myhook
+
Rec'd data packet on hook "myhook":
0000: 8f 00 80 35 00 00 00 02 00 00 00 00 00 00 00 00 ...5.....
0010: ff ff 00 20 8c 08 40 00          ... ..@.
+
Rec'd data packet on hook "myhook":
0000: 8f 00 80 35 00 00 00 02 00 00 00 00 00 00 00 00 ...5.....
0010: ff ff 00 20 b3 18 00 17

```

Αν θέλετε να δείτε όλους τους κόμβους που υπάρχουν αυτή τη στιγμή στο σύστημα, εκτελέσετε την εντολή list.

### Πίνακας 3.8. Οδηγός εντολών ngctl μέρος 6

```

+ list
There are 5 total nodes:
Name: <unnamed>      Type: cisco      ID: 00000047  Num hooks: 1
Name: mytee            Type: tee       ID: 00000046  Num hooks: 2
Name: ngctl652          Type: socket    ID: 00000045  Num hooks: 1
Name: fxp1              Type: ether     ID: 00000002  Num hooks: 0
Name: fxp0              Type: ether     ID: 00000001  Num hooks: 0
+
Rec'd data packet on hook "myhook":
0000: 8f 00 80 35 00 00 00 02 00 00 00 00 00 00 00 00 ...5.....
0010: ff ff 00 22 4d 40 40 00

```

Τερματίζεται ο κόμβος Cisco και κατά συνέπεια, τερματίζεται η μετάδοση δεδομένων από τον κόμβο που είναι συνδεδεμένος με το αριστερό άγκιστρο του κόμβου mytee.

### Πίνακας 3.9. Οδηγός εντολών ngctl μέρος 7

```

+ shutdown mytee:left
+ show mytee:
Name: mytee            Type: tee       ID: 00000046  Num hooks: 1
Local hook   Peer name   Peer type   Peer ID   Peer hook
-----      -----      -----      -----
right       ngctl652      socket      00000045      myhook

```

Όταν θέλετε να λάβετε τα στατιστικά ενός κόμβου (εδώ ο κόμβος mytee), στέλνετε ένα μήνυμα ελέγχου και λαμβάνετε πίσω μια άμεση απάντηση, όπου τόσο η απάντηση όσο και η εντολή μετατρέπονται σε/από ASCII αυτόματα για εσάς από το ngctl για να είναι κατανοητές. Βλέπετε ότι τρία πλαίσια (και 72 οκτάδες) πέρασαν από τον κόμβο tee από αριστερά προς τα δεξιά. Κάθε πλαίσιο διπλασιάστηκε και πέρασε έξω από το άγκιστρο left2right (αλλά επειδή αυτό το άγκιστρο δεν ήταν συνδεδεμένο, τα διπλά αυτά πλαίσια χάθηκαν).

#### Πίνακας 3.10. Οδηγός εντολών ngctl μέρος 8

```
+ help msg
Usage:   msg path command [args ... ]
Aliases: cmd
Summary: Send a netgraph control message to the node at "path"
Description:
The msg command constructs a netgraph control message from the
command name and ASCII arguments (if any) and sends that
message to the node. It does this by first asking the node to
convert the ASCII message into binary format, and re-sending the
result. The typecookie used for the message is assumed to be
the typecookie corresponding to the target node's type.
+ msg mytee: getstats
Rec'd response "getstats" (1) from "mytee:":
Args:   { right={ outOctets=72 outFrames=3 } left={ inOctets=72 inFrames=3 }
        left2right={ outOctets=72 outFrames=3 } }
```

Τέλος αφού έγινε μια μικρή περιήγηση στο utility ngctl με την εντολή quit εξέρχεστε από αυτό.

#### Πίνακας 3.11. Οδηγός εντολών ngctl μέρος 9

```
+ quit
```

Περισσότερα παραδείγματα χρήσης του ngctl θα βρείτε στο Παράρτημα 2.



# **Κεφάλαιο 4.**

# **Δημιουργία**

# **κόμβου στο**

# **Netgraph**

## **4.1 Περιγραφή δομής κόμβου**

Για να υλοποιήσει κάποιος ένα προσαρμοσμένο (custom) τύπο κόμβου στο Netgraph[2-5] χρειάζεται να κατανοήσει ότι ένας προσαρμοσμένος κόμβος είναι ουσιαστικά μια μονάδα(module) και ότι κάθε μονάδα στο Netgraph δημιουργείται από δύο βασικά αρχεία. Δηλαδή απαρτίζεται από ένα αρχείο κεφαλίδας(header file, example.h) και ένα αρχείο πηγαίου κώδικα( c file,example.c) όπου μετά θα πρέπει να μεταγλωττιστούν για να μετατραπούν στην μορφή του Netgraph, δηλαδή ng\_example.ko. Άρα για την υλοποίηση του «προσαρμοσμένου» τύπου κόμβου στο Netgraph θα πρέπει στο αρχείο .c που θα δημιουργηθεί να οριστεί το struct ng\_type που μέσα σε αυτό θα υπάρχουν όλες οι λειτουργίες που θέλει ο χρήστης να γίνονται, δηλαδή τόσο τους ορισμούς των συναρτήσεων(functions) όσο και τις σχετικές συναρτήσεις επιστροφής(callback functions).

Πίνακας 4.1. Ορισμός του ng\_type

### Ορισμός του ng\_type

```
struct ng_type typestruct {
    u_int32_t      version;          /* must equal NG_VERSION */
    const char     *name;            /* Unique type name */
    modeventhand_t mod_event;       /* Module event handler (optional) */
    ng_constructor_t *constructor;  /* Node constructor */
    ng_rcvmsg_t     *rcvmsg;          /* control messages come here */
    ng_shutdown_t   *shutdown;        /* reset, and free resources */
    ng_newhook_t    *newhook;         /* first notification of new hook */
    ng_findhook_t   *findhook;        /* only if you have lots of hooks */
    ng_connect_t    *connect;         /* final notification of new hook */
    ng_rcvdata_t    *rcvdata;         /* date comes here */
    ng_rcvdata_t    *rcvdataq;        /* or here if being queued */
    ng_disconnect_t *disconnect;    /* notify on disconnect */
    const struct    ng_cmdlist *cmdlist; /* commands we can convert */
    /* R/W data private to the base netgraph code DON'T TOUCH! */
    LIST_ENTRY(ng_type) types;        /* linked list of all types */
    int             refs;            /* number of instances */
};
```

Πίνακας 4.2. Παράδειγμα ορισμού ng\_example

### Παράδειγμα δημιουργίας ng\_type του κόμβου ng\_example

```
static struct ng_type ng_example_typestruct = {
    .version =      NG_ABI_VERSION,
    .name =        NG_EXAMPLE_NODE_TYPE,
    .constructor = ng_example_constructor,
    .rcvmsg =       ng_example_rcvmsg,
    .close =       ng_example_close,
    .shutdown =     ng_example_shutdown,
    .newhook =      ng_example_newhook,
    .rcvdata =      ng_example_rcvdata,
    .disconnect =   ng_example_disconnect,
    .cmdlist =      ng_example_cmds,
};
```

Αφού ολοκληρωθεί η διαδικασία ορισμού του `ng_type`, θα πρέπει μετά να γίνει η σύνδεση του με την μακροεντολή `NETGRAPH_INIT()`. Μέσω αυτής της μακροεντολής εξασφαλίζεται η σωστή αρχικοποίηση του τύπου κόμβου ώστε να είναι εφικτή η σύνδεση του με το σύστημα.

Το πρώτο όρισμα θα πρέπει να είναι το όνομα του τύπου κόμβου (π.χ. `echo`) και το δεύτερο ένας δείκτης στην δομή(`struct`) που ορίστηκε παραπάνω.

Πίνακας 4.3. Ορισμός του `NETGRAPH_INIT` για κόμβο με τύπο `ng_type`

**Ορισμός του `ng_type` του `NETGRAPH_INIT()`.**

```
NETGRAPH_INIT(type_name, typestruct_ptr);
```

Πίνακας 4.4. Παράδειγμα ορισμού `NETGRAPH_INIT` για τον κόμβο με τύπο `example`

**Παράδειγμα σύνδεσης του `ng_example` με το `NETGRAPH_INIT()`.**

```
NETGRAPH_INIT(example, &ng_example_typestruct);
```

## 4.2 Διασύνδεση με σύστημα Netgraph

Για την διασύνδεση του τύπου κόμβου που αναφέρθηκε στο 4.1 θα πρέπει να περιέχονται ορισμένα χαρακτηριστικά τόσο στο αρχείο `.h` που θα δημιουργηθεί όσο και στο αρχείο `.c`. Δηλαδή να ακολουθηθεί μια δομή όπως έχουν και οι ήδη υπάρχων μονάδες(modules) στο Netgraph. Μελετώντας τις ήδη υπάρχων μονάδες η υλοποίηση των περιεχομένων του αρχείου `.h` ενδεικτικά μπορεί να ακολουθεί τη παρακάτω υψηλού επιπέδου(high level) σχεδίαση.

Πίνακας 4.5. Ενδεικτικά περιεχόμενα αρχείου .h για δημιουργία κόμβου μέρος 1

### Στοιχεία του .h file

Οδηγίες που εξασφαλίζουν να αποτραπεί η επαναληπτική εισαγωγή του αρχείου.

'Όπως για παράδειγμα → #ifndef \_NETGRAPH\_NG\_EXAMPLE\_H\_  
#define \_NETGRAPH\_NG\_EXAMPLE\_H\_

Ορισμός τύπου κόμβου και τύπου cookie

'Όπως για παράδειγμα → #define NG\_EXAMPLE\_NODE\_TYPE "example"  
#define NGM\_EXAMPLE\_COOKIE 916107047

Ορισμός αγκίστρων κόμβου

'Όπως για παράδειγμα → #define NG\_EXAMPLE\_HOOK\_RIGHT "right"  
#define NG\_EXAMPLE\_HOOK\_LEFT "left"

Δομή( struct) στατιστικών για κάθε άγκιστρο

'Όπως για παράδειγμα → # struct ng\_example\_hookstat {  
    u\_int64\_t inOctets;  
    u\_int64\_t inFrames;  
};

### Στοιχεία του .h file(συνέχεια..)

Μάκρος(Macros) που προετοιμάζουν τις πληροφορίες που περιέχονται στη δομή αποθήκευσης των στατιστικών για κάθε άγκιστρο

```
'Όπως για παράδειγμα → #define NG_EXAMPLE_HOOKSTAT_INFO {  
    { "inOctets",      &ng_parse_uint64_type }  
    { "inFrames",      &ng_parse_uint64_type }  
}
```

Δομή(struct) αποθήκευσης των συνολικών στατιστικών για τον κόμβο

```
'Όπως για παράδειγμα → struct ng_example_stats {  
    struct ng_tee_hookstat right;  
    struct ng_tee_hookstat left;  
};
```

Μάκρος(Macros) που προετοιμάζουν τις πληροφορίες που περιέχονται στη δομή αποθήκευσης των συνολικών στατιστικών για τον κόμβο

```
'Όπως για παράδειγμα → #define NG_EXAMPLE_STATS_INFO {  
    { "right",  (hstype) }  
    { "left",   (hstype) },  
}
```

Διάφορες εντολές για την αλληλεπίδραση του κόμβου με το σύστημα (γενικά μηνύματα ελέγχου )

```
'Όπως για παράδειγμα → enum {  
    NGM_TEE_GET_STATS = 1,      /* get stats */  
    NGM_TEE_CLR_STATS,        /* clear stats */  
};
```

Ακολουθώντας την ιδιά μορφή υψηλής σχεδίασης, παρακάτω θα παρουσιαστεί μια ενδεικτική υλοποίηση των περιεχομένων του αρχείου.c

## Στοιχεία του .c file

### Εισαγωγή Κεφαλίδων

```
'Όπως για παράδειγμα →#include <netgraph/ng_message.h>
                           #include <netgraph/netgraph.h>
                           #include <netgraph/ng_parse.h>
```

### Δομή( struct) για την αναπαράσταση των πληροφοριών ανά άγκιστρο

```
'Όπως για παράδειγμα →#struct hookinfo {
                           hook_p           hook;
                           struct hookinfo *dest, *dup;
                           struct ng_example_hookstat   stats;
};
```

### Δομή( struct) για την αναπαράσταση των πληροφοριών ανά κόμβο

```
'Όπως για παράδειγμα →struct privdata {
                           struct hookinfo      right;
                           struct hookinfo      left;
};
```

### Ορισμοί μεθόδων που είναι διαθέσιμες για χρήση μέσα στο αρχείο

```
'Όπως για παράδειγμα → static ng_constructor_t    ng_example_constructor;
                           static ng_rcvmsg_t       ng_example_rcvmsg;
                           static ng_close_t         ng_example_close;
```

### Τύπος για την ανάλυση πληροφοριών(parse type) για την δομή που πρέπει να έχουν τα στατιστικά των αγκίστρων

```
'Όπως για παράδειγμα → static const struct ng_parse_struct_field
ng_example_hookstat_type_fields[] =          NG_EXAMPLE_HOOKSTAT_INFO;
```

### Τύπος για την ανάλυση πληροφοριών(parse type) για την δομή που πρέπει να έχουν τα στατιστικά των κόμβων

```
'Όπως για παράδειγμα → static const struct ng_parse_struct_field ng_example
                           _stats_type_fields[]
                           = NG_EXAMPLE_STATS_INFO(&ng_example_hookstat_type);
```

### Στοιχεία του .c file(συνέχεια...)

Λίστα εντολών για την εισαγωγή ορισμάτων(arguments) ASCII

```
'Όπως για παράδειγμα → static const struct ng_cmdlist ng_example_cmds[] = {  
    {  
        NGM_EXAMPLE_COOKIE,  
        NGM_EXAMPLE_GET_STATS,  
        "getstats",  
        NULL,  
        &ng_example_stats_type  
    }  
}
```

Ορισμός τύπου κόμβου ng\_type όπως ορίστηκε στην ενότητα 4.1

```
'Όπως για παράδειγμα → static struct ng_type ng_example_typestruct = {  
    .version =      NG_ABI_VERSION,  
    .name =        NG_EXAMPLE_NODE_TYPE,  
    .constructor = ng_example_constructor,  
    .rcvmsg =       ng_example_rcvmsg,  
};  
NETGRAPH_INIT(example, &ng_example_typestruct);
```

Λειτουργίες μεθόδων κόμβου (δημιουργία κόμβου, προσθήκη αγκίστρου κλπ)

```
'Όπως για παράδειγμα → ng_example_constructor(node_p node){};  
ng_example_newhook(node_p node, hook_p hook,  
const char *name){};  
ng_tee_rcvmsg(node_p node, item_p item, hook_p  
lasthook){};
```

Βεβαίως αυτή η δομή που παρουσιάστηκε παραπάνω τόσο στο αρχείο.c, όσο και στο αρχείο .h αποτελεί ένα σκελετό που μπορεί κάποιος να δουλέψει πάνω σε αυτόν. Συγκεκριμένα κάποιος μπορεί να πειραματιστεί σε μεγάλο εύρος προσθέτοντας ή αφαιρώντας λειτουργίες του κόμβου όπως για παράδειγμα μεθόδους. Ακόμη μπορεί να πειραματιστεί με τις δομές του Netgraph που υπάρχουν στο κόμβο όπως για παράδειγμα να προσθέσει λειτουργικότητα σε κάποια άγκιστρα.

## 4.3 Εγκατάσταση και χρήση νέου κόμβου

Για να γίνει εγκατάσταση του νέου κόμβου (π.χ. κόμβος με όνομα uoi) απαιτούνται τρία βασικά αρχεία

- `ng_uoi.c`: Περιέχει τον πηγαίο κώδικα του κόμβου με τις λειτουργίες, συναρτήσεις και μεθόδους.
- `ng_uoi.h`: Περιέχει κεφαλίδες του κόμβου για την διασύνδεση με το υπόλοιπο σύστημα όπως δομές και μακροεντολές σχετικές με το χειρισμό μηνυμάτων.
- `Makefile`: Μέσω αυτού δημιουργείται ο νέος κόμβος με την εντολή `make` στο κατάλογο(directory) που βρίσκεται το `Makefile`.

Αρχικά θα πρέπει ο χρήστης να κατεβάσει τον κατάλογο `src` που περιέχονται τα αρχεία και οι υποκαταλόγοι με τους πηγαίους κώδικες του FreeBSD ,καθώς δεν περιέχονται στην αρχική εγκατάσταση. Αυτό γίνεται με την παρακάτω εντολή που παρουσιάζεται στο πίνακα 4.9.

Πίνακας 4.9. Εντολές για εγκατάσταση `src` καταλόγου

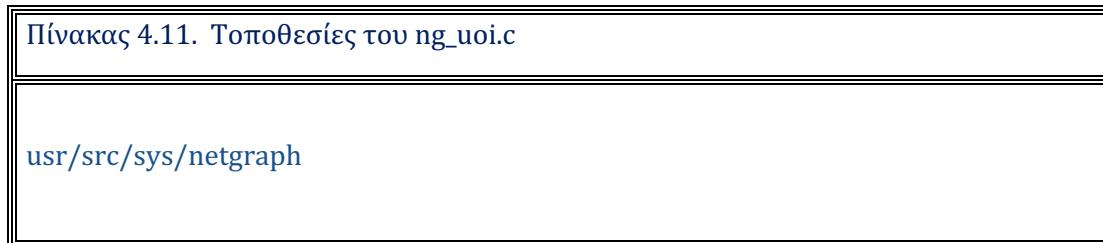
```
pkg install git  
git clone -depth 1 -b releng/13.2 https://git.freebsd.org/src.git /usr/src/sys
```

Συνεχίζοντας ο χρήστης θα πρέπει να τοποθετήσει στα παρακάτω μονοπάτια(paths), που παρουσιάζονται στο πίνακα 4.10 το αρχείο `ng_uoi.h`.

Πίνακας 4.10. Τοποθεσίες του `ng_uoi.h`

```
usr/src/include  
usr/src/sys/Netgraph
```

Έπειτα ο χρήστης θα πρέπει να τοποθετήσει στο παρακάτω μονοπάτι(path), που παρουσιάζονται στο πίνακα 4.11 το αρχείο ng\_uoi.c.



Ακόμη ο χρήστης οφείλει να δημιουργήσει ένα κατάλογο με όνομα uoi στην τοποθεσία usr/src/sys/modules/netgraph. Με την εισοδό του στο κατάλογο uoi θα δημιουργήσει το αρμόδιο Makefile. Έπειτα ο χρήστης πρέπει να εκτελέσει την εντολή make στην τοποθεσία usr/src/sys/modules/netgraph/uoi ώστε να δημιουργήσει το εκτελέσιμό του κόμβου [67].

The image shows a code editor window titled "Makefile" with the file path "/usr/src/sys/sys/modules/n...". The code in the editor is as follows:

```
1
2 KMOD= ng_uoi
3 SRCS= ng_uoi.c
4
5 CFLAGS = -DVIMAGE
6
7 .include <bsd.kmod.mk>
8
```

Εικόνα 4.1. Περιεχόμενα αρχείου Makefile

Στο parent κατάλογο δηλαδή στην τοποθεσία usr/src/sys/modules/netgraph υπάρχει άλλο ένα Makefile όπου έχει μια λίστα με όλους τους κόμβους του Netgraph. Ο χρήστης θα πρέπει να προσθέσει το όνομα του τύπου κόμβου δηλαδή στην περίπτωσή που εξετάζουμε το όνομα uoi(γραμμή 12 του αρχείου)και ακολούθως με την εντολή make να γίνουν τα απαραίτητα εκτελέσιμα.

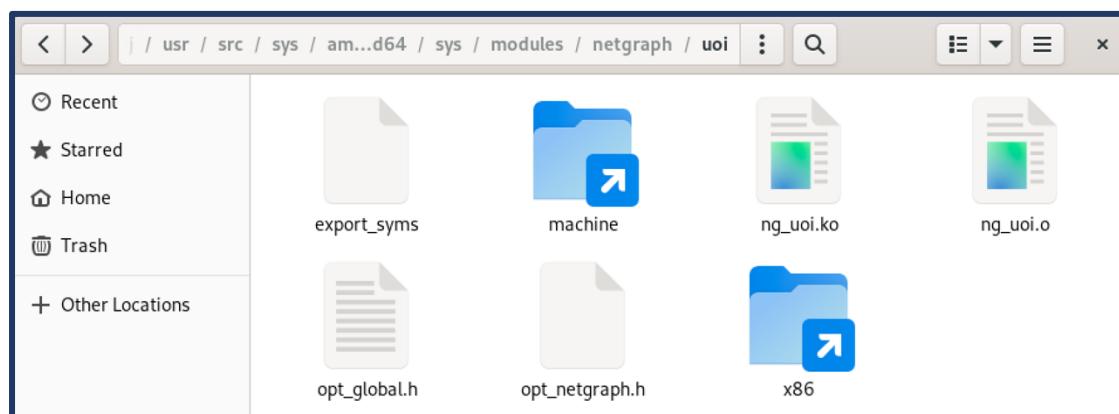
```

1 # $Whistle: Makefile,v 1.5 1999/01/24 06:48:37 archie Exp $
2 # $FreeBSD$
3
4 SYSDIR?=${SRCTOP}/sys
5 .include "${SYSDIR}/conf/kern.opts.mk"
6
7 SUBDIR=    async \
8     atm \
9     atmllc \
10    ${_bluetooth} \
11    bpf \
12    uoi \
13    bridge \
14    car \
15    checksum \
16    cisco \
17    deflate \
18    device \
19    echo \
20    eiface \
21    etf \
22    ether \
23    ether_echo \
24    frame_relay \
25    gif \
26    gif_demux \
27    hole \

```

Εικόνα 4.2. Περιεχόμενα αρχείου Makefile του “parent” κατάλογο

Έπειτα ο χρήστης θα πλοηγηθεί στην τοποθεσία `usr/obj/usr/src/sys/amd64.amd64/sys/modules/netgraph/uoi` ώστε να πάρει το αρχείο `ng_uoi.ko` και να το αντιγράψει στην τοποθεσία `boot/kernel`. Δηλαδή εκεί που βρίσκονται όλα τα υπόλοιπα kernel objects ώστε να μπορεί να είναι αναγνωρίσιμα από το σύστημα και να μπορούν να φορτωθούν στον πυρήνα(kernel).



Εικόνα 4.3. Τοποθεσία δημιουργίας του ζητούμενου αρχείου .ko

Ακόμη ο χρήστης μπορεί να φορτώσει στο πυρήνα(kernel) το κόμβο ώστε να μπορέσει αρχικά να επαληθεύσει την ορθή μετατροπή αλλά και λειτουργία. Μεταγενέστερα να μπορεί να το χρησιμοποιήσει στο δίκτυο του.

```
root@ ~ # kldload ng_uoi.ko
root@ ~ # kldstat
Id Refs Address          Size  Name
1   22 0xfffffff80200000 1f3e2d0 kernel
2   1 0xfffffff8213f000  a4a0 cryptodev.ko
3   1 0xfffffff8214b000  59dfa8 zfs.ko
4   1 0xfffffff82b18000  3218 intpm.ko
5   1 0xfffffff82b1c000  2180 smbus.ko
6   1 0xfffffff82b1f000  2138 ng_uoi.ko
7   1 0xfffffff82b22000  aac8 netgraph.ko
```

Εικόνα 4.4. Επιβεβαίωση της φόρτωσής του κόμβου στο kernel

## 4.4 Αναχαίτηση ροής δεδομένων στα επίπεδα της στοίβας πρωτοκόλλων TCP/IP

Με τον φράση «αναχαίτηση ροής δεδομένων» θεωρείτε ουσιαστικά η παρεμβολή του κόμβου με σκοπό την λήψη ή την επεξεργασία πληροφορίων σε ένα επίπεδο. Δηλαδή ο στόχος είναι να γίνει ένα «βραχυκύκλωμα» μεταξύ του κόμβου που θέλει ο χρήστης να παρακολουθήσει και του κόμβου παρεμβολής που τοποθετεί ο χρήστης. Στα παρακάτω παραδείγματα παρεμβολής χρησιμοποιείται ο κόμβος tee με σκοπό την καταγραφή στατιστικών. Πάραυτα ο χρήστης θα μπορούσε να τοποθετήσει αντί του κόμβου tee τον κόμβο της επιλογής του, ανάλογα το σκοπό και την λειτουργία που θέλει να πετύχει.

Παρακάτω παρουσιάζονται ενδεικτικά σχηματικά σχετικά με την αναχαίτηση της ροής δεδομένων στα επίπεδα.

Στο επίπεδο συνδέσμου η αναχαίτηση της ροής δεδομένων γίνεται με τον παρακάτω τρόπο δηλαδή με την χρήση του κόμβου παρεμβολής και του ether κόμβου.

The diagram illustrates the interception of network traffic at the Network Access Layer. A blue arrow labeled "Network Access Layer" points to the right. Above it, a yellow rounded rectangle labeled "Ether Node" contains a yellow curved arrow pointing left. Above the Ether Node is a blue rounded rectangle labeled "Flow Interception Node" containing a blue curved arrow pointing right. This visualizes how traffic is monitored before it reaches the physical layer.

Εικόνα 4.5. Αναπαράσταση αναχαίτισης της ροής δεδομένων στο επίπεδο συνδέσμου

Στο επίπεδο δικτύου η αναχαίτηση της ροής δεδομένων γίνεται με τον παρακάτω τρόπο δηλαδή με την χρήση του κόμβου παρεμβολής και του ipfw κόμβου.

The diagram illustrates the interception of network traffic at the Network Layer. A blue arrow labeled "Network Layer" points to the right. Above it, a yellow rounded rectangle labeled "Ipfw Node" contains a blue curved arrow pointing right. Above the Ipfw Node is a blue rounded rectangle labeled "Flow Interception Node" containing a yellow curved arrow pointing left. This visualizes how traffic is monitored at the network layer using both a flow interception node and an Ipfw node.

Εικόνα 4.6. Αναπαράσταση αναχαίτισης της ροής δεδομένων στο επίπεδο δικτύου

Στο επίπεδο εφαρμογής η αναχαίτηση της ροής δεδομένων γίνεται με τον παρακάτω τρόπο δηλαδή με την χρήση του κόμβου παρεμβολής και του socket κόμβου.

The diagram illustrates the interception of network traffic at the Application Layer. A blue arrow labeled "Application Layer" points to the right. Above it, a yellow rounded rectangle labeled "Socket Node" contains a blue curved arrow pointing right. Above the Socket Node is a blue rounded rectangle labeled "Flow Interception Node" containing a yellow curved arrow pointing left. This visualizes how traffic is monitored at the application layer using both a flow interception node and a socket node.

Εικόνα 4.7. Αναπαράσταση αναχαίτισης της ροής δεδομένων στο επίπεδο εφαρμογής

#### 4.4.1 Αναχαίτηση ροής δεδομένων στο επίπεδο συνδέσμου

Η αναχαίτηση ροής δεδομένων στο επίπεδο συνδέσμου μπορεί να γίνει με την σύνδεση του κόμβου παρεμβολής και του ng\_ether. Σαν κόμβο παρεμβολής χρησιμοποιείται σε

αυτό το παράδειγμα ο κόμβος tee. Ουσιαστικά ο κόμβος tee παρεμβαίνει στο κόμβο ether συνδέοντας τα άγκιστρά του right και left με τα άγκιστρά lower και upper αντίστοιχα. Έτσι παίρνει τα πακέτα που εισέρχονται και εξέρχονται από το κόμβο ether. Παρακάτω παρουσιάζεται ένα ενδεικτικό σενάριο παρεμβολής στο επίπεδο συνδέσμου.

**Πίνακας 4.11. Οδηγός εντολών για παρεμβολή στο επίπεδο συνδέσμου**

1. + <b>kldload</b> ng_ether.ko	1) Φόρτωση της μονάδας ether στο πυρήνα
2. + <b>kldload</b> ng_tee.ko	2) Φόρτωση της μονάδας tee στο πυρήνα
3. + <b>ngctl mkpeer</b> em0: tee lower right	3) Δημιουργία κόμβου τύπου tee και σύνδεση του lower αγκίστρου του κόμβου ether με το right αγκίστρο του κόμβου tee
4. + <b>ngctl name</b> em0: lower TEE	4) Ονοματοδοσία του συνδεδεμένου κόμβου tee με όνομα TEE
5. + <b>ngctl connect</b> TEE: em0: left upper	5) Σύνδεση του left αγκίστρου του κόμβου tee με το άγκιστρο upper του κόμβου ether
6. + <b>ngctl msg</b> TEE: getstats	6-7) Έλεγχος για την ροή δεδομένων μέσα από την αναπαράσταση των στατιστικών
7. Rec'd response "getstats" (1) from "[4]": Args: { right={ outOctets=176 outFrames=2 } left={ inOctets=176 inFrames=2 } }	

#### 4.4.2 Αναχαίτηση ροής δεδομένων στο επίπεδο δικτύου

Η επέμβαση στο επίπεδο δικτύου μπορεί να γίνει με την χρήση των κόμβων ng\_tee και ng\_ipfw. Ουσιαστικά ο κόμβος tee παρεμβαίνει στο κόμβο ipfw συνδέοντας τα άγκιστρά του right και left με τα άγκιστρά του ipfw. Έτσι παίρνει τα πακέτα που εισέρχονται και εξέρχονται από το κόμβο ipfw. Να σημειωθεί ότι τα άγκιστρα του ipfw δέχονται όνομα που αποτελείται από αριθμούς όπως (π.χ. άγκιστρο 50). Ακόμη για την λειτουργία του κόμβου, δηλαδή για την είσοδο και έξοδο πακέτων δικτύου θα πρέπει να γίνει μια προεργασία με το ήδη υπάρχων ipfw firewall του FreeBSD. Συγκεκριμένα πρέπει να γίνει ενεργοποίηση του ipfw firewall και αυτό επιτυγχάνεται με την προσθήκη των εντολών `firewall_enable="YES"` και `firewall_type="open"` στο αρχείο `/etc/rc.conf`.

Παρακάτω παρουσιάζεται ένα ενδεικτικό σενάριο παρεμβολής στο επίπεδο δικτύου.

**Πίνακας 4.12. Οδηγός εντολών για παρεμβολή στο επίπεδο δικτύου**

<pre> 1. +kldload ng_ipfw.ko 2. +kldload ng_tee.ko 3. +ngctl mkpeer ipfw: tee 50 left 4. +ngctl name ipfw: 50 TEE 5. +ngctl connect TEE: ipfw: right 51 6. +ipfw add 200 netgraph 51 all from any to any 7. +ipfw add 150 netgraph 50 all from any to any 8. +sysctl net.inet.ip.fw.one_pass=0 9. +ngctl msg TEE: getstats  Rec'd response "getstats" (1) from "[5]": Args: { right={ inOctets=3597 inFrames=36 outOctets=4088 outFrames=40 } left={ inOctets=4088 inFrames=40 outOctets=3597 outFrames=36 }} </pre>	<p>1) Φόρτωση της μονάδας ipfw στο πυρήνα      2) Φόρτωση της μονάδας tee στο πυρήνα      3) Δημιουργία κόμβου τύπου tee και σύνδεση του 50 αγκίστρου του κόμβου ipfw με το left άγκιστρο του κόμβου tee      4) Ονοματοδοσία του συνδεδεμένου κόμβου tee με όνομα TEE      5) Σύνδεση του right αγκίστρου του κόμβου tee με το άγκιστρο 51 του κόμβου ether      6-7) Προσθήκη κανόνων(150 και 200) στο firewall του FreeBSD όπου θα παραπέμπει τα πακέτα στο υποσύστημα Netgraph και συγκεκριμένα στα άγκιστρα 50 και 51.      8) Τα πακέτα που έχουν περάσει από το ipfw και στη συνέχεια επεξεργαστούν από το Netgraph θα περάσουν ξανά από το ipfw για περαιτέρω επεξεργασία από τους κανόνες του ipfw.      9) Έλεγχος για την ροή δεδομένων μέσα από την αναπαράσταση των στατιστικών</p>
--	--

#### **4.4.3 Αναχαίτηση ροής δεδομένων στο επίπεδο εφαρμογής**

Η επέμβαση στο επίπεδο εφαρμογών μπορεί να γίνει με την χρήση των κόμβων ng\_tee,ng\_socket(σε RAW mode) και ng\_ether. Ουσιαστικά ο κόμβος tee παρεμβαίνει μεταξύ των κόμβων ng\_ether και ng\_socket. Έτσι με το right και left άγκιστρο συνδέεται στο ether κόμβο στα upper και lower άγκιστρα αντίστοιχα, ενώ με το right2left και left2right άγκιστρα συνδέεται στο ng\_socket. Επομένως στα right2left και left2right άγκιστρα ύστον τα πακέτα εισόδου και εξόδου της εφαρμογής και μπορεί να γίνει ο ανάλογος χειρισμός από το χρήστη. Παρακάτω παρουσιάζεται ένα ενδεικτικό σενάριο παρεμβολής στο επίπεδο εφαρμογών στην περίπτωση που θέλουμε πακέτα δεδομένων σε RAW mode.

**Πίνακας 4.13. Οδηγός εντολών για παρεμβολή στο επίπεδο εφαρμογών**

<pre> 1. +kldload ng_ether.ko </pre>	<p>1) Φόρτωση της μονάδας ether στο πυρήνα</p>
--------------------------------------	--

<pre> 2. +kldload ng_tee.ko 3. +ngctl mkpeer em0: tee lower right 4. +ngctl name em0: lower TEE 5. +ngctl connect TEE: em0: left upper 6. +nghook -an TEE: left2right &gt;/dev/null &amp; 7. +ngctl name TEE: left2right SOCKET 8. +ngctl connect TEE: SOCKET: right2left hook1 9. + ngctl msg TEE: getstats 10. Rec'd response "getstats" (1) from "[4]": 11. Args: { {}} </pre>	<p>2) Φόρτωση της μονάδας tee στο πυρήνα  3) Δημιουργία κόμβου τύπου tee και σύνδεση του lower αγκίστρου του κόμβου ether με το right αγκίστρο του κόμβου tee  4) Ονοματοδοσία του συνδεδεμένου κόμβου tee με όνομα TEE  5) Σύνδεση του left αγκίστρου του κόμβου tee με το αγκίστρο upper του κόμβου ether  6) Παρακολούθηση της κίνηση των δεδομένων μέσω του κόμβου TEE από το αγκίστρο left2right και στέλνει τα δεδομένα στο /dev/null, δηλαδή απορρίπτει την έξοδο. Επιπλέον δεν εκτυπώνει τα δεδομένα καθώς χρησιμοποιούνται οι επιλογές -an στην εντολή  7) Ονοματοδοσία του συνδεδεμένου κόμβου socket με όνομα SOCKET  8) Σύνδεση του right2left αγκίστρου του κόμβου TEE με το αγκίστρο hook1(αυθαίρετο όνομα) του κόμβου SOCKET  9-10-11) Έλεγχος για την ροή δεδομένων μέσα από την αναπαράσταση των στατιστικών, εδώ δεν υπάρχουν καθώς γίνεται απόρριψη δεδομένων και ανακατεύθυνση σε μηδενικό(null) κατάλογο </p>
---	--

Παρατηρούμε ότι δεν βλέπουμε στατιστικά και αυτό συμβαίνει γιατί το αρχείο null που βρίσκεται στο κατάλογο dev δεν έχει δεδομένα ώστε να μπορούμε να έχουμε ροή. Αν αντί για dev/null κατευθύνουμε το socket σε κάποιο άλλο αρχείο τότε θα δούμε δεδομένα και κατά συνέπεια στατιστικά.

## 4.5 Κυκλοφορία και χειρισμός πακέτου ή μηνύματος ελέγχου στο Netgraph

Για να κατανοήσει κανείς πώς γίνεται ο χειρισμός ενός πακέτου δεδομένων στο Netgraph, πρώτα πρέπει να αντιληφθεί τι ακριβώς είναι ένα πακέτο δεδομένων στο FreeBSD. Στο FreeBSD, η δομή που διαχειρίζεται τα δικτυακά πακέτα ονομάζεται `mbuf`. Το `mbuf` περιλαμβάνει όλα τα στοιχεία ενός δικτυακού πακέτου, διασφαλίζοντας ότι μπορούν να μεταφερθούν ή να επεξεργαστούν αποτελεσματικά σε ολόκληρο το FreeBSD και κατεπέκταση το Netgraph. Με την σειρά του το Netgraph έχει ορίσει μεθόδους συναρτήσεις και μακροεντολές και με την συνεργασία τους προσδίδουν στο ήδη δικτυακό περιβάλλον του FreeBSD επιπλέον δικτυακά χαρακτηρίστηκα, αφού δίνουν επιπλέον δυνατότητες χειρισμού του ήδη υπάρχοντος `mbuf`. Συγκεκριμένα σε ένα κόμβο υλοποιούνται μέθοδοι όπου μέσα σε αυτές καλούνται συναρτήσεις ή μακροεντολές για να επιτευχθεί η λειτουργικότητα του κόμβου. Οι πιο σημαντικές μέθοδοι που υλοποιούνται μέσα στο κόμβο είναι η `recvdata` και η `rcvmsg` και οι λειτουργίες που γίνονται μέσα σε αυτές τις μεθόδους είναι καθοριστικής σημασίας για τον κόμβο. Παρακάτω θα αναλυθούν οι λειτουργίες που πρέπει να υλοποιηθούν μέσα στην `recvdata`, ώστε να μπορεί να γίνει σωστά ο χειρισμός του πακέτου.

### 4.5.1 Χειρισμός πακέτου δεδομένων

#### 4.5.1.1. Λήψη και ανάγνωση του πακέτου δεδομένων

Η λήψη και η εισαγωγή του πακέτου δεδομένων σε ένα κόμβο γίνεται μέσω της ικλήσης της μεθόδου `recvdata`. Αυτή η μέθοδος καλείται αυτόματα όταν ένα πακέτο φτάσει στο κόμβο. Επιπλέον για την ορθή λειτουργία της `recvdata` απαιτούνται δυο ορίσματα το `hook_p hook` και το `item_p item`. Συγκεκριμένα το `hook` είναι ένας δείκτης σε ένα άγκιστρο από το οποίο λαμβάνονται δεδομένα στο κόμβο ενώ το `item` είναι ένας δείκτης σε μια δομή `item` που περιέχει τα δεδομένα. Παρακάτω παρουσιάζεται το συντακτικό της μεθόδου

Πίνακας 4.14. Συντακτικό της `NGI_GET_M` μακροεντολής

```

static int rcvdata(hook_p hook, item_p item) {
    /ανάγνωση, τροποποίηση, κλήση συναρτήσεων κλπ./
}

```

Αφού έγινε η λήψη του πακέτου δεδομένων το επόμενο στη σειρά είναι η ανάγνωση του και η εξαγωγή των δεδομένων του. Η πιο συνηθισμένη μακροεντολή για αυτήν την διαδικασία είναι η NGI\_GET\_M. Μέσω αυτής της μακροεντολή μπορεί κανείς να αποκτήσει πρόσβαση στο mbuf που περιέχει τα δεδομένα ενός πακέτου. Για να εκτελεστεί η διαδικασία απαιτούνται δυο ορίσματα το item και το m. Οπού το item είναι το όρισμα που παίρνει από την ίδια την rcvdata και όπου το m είναι ένας δείκτης στο mbuf. Συγκεκριμένα η μακροεντολή εξάγει μέσα από το item το mbuf, άρα μετά το item παύει να περιέχει το mbuf. Παρακάτω παρουσιάζεται το συντακτικό της NGI\_GET\_M μακροεντολής.

**Πίνακας 4.15. Συντακτικό της NGI\_GET\_M μακροεντολής**

```
NGI_GET_M(item, m);
```

Παρόμοια δουλεία με την NGI\_GET\_M κάνει και η μακροεντολή NGI\_M. Η διαφορά τους είναι ότι η NGI\_M δεν εξάγει από το item το mbuf δηλαδή δεν αφήνει το item κενό. Απλά το χρησιμοποιεί παίρνει τα δεδομένα που θέλει και το item συνεχίζει να περιέχει το mbuf. Παρακάτω παρουσιάζεται το συντακτικό της NGI\_M μακροεντολής.

**Πίνακας 4.16. Συντακτικό της NGI\_M μακροεντολής**

```
NGI_GET_M(item);
```

#### 4.5.1.2 Τροποποίηση του πακέτου δεδομένων

Από την στιγμή που ο χρήστης λάβει πρόσβαση στο mbuf τότε μπορεί να το τροποποιήσει ανάλογα με το σκοπό που θέλει να πετύχει. Συγκεκριμένα για την τροποποίηση του πακέτου το Netgraph δεν έχει κάποια συνάρτηση ή μακροεντολή

όπως οι προηγούμενες περιπτώσεις. Όμως ο χρήστης μπορεί να χρησιμοποιήσει ορισμένες συναρτήσεις του mbuf που υπάρχουν γενικότερα στο FreeBSD όπως η συνάρτηση m\_pullup που αναδιαμορφώνει το πακέτο mbuf και συγκεντρώνει ένα συγκεκριμένο μήκος δεδομένων σε ένα σημείο. Έτσι όλα τα δεδομένα είναι διαθέσιμα μαζί, χωρίς διακοπές. Ως ορίσματα έχει το m που είναι δείκτης στο mbuf και το len που είναι το μήκος δεδομένων που θέλει ο χρήστης συνεχόμενα. Παρακάτω παρουσιάζεται το συντακτικό και ένα παράδειγμα χρήσης της m\_pullup συνάρτησης.

Πίνακας 4.17. Συντακτικό και παράδειγμα χρήσης της m_pullup συνάρτησης	
Συντακτικό	m_pullup(struct mbuf *m, int len);
Παράδειγμα χρήσης	<pre>if ((m = m_pullup(m, sizeof(struct ether_header))) == NULL) {     return ENOBUFS; }</pre>

Ακόμη υπάρχει η συνάρτηση mtod που είναι πολύ σημαντική καθώς μετατρέπει τον δείκτη ενός mbuf σε έναν δείκτη για έναν συγκεκριμένο τύπο δομής δεδομένων. Ως ορίσματα έχει το m που είναι δείκτης στο mbuf και type που είναι ο τύπος δεδομένων στον οποίο θέλετε να μετατρέψετε τον δείκτη. Παρακάτω παρουσιάζεται το συντακτικό και ένα παράδειγμα χρήσης της mtod συνάρτησης.

Πίνακας 4.18. Συντακτικό και παράδειγμα χρήσης της mtod συνάρτησης	
Συντακτικό	mtod(struct mbuf *mbuf, type);
Παράδειγμα χρήσης	<pre>struct ip *ip_header = mtod(m, struct ip *);</pre>

Πολύ σημαντική είναι και η m\_data η οποία περιέχει έναν δείκτης που δείχνει στην αρχή των δεδομένων που περιέχονται στο mbuf.

Πίνακας 4.19. Συντακτικό και παράδειγμα χρήσης της mtod συνάρτησης	
Συντακτικό	m_data (caddr_t)

Παράδειγμα

χρήσης

```
m->m_data[2] = 10;
```

#### 4.5.1.3 Δημιουργία του πακέτου δεδομένων

Αφού παρέλθει η διαδικασία της τροποποίησης ακολουθεί η δημιουργία του πακέτου από τον χρήστη. Αρχικά το Netgraph δεν διαθέτει συναρτήσεις ή μακροεντολές για την δημιουργία νέου πακέτου δεδομένων. Για την κάλυψη των αναγκών ο χρήστης μπορεί να χρησιμοποιήσεις συναρτήσεις που υπάρχουν στο ήδη υπάρχων δικτυακό σύστημα του FreeBSD. Πιο συγκεκριμένα με τη χρήση της συνάρτησης `m_get` φτιάχνεται ένα καινούργιο `mbuf` για την εισαγωγή δεδομένων. Ως ορίσματα έχει το `how` που αναφέρει το πως να εισαχθούν τα δεδομένα μέσα στο `mbuf` και το `type` που είναι ο τύπος του `mbuf` που θα δημιουργηθεί. Παρακάτω παρουσιάζεται το συντακτικό της `m_get` συνάρτησης.

Πίνακας 4.20. Συντακτικό της `m_get` συνάρτησης

```
m_get(int how, int type);
```

#### 4.5.1.4 Αποστολή και αποδέσμευση του πακέτου δεδομένων

Εφόσον δημιουργήθηκε το πακέτο δεδομένων σειρά έχει η αποστολή του. Συνηθώς για την αποστολή χρησιμοποιείται η μακροεντολή `NG_FWD_NEW_DATA`, η οποία στέλνει το πακέτο απευθείας σε έναν κόμβο ή σε ένα άγκιστρο. Η συνάρτηση δέχεται ως ορίσματα την μεταβλητή `error`, η οποία χρησιμοποιείται για να διαπιστωθεί αν η προώθηση των δεδομένων ήταν επιτυχής ή όχι. Επίσης, περιλαμβάνεται το όρισμα `m_new`, που είναι το νέο `mbuf` που θα αποσταλεί. Τέλος, το όρισμα `item` περιέχει τα δεδομένα που πρόκειται να σταλούν και το όρισμα `hook` δείχνει στο άγκιστρο μέσω του οποίου θα προωθηθούν τα δεδομένα. Παρακάτω παρουσιάζεται το συντακτικό της `NG_FWD_NEW_DATA` μακροεντολής.

Πίνακας 4.21. Συντακτικό της `NG_FWD_NEW_DATA` μακροεντολής

```
NG_FWD_NEW_DATA(error, item, hook, m_new);
```

Παρόμοια λειτουργία με την NG\_FWD\_NEW\_DATA έχει η συνάρτηση ng\_send\_data που δίνει την δυνατότητα στο χρήστη να αποστέλλει και μεταδεδομένα μαζί με το πακέτο δεδομένων του. Ως ορίσματα έχει το hook που είναι δείκτης στο άγκιστρο hook, το m που είναι δείκτης στην δομή mbuf και το meta που είναι δείκτης στα μεταδεδομένα meta. Παρακάτω παρουσιάζεται το συντακτικό της ng\_send\_data συνάρτησης

Πίνακας 4.22. Συντακτικό της ng\_send\_data συνάρτησης

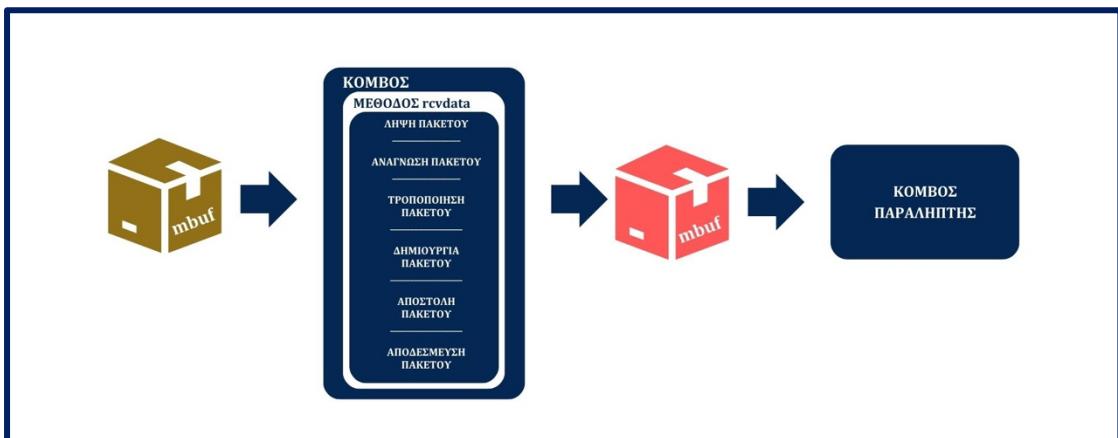
```
int ng_send_data(hook_p hook, struct mbuf *m, meta_p meta);
```

Τέλος ο χρήστης μπορεί να αποδεσμεύσει τους πόρους mbuf και item. Έτσι θα ελευθερώσει την μνήμη και θα είναι έτοιμος για να δεσμεύσει το επόμενο πακέτο. Αυτή η διαδικασία γίνεται με την εντολή NG\_FREE\_M για αποδέσμευση του mbuf και NG\_FREE\_ITEM για αποδέσμευση του item. Η NG\_FREE\_M μακροεντολή ως όρισμα έχει τον δείκτη m του πακέτου mbuf και η NG\_FREE\_ITEM μακροεντολή ως όρισμα έχει το item. Παρακάτω παρουσιάζεται το συντακτικό των δύο μακροεντολών

Πίνακας 4.23. Συντακτικό μακροεντολών αποδέσμευσης

Συντακτικό NG_FREE_M	NG_FREE_M(m)
Συντακτικό NG_FREE_ITEM	NG_FREE_ITEM(item)

Στην ακόλουθη εικόνα προβάλλεται η κυκλοφορίας ενός πακέτου δεδομένων στο δίκτυο.



Εικόνα 4.1. Αναπαράσταση ροής πακέτου δεδομένων στο δίκτυο.

Παρακάτω παρουσιάζεται ένα παράδειγμα διαχείρισης ενός πακέτου δεδομένων, χρησιμοποιώντας συνδυαστικά τις παραπάνω συναρτήσεις. Συγκεκριμένα λαμβάνει η μέθοδος το πακέτο και με την NGI\_GET\_M παίρνει το δείκτη m του πακέτου για να μπορεί να το χρησιμοποιήσει. Επειτα γίνεται έλεγχος για να εξασφαλιστεί ότι υπάρχει τουλάχιστον 1 byte διαθέσιμο με την συνάρτηση m\_pullup, αλλιώς αποδεσμεύει το πακέτο με τις συναρτήσεις NG\_FREE\_ITEM/M. Μετά τροποποιεί το πακέτο με την m\_data αλλάζωντας το πρώτο byte σε 42. Αφού έγινε αλλαγή του πακέτου το στέλνει μέσω του αγκίστρου hook στο κόμβο παραλήπτη με την συνάρτηση NG\_FWD\_NEW\_DATA. Τέλος αποδευσμεύει το πακέτο δεδομένων για να δουλέψει με το επόμενο πακέτο.

Πίνακας 4.15. Ενδεικτικό παράδειγμα διαχείρισης ενός πακέτου δεδομένου

```
static int rcvdata(hook_p hook, item_p item)
{
    struct mbuf *m;
    int error = 0;
    //Λήψη του πακέτου δεδομένων
    NGI_GET_M(item, m);
    //Τροποποίηση του πακέτου δεδομένων, για να εξασφαλίσουμε ότι έχουμε
    //τουλάχιστον 1 byte διαθέσιμο, αλλιώς αποδεσμεύει το πακέτο
    if (m_pullup(m, 1) == NULL) {
        NG_FREE_ITEM(item);
        NG_FREE_M(m);
        return (ENOBUFS);
    }
    // Αλλαγή του πρώτου byte σε 42
    m->m_data[0] = 42; //
    // Αποστολή του πακέτου δεδομένων
    NG_FWD_NEW_DATA(error, item, hook, m);
    // Αποδέσμευση του αρχικού πακέτου δεδομένων
    NG_FREE_M(m);
    NG_FREE_ITEM(item);
    return (error);
}
```

## 4.5.2 Χειρισμός μηνύματος ελέγχου

### 4.52.1 Λήψη και ανάγνωση του μηνύματος ελέγχου

Η εισαγωγή του μηνύματος ελέγχου σε ένα κόμβο του Netgraph γίνεται μέσω της κλήσης της μεθόδου `rcvmsg`. Αυτή η μέθοδο καλείται αυτόματα όταν ένα μηνύμα ελέγχου φτάσει στο κόμβο. Η μέθοδος χρειάζεται τρία ορίσματα το `node_p` node, το `item_p` item και το `hook_p` lasthook. Συγκεκριμένα το `lasthook` είναι το τελευταίο άγκιστρο που χρησιμοποιήθηκε για την παράδοση του item στον κόμβο. Το όρισμα item είναι ένας δείκτης σε μια δομή item και είναι το στοιχείο που θα επεξεργαστεί η συνάρτηση. Τέλος υπάρχει το όρισμα `node` που αναφέρεται στον κόμβο που συμμετέχει στο μήνυμα. Παρακάτω παρουσιάζεται το συντακτικό της μεθόδου

Πίνακας 4.18. Συντακτικό της `rcvmsg` μεθόδου

```
static int rcvmsg(node_p node, item_p item, hook_p lasthook) {  
    //ανάγνωση, τροποποίηση, κλήση συναρτήσεων κλπ./  
}
```

Η επόμενη διαδικασία που ακολουθεί μετά την λήψη του μηνύματος είναι η ανάγνωση του. Για αυτήν την διαδικασία συνήθως χρησιμοποιείται η μακροεντολή `NGI_GET_MSG`. Η μακροεντολή αυτή έχει ως ορίσματα το item και το msg. Πιο αναλυτικά το item είναι το όρισμα που το παίρνει από την ίδια την `rcvmsg` και το msg είναι ένας δείκτης στο `ng_mesg`. Συγκεκριμένα η μακροεντολή εξάγει μέσα από το item το μήνυμα, άρα μετά το item δεν περιέχει πλέον το μήνυμα και έτσι μπορεί να αποφευχθεί η διαρροή μνήμης. Παρακάτω παρουσιάζεται το συντακτικό της `NGI_GET_MSG` μακροεντολής.

Πίνακας 4.19. Συντακτικό της `NGI_GET_MSG` μακροεντολής

```
NGI_GET_MSG(item, msg);
```

Παρόμοια δουλεία με την `NGI_GET_MSG` κάνει και η μακροεντολή `NGI_MSG`. Η διαφορά τους είναι ότι η `NGI_MSG` δεν εξάγει από το item το μήνυμα δηλαδή δεν αφήνει το item κενό. Απλά το χρησιμοποιεί παίρνει τα δεδομένα που θέλει και το item συνεχίζει να

περιέχει το μήνυμά. Παρακάτω παρουσιάζεται το συντακτικό της NGI\_MSG μακροεντολής.

Πίνακας 4.20. Συντακτικό της NGI\_MSG μακροεντολής

```
NGI_MSG(item);
```

#### 4.5.2.2 Τροποποίηση του μηνύματος ελέγχου

Αφού λοιπόν έχει γίνει ανάγνωση των στοιχείων του μηνύματος μπορεί κανείς να ξεκινήσει την διαδικασία τροποποίησης του. Ουσιαστικά η τροποποίηση του μηνύματος γίνεται με την αλλαγή και επεξεργασία των πεδίων της δομής(struct) ng\_mesg. Μέσα σε αυτή την δομή υπάρχει η κεφαλίδα (header) του μηνύματος όσο και τα δεδομένα του. Επομένως δεν χρησιμοποιείται κάποια συνάρτηση η μακροεντολή του Netgraph για την τροποποίηση του μηνύματος. Παρακάτω παρουσιάζεται ένα ενδεικτικό παράδειγμα τροποποίησης.

Πίνακας 4.21. Οδηγός εντολών για παρεμβολή στο επίπεδο συνδέσμου

```
msg->header.type = NGM_HEADER;  
strcpy(msg->data, "new data");
```

#### 4.5.2.3 Δημιουργία του μηνύματος ελέγχου

Μετά την τροποποίηση του μηνύματος ακολουθεί η δημιουργία νέου μηνύματος προς κάποιον κόμβο παραλήπτη. Υπεύθυνη για αυτήν την δουλεία είναι κυρίως η μακροεντολή NG\_MKMESSAGE η οποία δημιουργεί ένα νέο μήνυμα από την αρχή. Η μακροεντολή χρησιμοποιεί πέντε βασικά ορίσματα για να δημιουργήσει ένα νέο μήνυμα. Αρχικά, το msg είναι ένας δείκτης που αναφέρεται στη νέα δομή του μηνύματος που θα δημιουργηθεί. Επιπλέον, το type αναφέρεται στον τύπο του κόμβου που θα λάβει αυτό το μήνυμα. Το cmd καθορίζει την εντολή που πρέπει να εκτελέσει ο κόμβος παραλήπτης, ενώ το len δείχνει το μέγεθος των δεδομένων του μηνύματος. Τέλος, τα flags ρυθμίζουν το πώς θα γίνει η εικώρηση μνήμης, είτε αμέσως (M\_NOWAIT) είτε περιμένοντας, όταν η

μνήμη δεν είναι διαθέσιμη (M\_WAITOK). Παρακάτω παρουσιάζεται το συντακτικό της NG\_MKMESSAGE μακροεντολής.

Πίνακας 4.22. Συντακτικό της NG\_MKMESSAGE μακροεντολής

```
NG_MKMESSAGE(struct ng_mesg *msg, u_int32_t type, u_int32_t cmd, int len, int flags);
```

Από την άλλη πλευρά άμα κάποιος θέλει απλά την δημιουργία απάντησης σε κάποιο μήνυμα που είχε λάβει τότε κατάλληλη μακροεντολή είναι η NG\_MKRESPONSE. Αυτή η μακροεντολή έχει τέσσερα ορίσματα για να δημιουργήσει το μήνυμα απάντησης. Το resp είναι ο δείκτης που οδηγεί στο νέο μήνυμα απόκρισης, ενώ το msg είναι το αρχικό μήνυμα. Το space καθορίζει το μέγεθος των δεδομένων της απόκρισης και τα flags ρυθμίζουν το πώς θα γίνει η εκχώρηση της μνήμης, είτε αμέσως (M\_NOWAIT) είτε περιμένοντας όταν η μνήμη δεν είναι διαθέσιμη (M\_WAITOK). Παρακάτω παρουσιάζεται το συντακτικό της NG\_MKRESPONSE μακροεντολής.

Πίνακας 4.23. Συντακτικό της NG\_MKRESPONSE μακροεντολής

```
NG_MKRESPONSE(struct ng_mesg *rsp, struct ng_mesg *msg, int len, int flags);
```

#### 4.5.2.4 Αποστολή και αποδέσμευση του μηνύματος ελέγχου

Έχοντας δημιουργήσει το μήνυμα ελέγχου χρειάζεται μετέπειτα να αποσταλεί στο παραλήπτη για να ολοκληρωθεί η διαδικασία. Υπεύθυνη για αυτήν την δουλεία είναι η συνάρτηση ng\_send\_msg η οποία αποστέλλει ένα μήνυμα από έναν κόμβο σε έναν άλλο κόμβο μέσα στο δίκτυο. Η συνάρτηση χρησιμοποιεί τέσσερα ορίσματα. Το node που είναι ο κόμβος που στέλνει το μήνυμα, το msg που είναι το ίδιο το μήνυμα, το path που δείχνει πού πρέπει να πάει το μήνυμα μέσα στο δίκτυο, και το path\_id που χρησιμοποιείται για να διευκρινίσει τη διαδρομή του μηνύματος. Να σημειωθεί ότι η συνάρτηση αυτή συνηθίζεται να χρησιμοποιείται σε συνδυασμό με την NG\_MKMESSAGE. Παρακάτω ακολουθεί το συντακτικό και ένα παράδειγμα χρήσης της συνάρτησης ng\_send\_msg.

Πίνακας 4.24. Συντακτικό και παράδειγμα χρήσης της ng\_send\_msg συνάρτησης

Συντακτικό	<pre>int ng_send_msg(node_p node, struct ng_mesg *msg, const char *path, u_int32_t path_id);</pre>
Παράδειγμα χρήσης	<pre>struct ng_mesg *msg; NG_MKMESSAGE(msg, my_type, my_cmd, sizeof(my_data), M_NOWAIT); if (msg != NULL) {     memcpy(msg-&gt;data, my_data, sizeof(my_data));     ng_send_msg(node, msg, NULL, 0); } else {     printf("error!!\n"); }</pre>

Ακόμη για αποστολή μηνύματος μπορεί κάποιος να χρησιμοποιήσει την μακροεντολή NG RESPOND\_MSG. Η ιδιαιτερότητα αυτής της μακροεντολής είναι ότι χρησιμοποιείται για την αποστολή μηνυμάτων απάντησης και συνδυάζεται μαζί με την χρήση της μακροεντολής NG\_MKRESPONSE. Η μακροεντολή χρησιμοποιεί τέσσερα ορίσματα. Το error που είναι υπεύθυνο για τα σφάλματα. Το node που αντιπροσωπεύει τον κόμβο που στέλνεται η απάντηση, το item που περιέχει το αρχικό μήνυμα, και το resp είναι το μήνυμα απόκρισης που θα σταλεί. Παρακάτω ακολουθεί το συντακτικό και ένα παράδειγμα χρήσης της NG RESPOND\_MSG μακροεντολής.

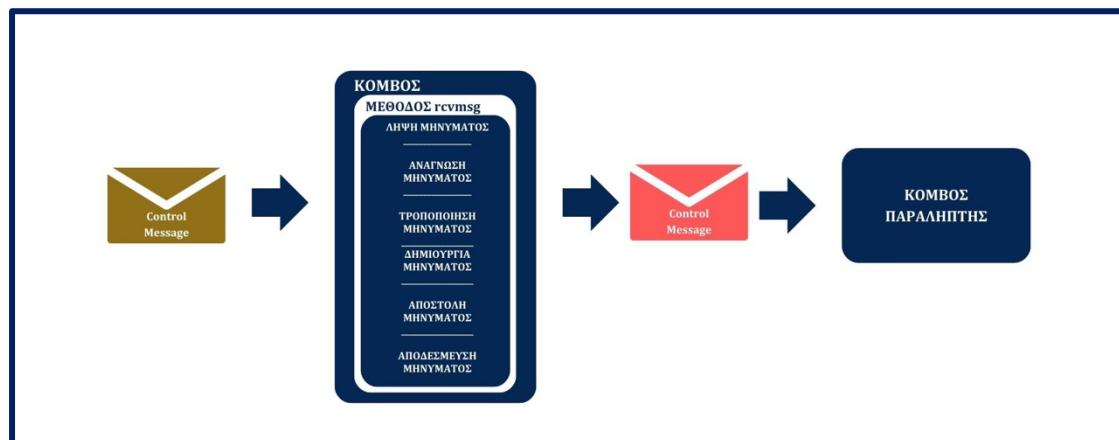
Πίνακας 4.25. Συντακτικό και παράδειγμα χρήσης της NG RESPOND\_MSG μακροεντολής

Συντακτικό	<pre>int NG_RESPOND_MSG(int error, node_p node, item_p item, struct ng_mesg *resp);</pre>
Παράδειγμα χρήσης	<pre>struct ng_mesg *resp; NG_MKRESPONSE(resp, orig_msg, sizeof(data), M_NOWAIT); if (resp != NULL) {     memcpy(resp-&gt;data, &amp;data, sizeof(data));     NG_RESPOND_MSG(0, node, item, resp); } else {     NG_RESPOND_MSG(ENOMEM, node, item, NULL); }</pre>

Εν κατακλείδι ο χρήστης μπορεί να απελευθερώσει το μήνυμα χρησιμοποιώντας την μακροεντολή NG\_FREE\_MSG. Έτσι απελευθερώνεται η μνήμη που είχε δεσμευτεί για ένα μήνυμα και είναι έτοιμη να δεχτεί το επόμενο μήνυμα. Ως όρισμα έχει ένα δείκτη msg που δείχνει στη δομή ng\_mesg του μηνύματος που απελευθερώνεται. Παρακάτω ακολουθεί το συντακτικό της NG\_FREE\_MSG μακροεντολής.

Πίνακας 4.26. Συντακτικό της NG_FREE_MSG μακροεντολής
NG_FREE_MSG(msg)

Στην ακόλουθη εικόνα παρουσιάζεται μια απεικόνιση της κυκλοφορίας ενός μηνύματος ελέγχου στο δίκτυο.



Εικόνα 4.2. Αναπαράσταση ροής μηνύματος ελέγχου στο δίκτυο.

Βέβαια μέσα στο κόμβο υπάρχουν και άλλες συναρτήσεις και μέθοδοι που μπορούν να υλοποιηθούν. Ανάλογα το σκοπό ο εκάστοτε χρήστης μπορεί να τις συνδυάσει και να διαμορφώσει την λειτουργικότητα του κόμβου του. Τόσο οι συναρτήσεις όσο και οι μακροεντολές που υπάρχουν στο Netgraph παρατίθενται στο Παράρτημα 2.

Παρακάτω παρουσιάζεται ένα παράδειγμα διαχείρισης ενός μηνύματος ελέγχου, χρησιμοποιώντας συνδυαστικά τις παραπάνω συναρτήσεις. Συγκεκριμένα λαμβάνει η μέθοδος το μήνυμα και με την NGI\_GET\_MSG παίρνει το δείκτη m και item του μηνύματος για να μπορει να το χρησιμοποιήσει. Επειτα γίνεται τυπικός έλεγχος με την εκτύπωση του μυνήματος για να δει ο χρήστης ορισμένα χαρακτηριστικά του μυνήματος και μετα με την strcpy αντιγραφει την συμβολοσειρά NEW DATA στα δεδομένα του μυνήματος.

Μετά γίνεται η δημιουργία ενός νέου μυνήματος με την συνάρτηση NG\_MKMESSAGE. Στο εσωτερικό της γίνεται έλεγχος αν το μήνυμα δημιουργηθηκέ επιτυχώς ή όχι. Αμα δεν μπόρεσε να το δημιουργήσει τότε το αποδευσμεύει με την συνάρτηση NG\_FREE\_MSG. Αν κατάφερε να το δημιουργησεί τότε το στέλνει έπειτα στο κόμβο παραλήπτη με την συνάρτηση ng\_send\_msg. Μετά αποδεμεύεται το παρών μήνυμα με την συνάρτηση NG\_FREE\_MSG για να λάβει η μέθοδος το επόμενο μήνυμα.

Πίνακας 4.15. Ενδεικτικό παράδειγμα διαχείρισης ενός μηνύματος ελέγχου

```
static int rcvmsg(node_p node, item_p item, hook_p lasthook)
{
    struct ng_mesg *msg;
    struct ng_mesg *new_msg;
    // Λήψη του μηνύματος
    NGI_GET_MSG(item, msg);
    // Ανάγνωση του μηνύματος
    printf("Received message: type=%d, cmd=%d, data=%s\n",
           msg->header.type, msg->header.cmd, (char *)msg->data);
    // Τροποποίηση του μηνύματος
    strcpy((char *)msg->data, "NEW DATA...");
    // Δημιουργία νέου μηνύματος
    NG_MKMESSAGE(new_msg, NGM_GENERIC_COOKIE, NGM_ECHO, strlen("reply") + 1,
M_NOWAIT);
    if (new_msg == NULL) {
        NG_FREE_MSG(msg);
        return (ENOMEM);
    }
    strcpy((char *)new_msg->data, "reply");
    // Αποστολή του νέου μηνύματος
    ng_send_msg(node, new_msg, lasthook->name, NG_HOOK_NODE_ID(lasthook));
    // Αποδέσμευση του αρχικού μηνύματος
    NG_FREE_MSG(msg);
    return (0);
}
```

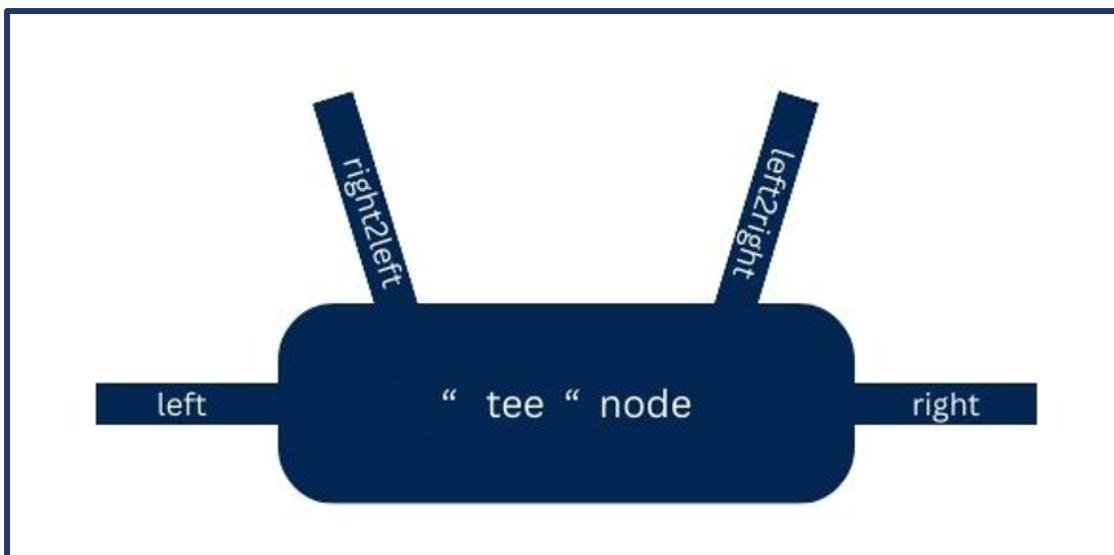
# Κεφάλαιο 5.

## Δημιουργία κόμβου **ng\_triple\_tee**

### 5.1 Ιδέα του κόμβου **ng\_triple\_tee**

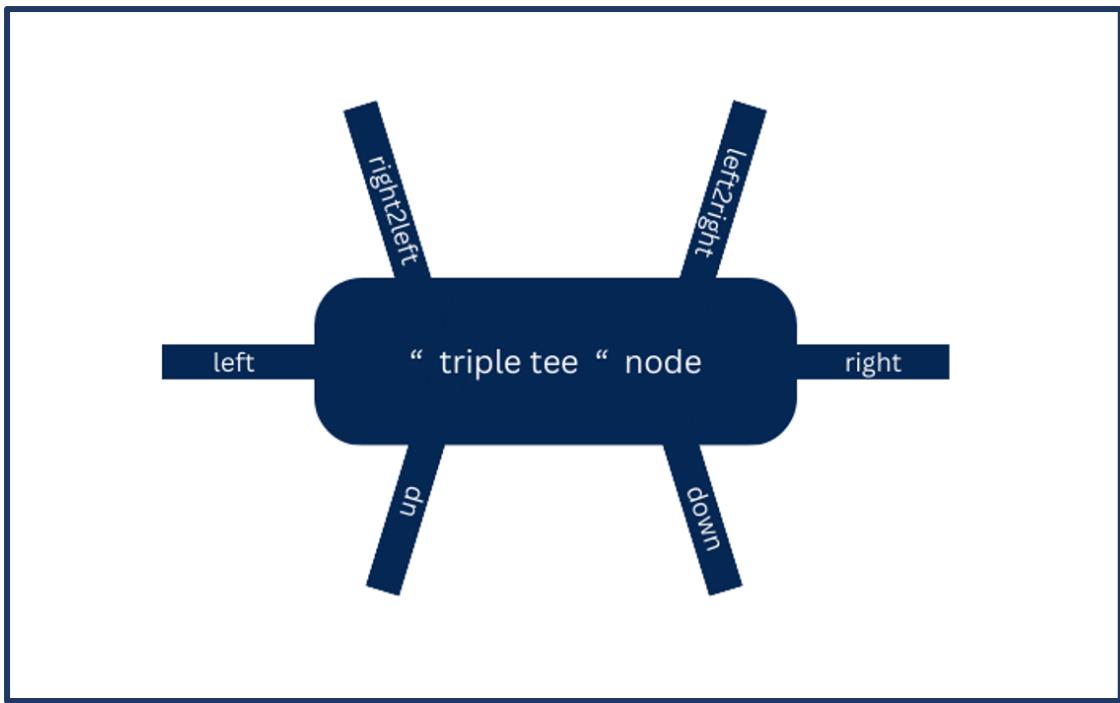
Ο κόμβος **ng\_triple\_tee** αποτελεί παραλλαγή του ήδη υπάρχοντος κόμβου **ng\_tee**. Για να γίνει κατανοητή η λειτουργικότητα του κόμβου **ng\_triple\_tee**, προέχει η κατανόηση του κόμβου **tee**.

Αρχικά για την καλύτερη κατανόηση της λειτουργίας του κόμβου **tee** η περιγραφή του θα χωριστεί σε δύο πλευρές την δομική και την λειτουργική. Από δομικής πλευράς ο κόμβος **tee** αποτελείται από τέσσερα άγκιστρα το **right**, το **left**, το **right2left** και το **left2right**. Πιο συγκεκριμένα τα **right** και **left** άγκιστρα λαμβάνουν δεδομένα ενώ τα **right2left** και **left2right** άγκιστρα προωθούν τα δεδομένα. Το **right2left** άγκιστρο προωθεί δεδομένα που έλαβε το **right** άγκιστρο ενώ το **left2right** άγκιστρο προωθεί δεδομένα που έλαβε το **left** άγκιστρο. Στην εικόνα 5.1 απεικονίζεται η μορφή του κόμβου **tee**. Από λειτουργικής πλευράς, ο κύριος σκοπός του κόμβου **tee** είναι να αντιγράφει τα δεδομένα που λαμβάνονται στα άγκιστρα **right** και **left** και στην συνέχεια τα διπλότυπα δεδομένα να τα προωθεί στα **right2left** και **left2right** άγκιστρα. Αυτό θεωρείτε πολύ σημαντικό γιατί ο κόμβος **tee** μπορεί να εμπλακεί σε πολλά σενάρια χρήσης. Για παράδειγμα, μπορεί να χρησιμοποιηθεί για την καταγραφή πακέτων, την παρακολούθηση της κυκλοφορίας δικτύου σε πραγματικό χρόνο ή ακόμα και την αποσφαλμάτωση εφαρμογών δικτύου.



Εικόνα 5.1. Σχηματική αναπαράσταση κόμβου tee

Μελετώντας το κόμβο tee, παρατηρήθηκε ότι με μερικές σημαντικές τροποποιήσεις θα δημιουργούνται ένα πραγματικά ισχυρό εργαλείο στα χέρια του χρήστη. Έτσι προστέθηκαν τα άγκιστρα up και down, τα οποία έχουν την ίδια λειτουργία με τα άγκιστρα right2left και left2right. Συγκεκριμένα το up άγκιστρο προωθεί δεδομένα του right αγκίστρου και το down προωθεί δεδομένα του left αγκίστρου. Ταυτόχρονα τα άγκιστρα up και down έχουν την δυνατότητα να καταγράφουν δεδομένα σχετικά με τις κεφαλίδες IP και Ethernet. Ειδικότερα το άγκιστρο down έχει την δυνατότητα να εξάγει δεδομένα από τις κεφαλίδες IP που διέρχονται από αυτό είτε αυτές προέρχονται από το επίπεδο δικτύου είτε από το επίπεδο συνδέσμου. Τα δεδομένα που κρατάει αφορούν την διεύθυνση αποστολέα, την διεύθυνση παραλήπτη καθώς και το όνομα του πρωτοκόλλου επιπέδου μεταφοράς. Από την άλλη πλευρά το άγκιστρο up έχει την δυνατότητα να εξάγει δεδομένα από τις κεφαλίδες ETHERNET που διέρχονται από αυτό. Πιο αναλυτικά κρατάει την διεύθυνση MAC του αποστολέα, την διεύθυνση MAC του παραλήπτη καθώς και τον τύπο του ether(EtherType). Ακόμη προστέθηκε η χρονομέτρηση η οποία ξεκινάει όταν εισέρχονται τα δεδομένα στο κόμβο και ο χρήστης μόλις πατήσει την εντολή getstats λαμβάνει σε ns το χρόνο που έχει περάσει μέχρι εκείνη την στιγμή. Τέλος προστέθηκε και ένα μετρητής error που μετράει άμα συνέβη κάποιο σφάλμα στην παράδοση των διπλότυπων πακέτων. Έτσι δημιουργήθηκε ο κόμβος triple\_tee του οποίου η μορφή απεικονίζεται στην εικόνα 5.2.



Εικόνα 5.2. Σχηματική αναπαράσταση κόμβου triple\_tee

## 5.2 Υλοποίηση του κόμβου ng\_triple\_tee

Παρακάτω παρατίθεται το Makefile για τη μετατροπή του κώδικα του κόμβου σε kernelobject αρχείου ώστε να μπορεί να φορτωθεί στο πυρήνα. Ακόμη παρατίθεται ο πρόσθετος κώδικας ώστε ο κόμβος tee να μετατραπεί σε triple\_tee. Επομένως προστέθηκε κώδικας στο αρχείο ng\_tee.c[68] και δημιουργήθηκε το ng\_triple\_tee.c, καθώς και στο αρχείο ng\_tee.h[69] και δημιουργήθηκε το ng\_triple\_tee.h. Παρακάτω με κόκκινο χρώμα είναι τα καινούργια στοιχεία κώδικα που προστέθηκαν για να δημιουργηθεί ο νέος κόμβος. Τέλος με μαύρο χρώμα είναι τα κομμάτια του κώδικα που παρέμειναν ίδια.

Πίνακας 5.1. Περιεχόμενα αρχείου Makefile του κόμβου ng\_triple\_tee

### Makefile ng\_triple\_tee

```
KMOD = ng_triple_tee
SRCS = ng_triple_tee.c

CFLAGS = -DVIMAGE
```

```
.include<bsd.kmod.mk>
```

## Πίνακας 5.2. Περιεχόμενα αρχείου ng\_triple\_tee.h

### ng\_triple\_tee.h

```
1. #ifndef _NETGRAPH_NG_TRIPLE_TEE_H_
2. #define _NETGRAPH_NG_TRIPLE_TEE_H_

3. /* Node type name and magic cookie */
4. #define NG_TRIPLE_TEE_NODE_TYPE      "triple_tee"
5. #define NGM_TRIPLE_TEE_COOKIE       918107059

6. /* Hook names */
7. #define NG_TRIPLE_TEE_HOOK_RIGHT    "right"
8. #define NG_TRIPLE_TEE_HOOK_LEFT     "left"
9. #define NG_TRIPLE_TEE_HOOK_RIGHT2LEFT "right2left"
10. #define NG_TRIPLE_TEE_HOOK_LEFT2RIGHT "left2right"
11. #define NG_TRIPLE_TEE_HOOK_RIGHT2LEFT "up"
12. #define NG_TRIPLE_TEE_HOOK_LEFT2RIGHT "down"

13. /* Statistics structure for one hook */
14. struct ng_triple_tee_hookstat {
15.     u_int64_t    inOctets;
16.     u_int64_t    inFrames;
17.     u_int64_t    outOctets;
18.     u_int64_t    outFrames;
19.     u_int64_t    Delay;
20.     u_int64_t    Errors;

21. };

22. /* Keep this in sync with the above structure definition */
23. #define NG_TRIPLE_TEE_HOOKSTAT_INFO { \
24.     { "inOctets",      &ng_parse_uint64_type },      \
25.     { "inFrames",      &ng_parse_uint64_type },      \
26.     { "outOctets",     &ng_parse_uint64_type },      \
27.     { "outFrames",     &ng_parse_uint64_type },      \
28.     { "Delay",         &ng_parse_uint64_type },      \
29.     { "Errors",        &ng_parse_uint64_type },      \
30.     { NULL }          \
31. }
```

```

32. /* Statistics structure returned by NGM_TEE_GET_STATS */
33. struct ng_triple_tee_stats {
34.     struct ng_triple_tee_hookstat    right;
35.     struct ng_triple_tee_hookstat    left;
36.     struct ng_triple_tee_hookstat   right2left;
37.     struct ng_triple_tee_hookstat   left2right;
38.     struct ng_triple_tee_hookstat   up;
39.     struct ng_triple_tee_hookstat   down;
40. };
41. /* Keep this in sync with the above structure definition */
42. #define NG_TRIPLE_TEE_STATS_INFO(hstype) { \
43.     { "right",      (hstype) }, \
44.     { "left",       (hstype) }, \
45.     { "right2left", (hstype) }, \
46.     { "left2right", (hstype) }, \
47.     { "up",         (hstype) }, \
48.     { "down",       (hstype) }, \
49.     { NULL } \
50. }
51. /* Netgraph commands */
52. enum {
53.     NGM_TRIPLE_TEE_GET_STATS = 1,          /* get stats */
54.     NGM_TRIPLE_TEE_CLR_STATS,            /* clear stats */
55.     NGM_TRIPLE_TEE_GETCLR_STATS,        /* atomically get and clear stats */
56. };
57. #endif /* _NETGRAPH_NG_TRIPLE_TEE_H_ */

```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο `ng_triple_tee.h` και απεικονίζεται στο πίνακα 5.2

Γραμμές 4-5: Ορισμός ονόματος του τύπου κόμβου ως `triple_tee` και ορισμός του `typecookie` του.

Γραμμές 11-12 && 38-39 && 47-48: Γίνεται ο ορισμός των ονομάτων των καινούργιων αγκίστρων `up` και `down` και έπειτα γίνεται προσθήκη στις ήδη υπάρχουσες δομές των αγκίστρων.

Γραμμές 19-20 && 28-29: Προσθήκη επιπλέων πληροφοριών στην δομή των στατιστικών και έπειτα γίνεται η προσθήκη τους στην ήδη υπάρχουσα δομή για την αναπαράσταση των στατιστικών

### Πίνακας 5.3. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 1

```
1. #include <sys/param.h>
2. #include <sys/sysctl.h>
3. #include <sys/errno.h>
4. #include <sys/kernel.h>
5. #include <sys/malloc.h>
6. #include <sys/libkern.h>
7. #include <netinet/in.h>
8. #include <netinet/ip.h>
9. #include <net/ether.h>
10. #include <sys/socket.h>
11. #include <sys/types.h>
12. #include <net/if.h>
13. #include <sys/mbuf.h>
14. #include <netgraph/ng_message.h>
15. #include <sys/time.h>
16. #include <netgraph/netgraph.h>
17. #include <netgraph/ng_parse.h>
18. #include <netgraph/ng_triple_tee.h>

19. /* Per hook info */
20. struct hookinfo {
21.     hook_p           hook;
22.     struct hookinfo *dest, *dup;
23.     struct ng_triple_tee_hookstat  stats;
24. };
25. typedef struct hookinfo *hi_p;

26. /* Per node info */
27. struct privdata {
28.     struct hookinfo      left;
29.     struct hookinfo      right;
30.     struct hookinfo      left2right;
31.     struct hookinfo      right2left;
32.     struct hookinfo      up;//add up
33.     struct hookinfo      down;//add down

34. };
35. typedef struct privdata *sc_p;

36. /* Netgraph methods */
37. static ng_constructor_t    ng_triple_tee_constructor;
38. static ng_rcvmsg_t       ng_triple_tee_rcvmsg;
39. static ng_close_t        ng_triple_tee_close;
40. static ng_shutdown_t     ng_triple_tee_shutdown;
41. static ng_newhook_t      ng_triple_tee_newhook;
42. static ng_rcvdata_t      ng_triple_tee_rcvdata;
43. static ng_disconnect_t   ng_triple_tee_disconnect;

44. /* Parse type for struct ng_tee_hookstat */
45. static const struct ng_parse_struct_field ng_triple_tee_hookstat_type_fields[]
46. = NG_TRIPLE_TEE_HOOKSTAT_INFO;
47. static const struct ng_parse_type ng_triple_tee_hookstat_type = {
48.     &ng_parse_struct_type,
49.     &ng_triple_tee_hookstat_type_fields
50. };

51. /* Parse type for struct ng_tee_stats */
52. static const struct ng_parse_struct_field ng_triple_tee_stats_type_fields[]
53. = NG_TRIPLE_TEE_STATS_INFO(&ng_triple_tee_hookstat_type);
54. static const struct ng_parse_type ng_triple_tee_stats_type = {
55.     &ng_parse_struct_type,
56.     &ng_triple_tee_stats_type_fields
57. };

58. /* List of commands and how to convert arguments to/from ASCII */
59. static const struct ng_cmdlist ng_triple_tee_cmds[] = {
60. {
61.     NGM_TRIPLE_TEE_COOKIE,
62.     NGM_TRIPLE_TEE_GET_STATS,
63.     "getstats",
64.     NULL,
```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο ng\_triple\_tee.c και απεικονίζεται στο πίνακα 5.3

Γραμμές 7-8-12-15-18: Προσθήκη των κατάλληλων αρχείων κεφαλίδας

Γραμμές 32-33: Προσθήκη των αγκίστρων στην δομή που συλλέγονται πληροφορίες για τα άγκιστρα του κόμβου.

#### Πίνακας 5.4. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 2

```
1. &ng_triple_tee_stats_type
2. },
3. {
4. NGM_TRIPLE_TEE_COOKIE,
5. NGM_TRIPLE_TEE_CLR_STATS,
6. "clrstats",
7. NULL,
8. NULL
9. },
10. {
11. NGM_TRIPLE_TEE_COOKIE,
12. NGM_TRIPLE_TEE_GETCLR_STATS,
13. "getclrstats",
14. NULL,
15. &ng_triple_tee_stats_type
16. },
17. { 0 }
18. };
19. /* Netgraph type descriptor */
20. static struct ng_type ng_triple_tee_typestruct = {
21. .version = NG_ABI_VERSION,
22. .name = NG_TRIPLE_TEE_NODE_TYPE,
23. .constructor = ng_triple_tee_constructor,
24. .rcvmsg = ng_triple_tee_rcvmsg,
25. .close = ng_triple_tee_close,
26. .shutdown = ng_triple_tee_shutdown,
27. .newhook = ng_triple_tee_newhook,
28. .recvdata = ng_triple_tee_recvdata,
29. .disconnect = ng_triple_tee_disconnect,
30. .cmdlist = ng_triple_tee_cmds,
31. };
32. NETGRAPH_INIT(triple_tee, &ng_triple_tee_typestruct);
33. /*
34. Node constructor
35. */
36. static int
37. ng_triple_tee_constructor(node_p node)
38. {
39. sc_p privdata;
40. privdata = malloc(sizeof(*privdata), M_NETGRAPH, M_WAITOK | M_ZERO);
41. NG_NODE_SET_PRIVATE(node, privdata);
42. return (0);
43. }
44. /*
45. Add a hook
46. */
47. static int
48. ng_triple_tee_newhook(node_p node, hook_p hook, const char *name)
49. {
50. sc_p privdata = NG_NODE_PRIVATE(node);
51. hi_p hinfo;
52. /* Precalculate internal paths. */
53. if (strcmp(name, NG_TRIPLE_TEE_HOOK_RIGHT) == 0) {
54. hinfo = &privdata->right;
55. if (privdata->left.dest)
56. 55.1. privdata->left.dup = privdata->left.dest;
57. privdata->left.dest = hinfo;
58. privdata->right2left.dest = hinfo;
59. } else if (strcmp(name, NG_TRIPLE_TEE_HOOK_LEFT) == 0) {
60. hinfo = &privdata->left;
61. if (privdata->right.dest)
62. 61.1. privdata->right.dup = privdata->right.dest;
63. privdata->right.dest = hinfo;
64. privdata->left2right.dest = hinfo;
64. privdata->down.dest = hinfo;//add down
```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο ng\_triple\_tee.c και απεικονίζεται στο πίνακα 5.4

Γραμμές 58: Όταν λαμβάνεται πληροφορία στο right άγκιστρο τότε προωθείται επιπλέον και στο up άγκιστρο.

Γραμμές 64: Όταν λαμβάνεται πληροφορία στο left άγκιστρο τότε προωθείται επιπλέον και στο down άγκιστρο.

### Πίνακας 5.5. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 3

```
1.     &} else if (strcmp(name, NG_TRIPLE_TEE_HOOK_RIGHT2LEFT) == 0) {
2.         hinfo = &privdata->right2left;
3.         if (privdata->right.dest)
4.             3.1. privdata->right.dup = hinfo;
5.         } else if (strcmp(name, NG_TRIPLE_TEE_HOOK_LEFT2RIGHT) == 0) {
6.             hinfo = &privdata->left2right;
7.             if (privdata->left.dest)
8.                 7.1. privdata->left.dup = hinfo;
9.             } else if (strcmp(name, NG_TRIPLE_TEE_HOOK_UP) == 0) {
10.                hinfo = &privdata->up;//add up
11.                if (privdata->right.dest)
12.                    11.1.      privdata->right.dup = hinfo;
13.                } else if (strcmp(name, NG_TRIPLE_TEE_HOOK_DOWN) == 0) {
14.                    hinfo = &privdata->down;//add down
15.                    if (privdata->left.dest)
16.                        15.1.      privdata->left.dup = hinfo;
17.                    } else
18.                        return (EINVAL);
19.                    hinfo->hook = hook;
20.                    bzero(&hinfo->stats, sizeof(hinfo->stats));
21.                    NG_HOOK_SET_PRIVATE(hook, hinfo);
22.                    return (0);
23.                }

24. /*
25. Receive a control message
26. */
27. static int
28. ng_triple_tee_rcvmsg(node_p node, item_p item, hook_p lasthook)
29. {
30.     const sc_p sc = NG_NODE_PRIVATE(node);
31.     struct ng_mesg *resp = NULL;
32.     int error = 0;
33.     struct ng_mesg *msg;
34.     NGI_GET_MSG(item, msg);
35.     switch (msg->header.typecookie) {
36.     case NGM_TRIPLE_TEE_COOKIE:
37.         switch (msg->header.cmd) {
38.             case NGM_TRIPLE_TEE_GET_STATS:
39.             case NGM_TRIPLE_TEE_CLR_STATS:
40.             case NGM_TRIPLE_TEE_GETCLR_STATS:
40.1.             {
40.2.                 struct ng_triple_tee_stats *stats;

        40.2.1.1.     if (msg->header.cmd != NGM_TRIPLE_TEE_CLR_STATS) {
            40.2.1.1.1.     NG_MKRESPONSE(resp, msg,
            40.2.1.1.2.             sizeof(*stats), M_NOWAIT);
            40.2.2.     if (resp == NULL) {
            40.2.3.         error = ENOMEM;
            40.2.4.         goto done;
            40.2.5.     }
            40.2.6.     stats = (struct ng_triple_tee_stats *)resp->data;
            40.2.7.     bcopy(&sc->right.stats, &stats->right,
            40.2.8.             sizeof(stats->right));
            40.2.9.     bcopy(&sc->left.stats, &stats->left,
            40.2.10.            sizeof(stats->left));
            40.2.11.    bcopy(&sc->right2left.stats, &stats->right2left,
            40.2.12.            sizeof(stats->right2left));
            40.2.13.    bcopy(&sc->left2right.stats, &stats->left2right,
            40.2.14.            sizeof(stats->left2right));
```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο `ng_triple_tee.c` και απεικονίζεται στο πίνακα 5.5

Γραμμές 9-12.1: Ο κώδικας αυτός προσθέτει άγκιστρα σε ένα κόμβο, στη θέση `up`. Αν υπάρχει ήδη προορισμός (`dest`) στην αντίστοιχη κατεύθυνση (`right` για `up`) τότε αποθηκεύει την πληροφορία του νέου άγκιστρο ως διπλό (`dup`). Αν δεν υπάρχει προορισμός, τότε το ορίζει ως τον κύριο προορισμό (`dest`).

Γραμμές 13-16.1: Ο κώδικας αυτός προσθέτει άγκιστρα σε ένα κόμβο, στη θέση `down`. Αν υπάρχει ήδη προορισμός (`dest`) στην αντίστοιχη κατεύθυνση (`left` για `down`) τότε αποθηκεύει την πληροφορία του νέου άγκιστρο ως διπλό (`dup`). Αν δεν υπάρχει προορισμός, τότε το ορίζει ως τον κύριο προορισμό (`dest`).

Πίνακας 5.6. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 4

```

1.1.1. bcopy(&sc->up.stats, &stats->up,
1.1.2. sizeof(stats->up));//add up
1.1.3. bcopy(&sc->down.stats, &stats->down,
1.1.4. sizeof(stats->down));//add down
    if (msg->header.cmd != NGM_TRIPLE_TEE_GET_STATS) {
1.1.5. bzero(&sc->right.stats,
1.1.6. sizeof(sc->right.stats));
1.1.7. bzero(&sc->left.stats,
1.1.8. sizeof(sc->left.stats));
1.1.9. bzero(&sc->right2left.stats,
1.1.10. sizeof(sc->right2left.stats));
1.1.11. bzero(&sc->left2right.stats,
1.1.12. sizeof(sc->left2right.stats));
1.1.13. bzero(&sc->up.stats,//add up
1.1.14. sizeof(sc->up.stats));
1.1.15. bzero(&sc->down.stats,//add down
1.1.16. sizeof(sc->down.stats));

1.2. }
    1.2.1.1. break;
1.3. }

2. default:
2.1. error = EINVAL;
2.2. break;
3. }
4. break;
5. case NGM_FLOW_COOKIE:
6. if (lasthook == sc->left.hook || lasthook == sc->right.hook) {
6.1. hi_p const hinfo = NG_HOOK_PRIVATE(lasthook);
6.2. if (hinfo && hinfo->dest) {
6.2.1. NGI_MSG(item) = msg;
6.2.2. NG_FWD_ITEM_HOOK(error, item, hinfo->dest->hook);
6.2.3. return (error);
6.3. }
7. }
8. break;
9. default:
10. error = EINVAL;
11. break;
12. }
13. done:
14. NG_RESPOND_MSG(error, node, item, resp);
15. NG_FREE_MSG(msg);
16. return (error);
17. }

18. static uint64_t timespec_to_ns(struct timespec *ts) {
18.1.     return (ts->tv_sec * 1000000000ULL) + ts->tv_nsec;//calculate time in ns
19. }

20. static int
21. ng_triple_tee_rcvdata(hook_p hook, item_p item)
22. {
23. struct ip *ip_hdr;
24. const sc_p sc = NG_NODE_PRIVATE(NG_HOOK_NODE(hook));
25. const hi_p hinfo = NG_HOOK_PRIVATE(hook);
25.1.     hi_p h;
26. struct ether_header *eh=NULL;
27. int error = 0;
28. char *protocol;

29. struct timespec start, end, diff;

30. nanouptime(&start);//begin the timing
31. struct mbuf *m=NGI_M(item);

```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο ng\_triple\_tee.c και απεικονίζεται στο πίνακα 5.6

Γραμμές 1.1.1-1.1.4: Γίνεται αντιγραφή των στατιστικών από τις δομές sc->up.stats και sc->down.stats ώστε να γίνει αναπαράστασή τους με την getstats

Γραμμές 1.1.13-1.1.16: Γίνεται εκκαθάριση των στατιστικών που είναι αποθηκευμένες στις δομές των στατιστικών

Γραμμές 18-19: Συνάρτηση για τον μετατροπής της χρονομέτρησης των δεδομένων σε nanoseconds.

Γραμμές 29-30 Εκκίνηση της χρονομέτρησης με την συνάρτηση nanouptime

Γραμμές 23-24-26-28: Ορισμός δομών και μεταβλητών που βοηθάνε στην διαχείριση των κεφαλίδων (ip,ether).

Γραμμές 29-30: Ορισμός δομών και εκκίνηση της χρονομέτρησης

Πίνακας 5.7. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 5

```
1.1. if (sc->up.hook != NULL) {
1.2.     if (m->m_len < sizeof(struct ether_header)) {
1.2.1.         m = m_pullup(m, sizeof(struct ether_header));
1.2.2.         if (m == NULL) {
1.2.3.             NG_FREE_ITEM(item);
1.2.4.             return ENOBUFS;
1.2.5.         }
1.3.     }
1.4.     struct ether_header *eh = mtod(m, struct ether_header *);
1.5.     printf("SRC_MAC: %02x:%02x:%02x:%02x:%02x:%02x, DST_MAC:
1.5.         %02x:%02x:%02x:%02x:%02x:%02x, EtherType: 0x%04x\n",
1.5.1.         eh->ether_shost[0], eh->ether_shost[1], eh->ether_shost[2],
1.5.2.         eh->ether_shost[3], eh->ether_shost[4], eh->ether_shost[5],
1.5.3.         eh->ether_dhost[0], eh->ether_dhost[1], eh->ether_dhost[2],
1.5.4.         eh->ether_dhost[3], eh->ether_dhost[4], eh->ether_dhost[5],
1.5.5.         ntohs(eh->ether_type));

1.6. }
2. if (sc->down.hook != NULL) {
2.1.     if (m->m_pkthdr.len < sizeof(struct ether_header) + sizeof(struct ip)) {

2.1.1.         m = m_pullup(m, sizeof(struct ether_header) + sizeof(struct ip));
2.1.2.         if (m == NULL) {
2.1.3.             NG_FREE_ITEM(item);
2.1.4.             return ENOBUFS;
2.1.5.         }
2.2.     }
2.3.     eh = mtod(m, struct ether_header *);
2.4.     if (ntohs(eh->ether_type) == ETHERTYPE_IP) {
2.4.1.         ip_hdr = (struct ip *) (mtod(m, char *) + sizeof(struct ether_header));
2.4.2.         switch (ip_hdr->ip_p) {
2.4.3.             case IPPROTO_TCP:
2.4.3.1.                 protocol = "TCP";
2.4.3.2.                 break;
2.4.4.             case IPPROTO_UDP:
2.4.4.1.                 protocol = "UDP";
2.4.4.2.                 break;
2.4.5.             default:
2.4.5.1.                 protocol = "OTHER";
2.4.5.2.                 break;
2.4.6.         }
2.4.7.         printf("ETHER_SRC_IP: %d.%d.%d.%d, DST_IP: %d.%d.%d.%d, Protocol: %s\n",
2.4.8.         ((ip_hdr->ip_src.s_addr & 0xFF)), ((ip_hdr->ip_src.s_addr >> 8) & 0xFF),
2.4.9.         ((ip_hdr->ip_src.s_addr >> 16) & 0xFF), ((ip_hdr->ip_src.s_addr >> 24) &
2.4.10.        0xFF),
2.4.11.        ((ip_hdr->ip_dst.s_addr & 0xFF)), ((ip_hdr->ip_dst.s_addr >> 8) & 0xFF),
2.4.12.        ((ip_hdr->ip_dst.s_addr >> 16) & 0xFF), ((ip_hdr->ip_dst.s_addr >> 24) &
0xFF),
2.4.12.        protocol);
2.5.     }
}
```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο `ng_triple_tee.c` και απεικονίζεται στο πίνακα 5.7

Γραμμή 1.1: Ελέγχει αν ο hook "up" δεν είναι NULL, δηλαδή αν υπάρχει συνδεδεμένος κόμβος.

Γραμμή 1.2: Ελέγχει αν το μήκος του πακέτου είναι μικρότερο από το μέγεθος της δομής της κεφαλίδας `ether_header`. Αυτό είναι απαραίτητο για να βεβαιωθεί ότι το πακέτο έχει αρκετό χώρο για να περιέχει μια Ethernet κεφαλίδα.

Γραμμή 1.2.1: Αν το πακέτο δεν έχει αρκετό μήκος, καλεί τη συνάρτηση `m_pullup` για να τραβήξει περισσότερα δεδομένα και να εξασφαλίσει ότι το πακέτο έχει τουλάχιστον το μέγεθος του `ether_header`.

Γραμμές 1.2.2-1.2.5.: Ελέγχει αν απέτυχε η κλήση του `m_pullup` και αν ισχύει απελευθερώνει το στοιχείο item του μηνύματος.

Γραμμή 1.4: Μετατρέπει τα δεδομένα του πακέτου σε δείκτη προς την δομή `ether_header` χρησιμοποιώντας τη συνάρτηση `mtod`.

Γραμμή 1.5: Εκτυπώνει τις διευθύνσεις MAC πηγής και προορισμού, καθώς και τον τύπο Ethernet.

Γραμμή 2: Ελέγχει αν ο hook "down" δεν είναι NULL, δηλαδή αν υπάρχει συνδεδεμένος κόμβος.

Γραμμή 2.1: Ελέγχει αν το συνολικό μήκος του πακέτου είναι μικρότερο από το μέγεθος μιας Ethernet κεφαλίδας συν το μέγεθος μιας IP κεφαλίδας. Αυτό είναι απαραίτητο για να βεβαιωθεί ότι το πακέτο έχει αρκετό χώρο για να περιέχει και τα δύο headers.

Γραμμή 2.1.1: Αν το πακέτο δεν έχει αρκετό μήκος, καλεί τη συνάρτηση `m_pullup` για να τραβήξει περισσότερα δεδομένα και να εξασφαλίσει ότι το πακέτο έχει τουλάχιστον το μέγεθος των δύο κεφαλίδων (Ethernet και IP). Γραμμή 11: Αν το μήνυμα είναι NULL μετά την ανανέωση, απελευθερώνει τον πόρο item και επιστρέφει τον κωδικό λάθους ENOBUFS.

Γραμμές 2.1.2-2.1.4: Ελέγχει αν απέτυχε η κλήση του `m_pullup` και αν ισχύει απελευθερώνει το στοιχείο item του μηνύματος.

Γραμμή 2.3: Μετατρέπει τα δεδομένα του πακέτου σε δείκτη προς την δομή `ether_header` χρησιμοποιώντας τη μακροεντολή `mtod`.

Γραμμή 2.4: Ελέγχει αν ο τύπος του EtherType στην Ethernet κεφαλίδα είναι για IP πακέτα (ETHERTYPE\_IP).

Γραμμή 2.4.1: Υπολογίζει τον δείκτη για την IP κεφαλίδα προσθέτοντας το μέγεθος της Ethernet κεφαλίδας στην αρχή των δεδομένων.

Γραμμές 2.4.2-2.4.6: Ελέγχει το πρωτόκολλο στην IP κεφαλίδα για να δει τι τύπου είναι το πρωτόκολλο

Γραμμή 2.4.7: Εκτυπώνει τη διεύθυνση IP πηγής (ip\_src), τη διεύθυνση IP προορισμού (ip\_dst) και το πρωτόκολλο.

Πίνακας 5.8. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 6

```

1.1. else if (m->m_len >= sizeof(struct ip))

    1.1.1. ip_hdr = mtod(m, struct ip )
    1.1.2. switch (ip_hdr->ip_p) {
    1.1.3. case IPPROTO_TCP:
        1.1.3.1. protocol = "TCP";
        1.1.3.2. break;
    1.1.4. case IPPROTO_UDP:
        1.1.4.1. protocol = "UDP";
        1.1.4.2. break;
    1.1.5. default:
        1.1.5.1. protocol = "OTHER";
        1.1.5.2. break;
    1.1.6. }
    printf("DIRECT IP - SRC_IP: %d.%d.%d.%d, DST_IP: %d.%d.%d.%d, Protocol: %s\n",
        (ip_hdr->ip_src.s_addr & 0xFF), ((ip_hdr->ip_src.s_addr >> 8) & 0xFF),
        ((ip_hdr->ip_src.s_addr >> 16) & 0xFF), ((ip_hdr->ip_src.s_addr >> 24) & 0xFF),
        2. (ip_hdr->ip_dst.s_addr & 0xFF), ((ip_hdr->ip_dst.s_addr >> 8) & 0xFF),
        ((ip_hdr->ip_dst.s_addr >> 16) & 0xFF), ((ip_hdr->ip_dst.s_addr >> 24) & 0xFF),
        protocol);

    2.1.     }
    2.2. }

2.3. /* Update stats on incoming hook */
2.4. hinfo->stats.inOctets += m->m_pkthdr.len;
2.5. hinfo->stats.inFrames++;
2.6. /* Duplicate packet if required */
2.7. if (hinfo->dup) {
    2.7.1. struct mbuf *m2;

    2.7.2. /* Copy packet (failure will not stop the original)*/
    2.7.3. m2 = m_dup(m, M_NOWAIT);
    2.7.4. if (m2) {
        2.7.4.1. /* Deliver duplicate */
        2.7.4.2. h = hinfo->dup;
        2.7.4.3. NG_SEND_DATA_ONLY(error, h->hook, m2);
        2.7.4.4. if (error == 0) {
            2.7.4.4.1. h->stats.outOctets += m->m_pkthdr.len;
            2.7.4.4.2. h->stats.outFrames++;
        2.7.4.5. }else{
            2.7.4.5.1. h->stats.Errors++;//if duplication data delivery fail
            then increase
        2.7.4.6. }
    2.7.5. }

2.8. }
2.9. /* Deliver frame out destination hook */
2.10. if (hinfo->dest) {
    2.10.1. h = hinfo->dest;
    2.10.2. h->stats.outOctets += m->m_pkthdr.len;
    2.10.3. h->stats.outFrames++;
    2.10.4. NG_FWD_ITEM_HOOK(error, item, h->hook);
2.11. } else
    2.11.1. NG_FREE_ITEM(item);
2.12. nanouptime(&end);//end the timing
2.13. timespecsub(&end, &start, &diff);//call function for calculation
2.14. hinfo->stats.Delay += timespec_to_ns(&diff);//send the delay stat to the
        struct stats
2.15. return (error);
3. }
4. static int
5. ng_triple_tee_close(node_p node)
6. {
7. const sc_p privdata = NG_NODE_PRIVATE(node);

8. if (privdata->left.hook && privdata->right.hook)
9. ng_bypass(privdata->left.hook, privdata->right.hook);

```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο ng\_triple\_tee.c και απεικονίζεται στο πίνακα 5.7

Γραμμές 1.1-2.1: Ιδιά διαδικασία όπως επεξηγήθηκε για τον πίνακα 5.6 όμως σε αυτές τις γραμμές ελέγχεται άμα το πακέτο προέρχεται απευθείας από το επίπεδο δικτύου

Γραμμές 2.12-2.13: Τελειώνει η χρονομέτρηση και καλείται η συνάρτηση μετατροπής του χρόνου σε nanoseconds

Γραμμή 2.14: Αποθηκεύονται τα δεδομένα για την χρονομέτρηση στην δομή των στατιστικών

Πίνακας 5.9. Περιεχόμενα αρχείου ng\_triple\_tee.c μέρος 7

```
1. return (0);
2. }

4. /*
5. Shutdown processing
6. */
7. static int
8. ng_triple_tee_shutdown(node_p node)
9. {
10. const sc_p privdata = NG_NODE_PRIVATE(node);

11. NG_NODE_SET_PRIVATE(node, NULL);
12. free(privdata, M_NETGRAPH);
13. NG_NODE_UNREF(node);
14. return (0);
15. }

16. /*
17. Hook disconnection
18. */
19. static int
20. ng_triple_tee_disconnect(hook_p hook)
21. {
22. sc_p sc = NG_NODE_PRIVATE(NG_HOOK_NODE(hook));
23. hi_p const hinfo = NG_HOOK_PRIVATE(hook);

24. KASSERT(hinfo != NULL, ("%s: null info", __func__));
25. hinfo->hook = NULL;

26. /* Recalculate internal paths. */
27. if (sc->left.dest == hinfo) {
28. sc->left.dest = sc->left.dup;
29. sc->left.dup = NULL;
30. } else if (sc->left.dup == hinfo)
31. sc->left.dup = NULL;
32. if (sc->right.dest == hinfo) {
33. sc->right.dest = sc->right.dup;
34. sc->right.dup = NULL;
35. } else if (sc->right.dup == hinfo)
36. sc->right.dup = NULL;
37. if (sc->left2right.dest == hinfo)
38. sc->left2right.dest = NULL;
39. if (sc->right2left.dest == hinfo)
40. sc->right2left.dest = NULL;
41. if (sc->up.dest == hinfo)//add up
42. sc->up.dest = NULL;
43. if (sc->down.dest == hinfo)//add down
44. sc->down.dest = NULL;

45. /* Die when last hook disconnected. */
46. if ((NG_NODE_NUMHOOKS(NG_HOOK_NODE(hook)) == 0) &&
47. NG_NODE_IS_VALID(NG_HOOK_NODE(hook)))
48. ng_rmnnode_self(NG_HOOK_NODE(hook));
49. return (0);
50. }
```

Σύντομη επεξήγηση του κώδικα που προστέθηκε στο αρχείο ng\_triple\_tee.c και απεικονίζεται στο πίνακα 5.8

Γραμμές 41-44 : Κώδικας για την δυνατότητα αποσύνδεσης των αγκίστρων up και down θέτοντάς τα σε NULL.

Παρακάτω ακολουθούν δύο παραδείγματα χρήσης του κόμβου. Στο πρώτο ο κόμβος συνδέεται απευθείας με το επίπεδο δικτύου με τον κόμβο ipfw και κατά συνέπεια λαμβάνει απευθείας IP πακέτα .Στο δεύτερο παράδειγμα ο κόμβος συνδέεται με τον κόμβο ether επομένως λαμβάνει πλαίσια Ethernet. Τις πληροφορίες καταγραφής των up και down αγκίστρων μπορούμε να τις δούμε με την εντολή dmesg καθώς οι λειτουργία του κόμβου γίνονται στο kernel και δεν υπάρχει η δυνατότητα εγγραφής των πληροφοριών σε αρχείο.

Στο πίνακα 5. παρουσιάζεται το παράδειγμα χρήσης όταν ο κόμβος συνδέεται με τον κόμβο ipfw και λαμβάνει απευθείας (direct) IP πακέτα.

Πίνακας 5.8. Παράδειγμα χρήσης του κόμβου triple\_tee και καταγραφής κεφαλίδων από απευθείας IP πακέτα

```
+kldload ng_ipfw.ko
+kldload ng_triple_tee.ko
+ngctl mkpeer ipfw: triple_tee 60 left
+ngctl name ipfw:60 TRIPLE
+ngctl connect TRIPLE: ipfw: right 61
+ngctl mkpeer TRIPLE: hole down downHole
+ipfw add 200 netgraph 51 all from any to any
+ipfw add 150 netgraph 50 all from any to any
+sysctl net.inet.ip.fw.one_pass=0
+service ipfw restart
+dmesg
DIRECT SRC IP: 127.0.0.1, DST_IP: 10.0.39.255, Protocol: OTHER
DIRECT SRC IP: 0.0.0.0, DST_IP: 10.0.39.255, Protocol: OTHER
DIRECT SRC_IP: 10.0.2.15, DST_IP: 224.0.0.251, Protocol: UDP
DIRECT SRC_IP: 8.0.69.16, DST_IP: 1.72.0.0, Protocol: OTHER
DIRECT SRC_IP: 134.221.96.0, DST_IP: 0.0.0.44, Protocol: OTHER
DIRECT SRC_IP: 10.0.2.15, DST_IP: 224.0.0.251, Protocol: UDP
```

Όμως ο κόμβος έχει δυνατότητα καταγραφής και Ethernet πακέτων. Παρακάτω παρουσιάζεται ένα παράδειγμα χρήσης όταν ο κόμβος συνδέεται με τον κόμβο ether και λαμβάνει Ethernet πακέτα.

Πίνακας 5.9. Παράδειγμα χρήσης του κόμβου triple\_tee και καταγραφής κεφαλίδων από ether πακέτα

```

+klload ng_ether.ko
+klload ng_triple_tee.ko
+ngctl mkpeer em0: triple_tee lower right
+ngctl name em0: lower TRIPLE
+ngctl connect TRIPLE: em0: left upper
+ngctl mkpeer TRIPLE: hole up upHole
+ngctl name TRIPLE: up HOLE
+ngctl connect TRIPLE: HOLE: down downhole
+ngctl connect TRIPLE: HOLE: right2left r2lHole
+ngctl connect TRIPLE: HOLE: left2right l2rHole
+dmesg
SRC_MAC: 08:00:27:9c:2f:0b, DST_MAC: 52:54:00:12:35:02, EtherType: 0x0800
ETHER SRC_IP: 10.0.2.15, DST_IP: 192.168.1.1, Protocol: UDP
SRC_MAC: 52:54:00:12:35:02, DST_MAC: 08:00:27:9c:2f:0b, EtherType: 0x0800
ETHER SRC_IP: 192.168.1.1, DST_IP: 10.0.2.15, Protocol: UDP
SRC_MAC: 08:00:27:9c:2f:0b, DST_MAC: 52:54:00:12:35:02, EtherType: 0x0800
ETHER SRC_IP: 10.0.2.15, DST_IP: 88.99.137.18, Protocol: TCP
SRC_MAC: 52:54:00:12:35:02, DST_MAC: 08:00:27:9c:2f:0b, EtherType: 0x0800
ETHER SRC_IP: 88.99.137.18, DST_IP: 10.0.2.15, Protocol: TCP
SRC_MAC: 08:00:27:9c:2f:0b, DST_MAC: 52:54:00:12:35:02, EtherType: 0x0800
ETHER SRC_IP: 10.0.2.15, DST_IP: 88.99.137.18, Protocol: TCP

```

Ταυτόχρονα με την καταγραφή των πακέτων στα άγκιστρα up και down μπορεί κανείς να δεί με την εντολή getstats τα επιπλέον στατιστικά που προστέθηκαν, το Delay και το Errors. Βέβαια εδώ το στατιστικό Errors δεν εμφανίζεται καθώς δεν υπάρχουν σφάλματα επομένως είναι NULL.

**Πίνακας 5.10. Παρουσίαση στατιστικών από το παράδειγμα με τον κόμβο triple\_tee και επιβεβαίωσης λειτουργίας “ανίχνευσης” επιπλέον δεδομένων**

```

+ngctl msg TRIPLE: getstats
Rec'd response "getstats" (1) from "[4]:":
Args:{ right={ inOctets=80868 inFrames=148 outOctets=43816 outFrames=147 Delay=49775883 } left={ inOctets=43816 inFrames=147 outOctets=80868 outFrames=148 Delay=75261803 } right2left={ outOctets=80868 outFrames=148 } left2right={ outOctets=43816 outFrames=147 } }

```

Με τις προσθήκες αυτές ο κόμβος απέκτησε ένα πολύ δυνατό πλεονέκτημα σε εφαρμογές ανίχνευσης και παρακολούθησής δεδομένων(snooping). Καταγράφει όχι μόνο τον χρόνο και τα σφάλματα παράδοσης των διπλότυπων δεδομένων, αλλά ταυτόχρονα εξάγει σημαντικές πληροφορίες για αυτά.



# Κεφάλαιο 6. Βιβλιογραφία και αναφορές

- [1] J.Eischer, "layer2b" Available at <https://people.freebsd.org/~julian/layer2b.pdf> (last access Mar. 9, 2024)
- [2] A.Cobbs, "All about Netgraph" Available at <https://people.freebsd.org/~julian/netgraph.html> (last access Mar. 23, 2024)
- [3] A. Steinmann," Introduction to NETGRAPH on FreeBSD Systems" , AsiaBSDCon Tokyo, Mar 2011. Available at [https://www.netbsd.org/gallery/presentations/ast/2012\\_AsiaBSDCon/Tutorial\\_NETGRAPH.pdf](https://www.netbsd.org/gallery/presentations/ast/2012_AsiaBSDCon/Tutorial_NETGRAPH.pdf)
- [4] J.Eischer, "Netgraph in 5 and beyond" Available at <https://people.freebsd.org/~julian/BAFUG/talks/Netgraph/Netgraph.pdf> (last access Mar. 9, 2024)
- [5] A.Cobbs J. Elischer, "NETGRAPH Kernel Interfaces Manual" Available at [https://man.freebsd.org/cgi/man.cgi?netgraph\(4\)](https://man.freebsd.org/cgi/man.cgi?netgraph(4)) (last access Mar. 9, 2024)
- [6] V.Goncharov," Netgraph для пользователя: некоторые типы узлов" ,livejournal, Febr 2011. Available at <https://nuclight.livejournal.com/127034.html>
- [7]A.Cobbs J. Elischer, "NG\_ETHER Kernel Interfaces Manual",Jun 2011Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_ether&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ether&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html) (last access Mar. 9, 2024)
- [8] B.Davis, "NG\_GIF Kernel Interfaces Manual",Sep 2001 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_gif&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_gif&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html) (last access Mar. 9, 2024)

- [9] A.Cobbs A. Thompson, “NG\_TTY Kernel Interfaces Manual”,Dec 2008 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_tty&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_tty&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)  
(last access Mar. 9, 2024)
- [10] A.Cobbs, “NG\_IFACE Kernel Interfaces Manual”,Jul 2020 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_iface&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_iface&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)  
(last access Mar. 9, 2024)
- [11] G.Smirnoff,V.V.Belekhov, “NG{EIFACE Kernel Interfaces Manual” ,May 2019 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_eiface&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_eiface&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [12] G.Smirnoff M. Santcroos, “NG\_DEVICE Kernel Interfaces Manual”,Nov 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_device&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_device&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)  
(last access Mar. 9, 2024)
- [13] J. Elischer, “NG\_SOCKET Kernel Interfaces Manual” ,Jan 1999Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_socket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_socket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)  
(last access Mar. 9, 2024)
- [14] A.Cobbs, “NG\_KSOCKET Kernel Interfaces Manual”,Jan 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ksocket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ksocket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [15] M. Yevmenkin, “NG\_BT\_SOCKET Kernel Interfaces Manual”,Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_btsocket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_btsocket&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [16] A.Cobbs J. Elischer, “NG\_CISCO Kernel Interfaces Manual”,Jan 1999 Available at

[https://man.freebsd.org/cgi/man.cgi?query=ng\\_cisco&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_cisco&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[17] J. Elischer, “NG\_FRAME\_RELAY Kernel Interfaces Manual”, Jan 1999 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_frame\\_relay&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_frame_relay&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[18] B.Davis, “NG\_GIF\_DEMUX Kernel Interfaces Manual”, Sep 2001 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_gif\\_demux&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_gif_demux&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[19] A.Cobbs, “NG\_L2TP Kernel Interfaces Manual”, Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_l2tp&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_l2tp&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[20] J. Elischer, “NG\_LMI Kernel Interfaces Manual”, Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_lmi&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_lmi&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[21] A.Cobbs, “NG\_PPP Kernel Interfaces Manual”, Jun 2016 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ppp&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ppp&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[22] A.Cobbs, “NG\_MPPC Kernel Interfaces Manual”, Jun 2016 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_mppc&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_mppc&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[23] J. Elischer, “NG\_PPPOE Kernel Interfaces Manual”, May 2022 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_pppoe&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_pppoe&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[24] A.Cobbs , “NG\_PPTPGRE Kernel Interfaces Manual”, Nov 2018 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_pptpgre&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_pptpgre&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[25] R.Kurakin , “NG\_SPPP Kernel Interfaces Manual”, Mar 2004 Available at

[https://man.freebsd.org/cgi/man.cgi?query=ng\\_sppp&sektion=4&manpath=FreeBSD+5.3-RELEASE](https://man.freebsd.org/cgi/man.cgi?query=ng_sppp&sektion=4&manpath=FreeBSD+5.3-RELEASE)

[26] R.Ermilov, “NG\_VLAN Kernel Interfaces Manual”, Mar 2004 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_vlan&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_vlan&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[27] A.Popov G. Smirnoff, “NG\_TCPMSS Kernel Interfaces Manual”, June 2005 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_tcpmss&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_tcpmss&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[28] H.Brandt, “NG\_SSCFU Kernel Interfaces Manual”, Oct 2005 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_sscfu&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE](https://man.freebsd.org/cgi/man.cgi?query=ng_sscfu&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE)

[29] H.Brandt, “NG\_SSCOP Kernel Interfaces Manual”, Oct 2003 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_sscop&apropos=0&sektion=4&manpath=FreeBSD+11.4-RELEASE&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_sscop&apropos=0&sektion=4&manpath=FreeBSD+11.4-RELEASE&arch=default&format=html)

[30] J. Elischer, “NG\_RFC1490 Kernel Interfaces Manual”, Jan 1999 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_rfc1490&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_rfc1490&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[31] G.Smirnoff A. Chernikov A Motin R.Palagin, “NG\_NETFLOW Kernel Interfaces Manual”, Dec 2012 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_netflow&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_netflow&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[32] M.Yevmenkin, “NG\_L2CAP Kernel Interfaces Manual”, Jul 2002 Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_l2cap&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_l2cap&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[33] A.Cobbs J. Elischer, “NG\_DEFLATE Kernel Interfaces Manual” Available at [https://man.freebsd.org/cgi/man.cgi?query=ng\\_deflate&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_deflate&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

- [34] A.Motin “NG\_PRED1 Kernel Interfaces Manual”, Dec 2006 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_pred1&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_pred1&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [35] A.Cobbs, “NG\_BRIGDE Kernel Interfaces Manual”, May 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_bridge&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_bridge&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [36] R.Ermilov, “NG\_HUB Kernel Interfaces Manual”, May 2010 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_hub&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_hub&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [37] A.Cobbs R.R. Mulhuijzen, “NG\_ONE2MANY Kernel Interfaces Manual”, Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_one2many&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_one2many&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [38] J. Elischer, “NG\_TEE Kernel Interfaces Manual”, May 2004 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_tee&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_tee&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [39] V.V. Belekhov J. Elischer, “NG\_SPLIT Kernel Interfaces Manual”, Feb 2001 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_split&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_split&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [40] J. Elischer, “NG ETF Kernel Interfaces Manual”, Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_etf&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_etf&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [41] J. Elischer, “NG\_HOLE Kernel Interfaces Manual”, May 2004 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_hole&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_hole&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [42] J. Elischer, “NG\_ECHO Kernel Interfaces Manual”, Jan 1999 Available at

[https://man.freebsd.org/cgi/man.cgi?query=ng\\_echo&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_echo&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[43] J. Elischer, “NG\_ETHER\_ECHO Kernel Interfaces Manual”, Dec 2008 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ether\\_echo&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ether_echo&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[44] A.Cobbs, “NG\_ASYNC Kernel Interfaces Manual”, Nov 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_async&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_async&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[45] B.Rice, “NG\_ATMLLC Kernel Interfaces Manual”, Mar 2004 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_atmllc&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE](https://man.freebsd.org/cgi/man.cgi?query=ng_atmllc&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE)

[46] M. Yevmenkin, “NG\_BLUETOOTH Kernel Interfaces Manual”, Mar 2002 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_bluetooth&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_bluetooth&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[47] A.Cobbs, “NG\_BPF Kernel Interfaces Manual”, Sep 2020 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_bpf&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_bpf&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[48] A.Motin N. Artunes, “NG\_CAR Kernel Interfaces Manual”, Jan 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_car&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_car&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

[49] H. Brandt, “NG\_CCATM Kernel Interfaces Manual”, Mar 2005 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ccatm&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE](https://man.freebsd.org/cgi/man.cgi?query=ng_ccatm&sektion=4&apropos=0&manpath=FreeBSD+11.4-RELEASE)

[50] D. Vagin, “NG\_CHECKSUM Kernel Interfaces Manual”, Oct 2015 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_checksum&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_checksum&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

- [51] M. Yevmenkin, “NG\_CHECKSUM Kernel Interfaces Manual”, Jun 2002 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_hci&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_hci&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [52] B. Davis, “NG\_IP\_INPUT Kernel Interfaces Manual”, Sep 2001 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ip\\_input&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ip_input&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [53] G.Smirnoff, “NG\_IPFW Kernel Interfaces Manual”, Mar 2010 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ipfw&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ipfw&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [54] P.Nikander, “NG\_MACFILTER Kernel Interfaces Manual”, Dec 2018 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_macfilter&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_macfilter&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [55] G.Smirnoff, “NG\_NAT Kernel Interfaces Manual”, Jan 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_nat&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_nat&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [56] A.Cobbs, “NG\_PATCH Kernel Interfaces Manual”, May 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_patch&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_patch&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [57] L.Donnerhacke, “NG\_PIPE Kernel Interfaces Manual”, Oct 2019 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_pipe&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_pipe&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [58] D.Chapeskie, “NG\_SOURCE Kernel Interfaces Manual”, Jan 2021 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_source&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_source&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [59] V.Goncharov, “NG\_TAG Kernel Interfaces Manual”, Jun 2006 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_tag&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_tag&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)

- [60] M. Yevmenkin, “NG\_UBT Kernel Interfaces Manual”, Dec 2012 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_ubt&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_ubt&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [61] J. Elischer, “NG\_UI Kernel Interfaces Manual”, Jan 1999 Available at  
[https://man.freebsd.org/cgi/man.cgi?query=ng\\_UI&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html](https://man.freebsd.org/cgi/man.cgi?query=ng_UI&apropos=0&sektion=4&manpath=FreeBSD+14.0-RELEASE+and+Ports&arch=default&format=html)
- [62] D.Rabson “KLDLOAD System Manager's Manual” Available at  
[https://man.freebsd.org/cgi/man.cgi?kldload\(8\)](https://man.freebsd.org/cgi/man.cgi?kldload(8)) (last access Mar. 9, 2024)
- [63] S.C.Delgado, “Chapter 10. Configuring the FreeBSD Kernel” Available at  
<https://docs.freebsd.org/en/books/handbook/kernelconfig/> (last access Mar. 9, 2024)
- [64] A.Cobbs J. Elischer, “NETGRAPH Library Functions Manual” Available at  
[https://man.freebsd.org/cgi/man.cgi?netgraph\(3\)](https://man.freebsd.org/cgi/man.cgi?netgraph(3)) (last access Mar. 9, 2024)
- [65] A.Cobbs, “NGHOOK System Manager's Manual”, Jan 1999 Available at  
<https://man.freebsd.org/cgi/man.cgi?query=nghook&sektion=8&manpath=FreeBSD+4.6-RELEASE>
- [66] A.Cobbs, “NGCTL System Manager's Manual”, Jan 1999 Available at  
<https://man.freebsd.org/cgi/man.cgi?query=ngctl&sektion=8&manpath=FreeBSD+4.6-RELEASE>
- [67] I.Mrsi, “How I build a module” Available at  
<https://forums.freebsd.org/threads/how-i-built-a-module.76025> (last access Mar. 9, 2024)
- [68] J.Elischer, [http://fxr.watson.org/fxr/source/netgraph/ng\\_tee.c?v=FREEBSD-12-STABLE](http://fxr.watson.org/fxr/source/netgraph/ng_tee.c?v=FREEBSD-12-STABLE)  
Available at (last access Mar. 9, 2024)

[69] J.Elischer, [http://fxr.watson.org/fxr/source/netgraph/ng\\_tee.h?v=FREEBSD-12-STABLE](http://fxr.watson.org/fxr/source/netgraph/ng_tee.h?v=FREEBSD-12-STABLE)

Available at (last access Mar. 9, 2024)

## Παραρτήματα

### Παράρτημα 1. Επιπλέον παράδειγμα τα χρήσης του ngctl

#### Παράδειγμα 1

Το ακόλουθο σενάριο περιλαμβάνει τη δημιουργία ενός κόμβου ng\_ksocket, ο οποίος ρυθμίζεται για να λειτουργεί ως δέκτης διακοπτόμενων πακέτων (diverted packets) μέσω ενός συγκεκριμένου κόμβου i�푸. Πιο συγκεκριμένα ο κόμβος αυτός λαμβάνει το όνομα fred και στη συνέχεια λαμβάνει μια συγκεκριμένη διεύθυνση IP μέσω ενός μηνύματος ελέγχου που αποστέλλεται στον κόμβο. Στην πορεία γίνεται προσθήκη ενός

κόμβου ipfw για την ανακατεύθυνση (divert) των πακέτων από μια συγκεκριμένη IP προς οποιαδήποτε διεύθυνση, με σκοπό την παρακολούθηση των δεδομένων που μεταδίδονται από την εν λόγω διεύθυνση. Τα αποτελέσματα της διαδικασίας αυτής εμφανίζονται στην κονσόλα, δείχνοντας τα δεδομένα των πακέτων που έχουν ανακατευθυνθεί. Επιπλέον, το σενάριο παρέχει ένα παράδειγμα χρήσης του κόμβου ksocket για την πραγματοποίηση σύνδεσης TCP.

Αρχικά δημιουργείται ένας κόμβος ng\_ksocket της «οικογένειας» PF\_DIVERT και του τύπου SOCK\_RAW. Το foo είναι το όνομα του αγκίστρου στον κόμβο socket ενώ το inet/raw/divert είναι το όνομα του άγκιστρου στον κόμβο ksocket. Αυτό το άγκιστρο είναι υπεύθυνο για το τι είδους socket να θα δημιουργηθεί.

+ **mkpeer** ksocket foo divert/raw/0

Εδώ δίνεται στον κόμβο ksocket ένα παγκόσμιο(global) όνομα π.χ. fred.

+ **name** foo fred

Παρουσιάζεται παρακάτω και ένας χειροκίνητος τρόπος ονοματοδοσίας του κόμβου.

+ **msg** foo name { name="fred"}

Εδώ χρησιμοποιείται η μετατροπή μηνυμάτων ελέγχου ASCII σε δυαδικό σύστημα και το αντίστροφο. Γενικά το πρόγραμμα ngctl το κάνει αυτό αυτόματα όταν χρησιμοποιείτε η εντολή msg. Επιπλέον η σύνδεση της υποδοχής που σχετίζεται με τον κόμβο ksocket σε μια θύρα γίνεται στέλνοντας στον κόμβο ksocket ένα μήνυμα ελέγχου bind.

+ **msg** fred: bind inet/192.168.1.1

Το ksocket δέχεται αυθαίρετες δομές sockaddr, αλλά έχει επίσης ειδική υποστήριξη για τις οικογένειες πρωτοκόλλων PF\_LOCAL και PF\_INET. Γι' αυτό μπορούμε να καθορίσουμε το όρισμα struct sockaddr με την εντολή bind ως inet/192.168.1.1. Θα πρέπει να εφαρμοστεί αυτή η διαδικασία και για άλλες οικογένειες πρωτοκόλλων πέραν των PF\_INET και PF\_LOCAL. Επιπλέον μπορεί κανείς να παρατηρήσει την ισότητα "2=192". Αυτό γίνεται ώστε να μπορεί να γίνει παράληψη του αριθμού θύρας IP

```
+ msg fred: bind { family=2 len=16 data=[ 2=192 168 1 1 ] }
```

Αφού δεν έγινε ανάθεση συγκεκριμένης θύρας παραπάνω, για να δείτε τον αριθμό θύρας που πήρατε πρέπει να εκτελεστεί η getname. Ο αριθμός θύρας είναι ο 1029.

```
+ msg fred: getname
```

Rec'd response "getname" (5) from "fred":

Args: inet/192.168.1.1:1029

Με τις παρακάτω εντολές εφαρμόζεται εκτροπή σε πακέτα δεδομένων που προέρχονται από τη διεύθυνση IP 192.168.1.129.

```
+ ^Z
```

Suspended

```
$ ipfw add 100 divert 1029 ip from 192.168.1.129 to any
```

```
00100 divert 1029 ip from 192.168.1.129 to any
```

```
$ fg
```

Με την εντολή ping παρατηρούνται τα παρακάτω δεδομένα.

```
+
```

Rec'd data packet on hook "foo":

```
0000: 45 00 00 3c 57 00 00 00 20 01 bf ee c0 a8 01 81 E..<W... .....
```

```
0010: c0 a8 01 01 08 00 49 5c 03 00 01 00 61 62 63 64 .....I\....abcd
```

```
0020: 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 efgijklmnopqrst
```

```
0030: 75 76 77 61 62 63 64 65 66 67 68 69 uvwabcdefghijklmnoqrst
```

```
+
```

Rec'd data packet on hook "foo":

```
0000: 45 00 00 3c 58 00 00 00 20 01 be ee c0 a8 01 81 E..<X... .....
```

```
0010: c0 a8 01 01 08 00 48 5c 03 00 02 00 61 62 63 64 .....H\....abcd
```

```
0020: 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 efgijklmnopqrst
```

```
0030: 75 76 77 61 62 63 64 65 66 67 68 69 uvwabcdefghijklmnoqrst
```

```
+
```

Παρακάτω χρησιμοποιείται με την βοήθεια του κόμβου ksocket οποιοσδήποτε τύπος υποδοχής TCP. Γενικά μπορεί να χρησιμοποιηθεί οποιασδήποτε τύπος υποδοχής

```
+ mkpeer ksocket bar inet/stream/tcp
+ msg bar connect inet/192.168.1.33:13
ngctl: send msg: Operation now in progress
+
Rec'd data packet on hook "foo":
0000: 4d 6f 6e 20 4e 6f 76 20 32 39 20 31 37 3a 34 38 Mon Nov 29 17:48
0010: 3a 33 37 20 31 39 39 39 0d 0a :37 1999..
```

Μπορεί ακόμα να χρησιμοποιηθεί ως τύπος υποδοχής και ένα Unix domain

```
+ mkpeer ksocket bar local/stream/0
+ msg bar bind local://"tmp/bar.socket"
```

Ακολουθεί ένα παράδειγμα ενός πιο περίπλοκου argument μηνύματος ελέγχου ASCII. Αν κοιτάξετε στο/sys/netgraph/ng\_message.h, θα δείτε ότι ένας κόμβος ανταποκρίνεται σε ένα NGM\_LISTHOOKS με μια struct hooklist, η οποία περιέχει έναν πίνακα struct linkinfo

```
/* Structure used for NGM_LISTHOOKS */
struct linkinfo {
    char      ourhook[NG_HOOKSIZ]; /* hook name */
    char      peerhook[NG_HOOKSIZ]; /* peer hook */
    struct nodeinfo nodeinfo;
};

struct hooklist {
    struct nodeinfo nodeinfo; /* node information */
    struct linkinfo link[0]; /* info about each hook */
};
```

Στέλνοντας σε έναν κόμβο την εντολή listhooks χρησιμοποιώντας το ngctl, μπορούμε να δούμε αυτή τη δομή σε μορφή ASCII

```
+ msg bar bind local://"tmp/bar.socket"
+ msg bar listhooks
Rec'd response "listhooks" (7) from "bar":
Args: { nodeinfo={ type="ksocket" id=9 hooks=1 }
        linkinfo=[ { ourhook="local/stream/0" peerhook="bar"
        nodeinfo={ name="ngctl1327" type="socket" id=8 hooks=1 } } ] }
```

---

## Παράδειγμα 2

Αρχικά, δημιουργείται ένας κόμβος τύπου frame\_relay συνδεδεμένος στη σύγχρονη(synchronous) θύρα της κάρτας δικτύου, με σκοπό την υποστήριξη της τεχνολογίας Frame Relay για τη μεταφορά δεδομένων. Στη συνέχεια, προστίθεται ένας κόμβος διαχείρισης συνδέσμων (LMI) που συνδέεται με δύο αναγνωριστικά σύνδεσης διασύνδεσης δεδομένων (DLCI), ένας με αριθμό 0 και ένας με αριθμό 1023, για την αυτόματη διαμόρφωση. Επίσης, συνδέεται το DLCI σε έναν κόμβο που χειρίζεται την ενθυλάκωση πρωτοκόλλων ανάλογα με τις προδιαγραφές RFC1490. Τέλος, δημιουργείται μια διασύνδεση μεταξύ του προηγούμενου κόμβου και ενός δικτυακού κόμβου διεπαφής (interface node), ο οποίος εμφανίζεται ως ng0.

Με την παρακάτω εντολή δημιουργείτε ένας κόμβος τύπου frame\_relay και συνδέετε στη sync θύρα.

```
+ ngctl mkpeer ${CARD}: frame_relay rawdata downstream
```

Ακολουθεί η σύνδεση της έξοδου dlc1 του πολυπλέκτη/αποπολυπλέκτη σε ένα νέο κόμβο του πρωτοκόλλου διαχείρισης συνδέσεων

```
+ ngctl mkpeer ${CARD}:rawdata lmi dlc10 auto
```

Επίσης γίνεται η σύνδεση του dlc1 1023, καθώς χρειάζεται και τα δύο για να δοκιμάσει την αυτόματη διαμόρφωση. Το πρωτόκολλο διαχείρισης συνδέσεων είναι τώρα ενεργό και ανιχνεύει.

+ **ngctl mkpeer** \${CARD}:rawdata \${CARD}:rawdata.dlci0 dlci1023 auto1023

Συνδέεται το DLCI (κανάλι) που έχει εικχωρηθεί από π.χ. τον OTE σε έναν κόμβο, για να χειριστεί οποιαδήποτε ενθυλάκωση πρωτοκόλλου χρησιμοποιεί ο ομότιμος συνομιλητής. Σε αυτή την περίπτωση, χρησιμοποιείται η ενθυλάκωση RFC1490.

+ **ngctl mkpeer** \${CARD}:rawdata rfc1490 dlci\${DLCLI} downstream

Συνδέεται η έξοδος πρωτοκόλλου ip (inet) του πρωτοκόλλου μικ στην είσοδο ip (inet) ενός κόμβου διασύνδεσης (interface) του Netgraph (ο οποίος θα πρέπει να εμφανίζεται από το ifconfig ως ng0). Αν η διασύνδεση ng0 πρέπει να δημιουργηθεί, χρησιμοποιείται η εντολή mkpeer

+ **ngctl mkpeer** \${CARD}:rawdata.dlci\${DLCLI} iface inet inet

Αν το ng0 υπάρχει ήδη, χρησιμοποιείτε η εντολή CONNECT αντί για mkpeer. Στη συνέχεια, χρησιμοποιείτε το ifconfig στη διασύνδεση ng0.

+ **ngctl mkpeer** \${CARD}:rawdata.dlci\${DLCLI} ng0: inet inet

Μια παραλλαγή όλου αυτού του σεναρίου θα μπορούσε να είναι με την εντολή name για να το γίνει πιο ευανάγνωστο, αλλά δεν λειτουργεί αν υπάρχουν πολλαπλές γραμμές ή dlcis.

+ **ngctl mkpeer** \${CARD}: frame\_relay rawdata downstream

+ **ngctl name** \${CARD}: rawdata mux

+ **ngctl mkpeer** mux: lmi dlci0 auto0

+ **ngctl name** mux: dlci0 lmi

+ **ngctl connect** mux: lmi: dlci1023 auto1023

+ **ngctl mkpeer** mux: rfc1490 dlci\${DLCLI} downstream

+ **ngctl mux:** dlci\${DLCLI} protomux

+ **ngctl mkpeer** protomux: iface inet inet

---

## **Παράρτημα 2. Συναρτήσεις, Μέθοδοι και Κεφαλίδες**

### **Συναρτήσεις**

Οι συναρτήσεις καλούνται κυρίως απευθείας από τις μεθόδους που ορίζονται στους κόμβους αλλά και από άλλες συναρτήσεις ή μηνύματα ελέγχου παρέχοντας έτσι καλύτερη διαχείριση του δικτύου.

Επίσης τα header αρχεία netgraph.h και ng\_message.h που θα αναφερθούμε παρακάτω εκτενέστερα χρησιμοποιούν συναρτήσεις και μακροεντολές. Μερικές από τις συναρτήσεις είναι[2][4]:

```
int ng_send_data(hook_p hook, struct mbuf *m, meta_p meta)
```

Σκοπός: Η συνάρτηση ng\_send\_data είναι υπευθύνη για να αποστέλλει δεδομένα μέσω ενός συγκεκριμένου hook του Netgraph, χρησιμοποιώντας έναν buffer (mbuf) για τα δεδομένα και προαιρετικά μεταδεδομένα (meta)

Ανάλυση: Παραδίδει το mbuf m και τα σχετικά μεταδεδομένα meta από το hook hook και θέτει το error στον κωδικό σφάλματος που προκύπτει. Ένα από τα m και meta ή και τα δύο μπορεί να είναι NULL. Σε όλες τις περιπτώσεις, η ευθύνη για την απελευθέρωση των m και meta αίρεται όταν καλείται αυτή η συνάρτηση (ακόμη και αν υπάρχει σφάλμα), οπότε αυτές οι μεταβλητές θα πρέπει να οριστούν σε NULL μετά την κλήση (αυτό γίνεται αυτόματα αν χρησιμοποιήσετε τη μακροεντολή NG\_SEND\_DATA() αντί αυτής).

```
int ng_send_dataq(hook_p hook, struct mbuf *m, meta_p meta)
```

Σκοπός: Η συνάρτηση `ng_send_dataq` επιτελεί την ίδια λειτουργία με την `ng_send_data`.  
Ανάλυση: Το ίδιο με την `ng_send_data()`, με τη διαφορά ότι ο παραλήπτης κόμβος λαμβάνει τα δεδομένα μέσω της μεθόδου `rcvdataq()` αντί της μεθόδου `recvdata()`. Εάν ο τύπος του κόμβου δεν υπερισχύει της `rcvdataq()`, τότε η κλήση αυτής της μεθόδου είναι ισοδύναμη με την κλήση της `ng_send_data()`.

```
int ng_queue_data(hook_p hook, struct mbuf *m, meta_p meta)
```

Σκοπός: Η συνάρτηση `ng_queue_data` προσθέτει δεδομένα (πακέτο δεδομένων `m`) και προαιρετικά μεταδεδομένα (`meta`) στην ουρά ενός συγκεκριμένου `hook` (`hook`) για επεξεργασία ή μετάδοση.

Ανάλυση: Το ίδιο με την `ng_send_data()`, με τη διαφορά ότι αυτή η κλήση είναι ασφαλής από ένα πλαίσιο `non-splnet()`. Το `mbuf` και τα μεταδεδομένα θα μπουν σε ουρά και θα παραδοθούν αργότερα στη `splnet()`.

```
int ng_send_msg(node_p here, struct ng_mesg *msg, const char *address, struct ng_mesg **resp)
```

Σκοπός: Η συνάρτηση `ng_send_msg` στέλνει ένα μήνυμα μέσω του Netgraph από έναν κόμβο σε έναν άλλο, προαιρετικά περιμένοντας μια απάντηση.

Ανάλυση: Αποστέλλει το μήνυμα ελέγχου Netgraph που υποδεικνύεται από το `msg` από τον τοπικό κόμβο εδώ στον κόμβο που βρίσκεται στη διεύθυνση, η οποία μπορεί να είναι μια απόλυτη ή σχετική διεύθυνση. Εάν η `resp` δεν είναι `NULL` και ο κόμβος παραλήπτης επιθυμεί να επιστρέψει μια σύγχρονη απάντηση, θα θέσει την `*resp` να δείχνει σε αυτήν. Σε αυτή την περίπτωση, είναι ευθύνη του καλούντος κόμβου να επεξεργαστεί και να ελευθερώσει την `*resp`.

```
int ng_queue_msg(node_p here, struct ng_mesg *msg, const char *address)
```

**Σκοπός:** Η συνάρτηση `ng_queue_msg` θέτει σε ουρά ένα μήνυμα σε έναν κόμβο βάσει διεύθυνσης με σκοπό την επεξεργασία του.

**Ανάλυση:** Το ίδιο με την `ng_send_msg()`, με την διαφορά ότι αυτή η κλήση είναι ασφαλής από ένα πλαίσιο που δεν είναι `splnet()`. Το μήνυμα θα μπει σε ουρά και θα παραδοθεί αργότερα στη `splnet()`. Δεν είναι δυνατή η σύγχρονη απάντησης(reply).

### **int ng\_name\_node(node\_p node, const char \*name)**

**Σκοπός:** Η συνάρτηση `ng_name_node` δίνει ένα όνομα σε έναν κόμβο.

**Ανάλυση:** Εκχώρηση του γενικού ονόματος `name` στον κόμβο `node`. Το όνομα πρέπει να είναι μοναδικό. Αυτό καλείται συχνά μέσα από τους κατασκευαστές κόμβων για κόμβους που σχετίζονται με κάποια άλλη οντότητα του πυρήνα με όνομα, π.χ. μια συσκευή ή μια διασύνδεση. Η εκχώρηση ενός ονόματος σε έναν κόμβο αυξάνει τον αριθμό των αναφορών του κόμβου.

### **void ng\_cutlinks(node\_p node)**

**Σκοπός:** Διακόπτει όλες τις συνδέσεις (hooks) για τον κόμβο(`node`). Συνήθως αυτό καλείται κατά τον τερματισμό του κόμβου.

### **void ng\_unref(node\_p node)**

**Σκοπός:** Μειώνει τον αριθμό των αναφορών ενός κόμβου και ελευθερώνει τον κόμβο εάν ο αριθμός αυτός μηδενιστεί. Συνήθως καλείται στη μέθοδο `shutdown()` για να απελευθερώσει την αναφορά που δημιουργήθηκε από την `ng_make_node_common()`.

### **void ng\_unname(node\_p node)**

**Σκοπός:** Καταργεί το καθολικό όνομα που έχει εικωρηθεί στον κόμβο και μειώνει τον αριθμό των αναφορών. Εάν ο κόμβος δεν έχει όνομα, αυτή η συνάρτηση δεν έχει κανένα αποτέλεσμα. Θα πρέπει να καλείται στη μέθοδο `shutdown()` πριν από την απελευθέρωση του κόμβου (μέσω της `ng_unref()`).

### **NG\_SEND\_DATA(error, hook, m, meta)**

Σκοπός: Η συνάρτηση NG\_SEND\_DATA στέλνει δεδομένα (m) μέσω ενός συγκεκριμένου αγκίστρου του Netgraph, με προαιρετικά μεταδεδομένα (meta), και επιστρέφει έναν κωδικό σφάλματος (error) εάν η αποστολή αποτύχει.

Ανάλυση: Αυτή απλά καλεί την `ng_send_data()` και στη συνέχεια θέτει τα `m` και `meta` σε `NULL`. Η `m` και η `meta` μπορούν να είναι `NULL`, αν και πρέπει να είναι πραγματικές μεταβλητές (εξαιτίας του τρόπου λειτουργίας της μακροεντολής). Επίσης θεωρείτε πιο ασφαλη από την `ng_send_data()`

### **NG\_SEND\_DATAQ(error, hook, m, meta)**

Σκοπός: Η συνάρτηση NG\_SEND\_DATAQ στέλνει δεδομένα (m) μέσω ενός συγκεκριμένου αγκίστρου του Netgraph, με προαιρετικά μεταδεδομένα (meta), και επιστρέφει έναν κωδικό σφάλματος (error) εάν η αποστολή αποτύχει.

Ανάλυση: Αυτή απλά καλεί την `ng_send_dataq()` και στη συνέχεια θέτει τα `m` και `meta` σε `NULL`. Η `m` και η `meta` μπορούν να είναι `NULL`, αν και πρέπει να είναι πραγματικές μεταβλητές (εξαιτίας του τρόπου λειτουργίας της μακροεντολής). Επίσης θεωρείτε πιο ασφαλη από την `ng_send_dataq()`

### **NG\_FREE\_DATA(m, meta)**

Σκοπός: Η συνάρτηση NG\_FREE\_DATA(`m`, `meta`) απελευθερώνει τη μνήμη που έχει καταληφθεί από τα δεδομένα και τα μεταδεδομένα ενός μηνύματος

Ανάλυση: Απελευθερώνει τα `m` και `meta` και τα θέτει σε `NULL`. Η `m` και η `meta` μπορούν να είναι `NULL`, αν και πρέπει να είναι πραγματικές μεταβλητές (δεν μπορούν να είναι η σταθερά `NULL` λόγω του τρόπου λειτουργίας της μακροεντολής).

### **NG\_FREE\_META(meta)**

Σκοπός: Η συνάρτηση NG\_FREE\_META(`meta`) απελευθερώνει τη μνήμη που έχει καταληφθεί από τα μεταδεδομένα ενός μηνύματος

Ανάλυση: Η meta μπορεί να είναι NULL, αν και πρέπει να είναι πραγματική μεταβλητή (εξαιτίας του τρόπου λειτουργίας της μακροεντολής).

### **NG\_MKMESSAGE(msg, cookie, cmdid, len, how)**

Σκοπός: Η συνάρτηση NG\_MKMESSAGE(msg, cookie, cmdid, len, how) δημιουργεί ένα νέο μήνυμα Netgraph, ορίζοντας τον τύπο του μηνύματος, το μήκος του, και πώς θα διαχειριστεί η μνήμη του μηνύματος.

Ανάλυση: Το msg πρέπει να είναι τύπου struct ng\_mesg \*. Τα cookie και cmdid είναι το typecookie του μηνύματος και το ID της εντολής. Ο τρόπος δέσμευσης μνήμης how είναι είτε M\_WAIT ή M\_NOWAIT (είναι ασφαλέστερο να χρησιμοποιείται το M\_NOWAIT). Θέτει το msg σε NULL εάν η κατανομή μνήμης αποτύχει. Αρχικοποιεί το διακριτικό μηνύματος στο μηδέν.

### **NG\_MKRESPONSE(rsp, msg, len, how)**

Σκοπός: Η συνάρτηση NG\_MKRESPONSE(rsp, msg, len, how) δημιουργεί ένα μήνυμα response rsp σε ένα αρχικό μήνυμα msg, δεσμεύοντας για αυτό μνήμη με μέγεθος len και καθορίζοντας τον τρόπο δέσμευσης μνήμης how.

Ανάλυση: Αρχικοποιεί ένα νέο μήνυμα ελέγχου rsp που προορίζεται να αποτελέσει απάντηση στο msg. Η απάντηση θα έχει len bytes χώρου (το len θα πρέπει να είναι μηδέν αν το rsp δεν απαιτεί επιπλέον δεδομένα). Το msg θα πρέπει να είναι ένας δείκτης σε ένα υπάρχον struct ng\_mesg ενώ το rsp θα πρέπει να είναι τύπου struct ng\_mesg . Ο τρόπος δέσμευσης μνήμης how είναι είτε M\_WAIT ή M\_NOWAIT (είναι ασφαλέστερο να χρησιμοποιείται το M\_NOWAIT). Θέτει το msg σε NULL εάν η κατανομή μνήμης αποτύχει. Αρχικοποιεί το διακριτικό μηνύματος στο μηδέν.

### **NgMkSockNode(const char \*name, int \*csp,int \*dsp)**

Σκοπός: Δημιουργεί έναν νέο κόμβο τύπου socket στο Netgraph μαζί με συνδεδεμένα control και data sockets. Αν δοθεί ένα όνομα (που δεν είναι NULL), ο κόμβος λαμβάνει αυτό το παγκόσμιο όνομα. Οι παράμετροι csp και dsp ορίζονται στα νέα ανοιχτά control και data sockets αντίστοιχα. Επίσης, φορτώνει τον τύπο κόμβου socket KLD αν δεν έχει ήδη φορτωθεί.

### **NgNameNode(int cs, const char \*path, const char \*fmt, ...)**

Σκοπός: Αναθέτει ένα παγκόσμιο όνομα σε έναν κόμβο που προσδιορίζεται από τη διεύθυνση path.

### **NgSendMsg(int cs, const char \*path, int cookie, int cmd, const void \*arg, size\_t arglen);**

Σκοπός: Αποστέλλει ένα δυαδικό μήνυμα ελέγχου από το socket node που συνδέεται με το control socket cs προς τον κόμβο που προσδιορίζεται από τη διεύθυνση path. Οι παράμετροι cookie και cmd καθορίζουν την εντολή και τα πρόσθετα δεδομένα που ορίζονται από το arg και το arglen.

### **NgSendReplyMsg(int cs, const char \*path, struct ng\_mesg \*msg, const void \*arg, size\_t arglen)**

Σκοπός: Αποστέλλει απάντηση σε ένα προηγουμένως ληφθέν μήνυμα ελέγχου. Η αρχική κεφαλίδα του μηνύματος πρέπει να δείχνει στην τοποθεσία που δείχνει η μεταβλητή msg.

### **NgSendAsciiMsg(int cs, const char \*path, const char \*fmt, ...)**

Σκοπός: Λειτουργεί όπως η NgSendMsg(), αλλά υποστηρίζει την κωδικοποίηση των μηνυμάτων ελέγχου σε ASCII. Διαμορφώνει την είσοδο όπως η printf και στη συνέχεια αποστέλλει το ASCII string που προέκυψε στον κόμβο.

### **NgRecvMsg(int cs, struct ng\_mesg \*rep, size\_t replen, char \*path)**

Σκοπός: Διαβάζει το επόμενο μήνυμα ελέγχου που λαμβάνεται από τον κόμβο που συνδέεται με το control socket cs. Εάν η διεύθυνση path δεν είναι NULL, γεμίζει ένα buffer με τη διεύθυνση του κόμβου που έστειλε το μήνυμα.

### **NgAllocRecvMsg(int cs, struct ng\_mesg \*\*rep, char \*path)**

**Σκοπός:** Λειτουργεί όπως η NgRecvMsg(), αλλά δυναμικά διαθέτει buffer για το μήνυμα για να εξασφαλίσει ότι το μήνυμα δεν θα περικοπεί. Το μέγεθος του buffer είναι ίσο με το μέγεθος του buffer λήψης του socket.

#### **NgRecvAsciiMsg(int cs, struct ng\_mesg \*rep, size\_t replen, char \*path);**

**Σκοπός:** Λειτουργεί όπως η NgRecvMsg(), αλλά μετατρέπει τυχόν δυαδικά επιχειρήματα σε ASCII αποστέλλοντας ένα αίτημα NGM\_BINARY2ASCII πίσω στον κόμβο προέλευσης.

#### **NgAllocRecvAsciiMsg(int cs, struct ng\_mesg \*\*rep, char \*path)**

**Σκοπός:** Λειτουργεί όπως η NgRecvAsciiMsg(), αλλά δυναμικά διαθέτει buffer για το μήνυμα για να εξασφαλίσει ότι το μήνυμα δεν θα περικοπεί.

#### **NgSendData(int ds, const char \*hook, const u\_char \*buf, size\_t len)**

**Σκοπός:** Αποστέλλει ένα πακέτο δεδομένων μέσω του συγκεκριμένου γάντζου ενός κόμβου που αντιστοιχεί στο data socket ds. Ο κόμβος πρέπει ήδη να είναι συνδεδεμένος με κάποιον άλλο κόμβο μέσω αυτού του αγκίστρου.

#### **NgRecvData(int ds, u\_char \*buf, size\_t len, char\*hook)**

**Σκοπός:** Διαβάζει το επόμενο πακέτο δεδομένων που λαμβάνεται από τον κόμβο που αντιστοιχεί στο data socket ds και το αποθηκεύει στο buffer.

#### **NgAllocRecvData(int ds, u\_char \*\*buf, char \*hook)**

**Σκοπός:** Λειτουργεί όπως η NgRecvData(), αλλά δυναμικά διαθέτει buffer για το πακέτο δεδομένων για να εξασφαλίσει ότι το πακέτο δεν θα περικοπεί.

#### **NgSetDebug(int level)**

**Σκοπός:** Η NgSetDebug() ορίζει το επίπεδο αποσφαλμάτωσης και επιστρέφει την προηγούμενη ρύθμιση.

#### **NgSetErrLog(void(\*log)(const char \*fmt,...),void(\*logx)(const char \*fmt, ...))**

**Σκοπός:** Η NgSetErrLog() καθορίζει τις συναρτήσεις καταγραφής για τα μηνύματα αποσφαλμάτωσης και λάθους.

---

## Μέθοδοι

Κάθε κόμβος υλοποιεί τις μεθόδους που ορίζονται στο struct ng\_type. Μέσα σε αυτές υλοποιούνται ικλήσεις συναρτήσεων και επεξεργάζονται πακέτα ή γίνονται λειτουργίες που αφορούν την διασύνδεση του κόμβου με κάποιον άλλο κόμβο. Όλοι οι παρακάτω μέθοδοι πρέπει να βρίσκονται σε όλου τους κόμβους του συστήματος και συγκεκριμένα να ορίζονται στο struct ng\_type του κάθε κόμβου.

### **int constructor(node\_p \*node)**

Σκοπός: Αρχικοποίηση ενός νέου κόμβου καλώντας την ng\_make\_node\_common() και θέτοντας το node->private αν χρειάζεται. Η αρχικοποίηση ανά κόμβο και η κατανομή μνήμης θα πρέπει να συμβαίνουν εδώ. Η ng\_make\_node\_common() θα πρέπει να καλείται πρώτη ( δημιουργεί τον κόμβο και θέτει τον αριθμό των αναφορών σε ένα). Στο default mode καλεί την ng\_make\_node\_common() και άμα χρειαστεί να γίνει συγκεκριμένη αρχικοποίηση κόμβου ή κατανομή πόρων τότε πρέπει να γίνει override της μεθόδου.

### **int rcvmsg(node\_p node, struct ng\_mesg \*msg, const char \*retaddr, struct ng\_mesg \*\*resp)**

Σκοπός: Λήψη και χειρισμός ενός μηνύματος ελέγχου. Η διεύθυνση του αποστολέα βρίσκεται στο retaddr. Η συνάρτηση rcvmsg() είναι υπεύθυνη για την αποδέσμευση του msg. Η απάντηση, εάν υπάρχει, μπορεί να επιστραφεί συγχρονισμένα εάν resp != NULL θέτοντας \*resp να δείχνει σε αυτήν. Τα γενικά μηνύματα ελέγχου (εκτός από το NGM\_TEXT\_STATUS) τα διαχειρίζεται το βασικό σύστημα και δεν χρειάζεται να γινει η διαχειριση τους εδώ. Στο default mode διαχειρίζεται όλα τα γενικά μηνύματα ελέγχου, διαφορετικά επιστρέφει EINVAL και εάν ορίζετε ειδικά μηνύματα ελέγχου τύπου ή θέλετε να υλοποιήσετε μηνύματα ελέγχου που ορίζονται από κάποιον άλλο τύπο κόμβου, τότε πρέπει να γίνει override της μεθόδου.

### **int shutdown(node\_p node)**

Σκοπός: Τερματισμός λειτουργίας του κόμβου. Θα πρέπει να αποσυνδέσει όλα τα άγκιστρα καλώντας την `ng_cutlinks()`, να απελευθερώσει όλη την ιδιωτική μνήμη ανά κόμβο, να απελευθερώσει το όνομα που έχει εικωρηθεί (αν υπάρχει) μέσω της `ng_unname()` και να απελευθερώσει τον ίδιο τον κόμβο καλώντας την `ng_unref()` (αυτή η κλήση απελευθερώνει την αναφορά που προστέθηκε από την `ng_make_node_common()`). Στην περίπτωση των μόνιμων κόμβων, όλα τα άγκιστρα θα πρέπει να αποσυνδεθούν και η σχετική συσκευή (ή οτιδήποτε άλλο) να μηδενιστεί, αλλά ο κόμβος δεν θα πρέπει να αφαιρεθεί (δηλ. να καλέσετε μόνο την `ng_cutlinks()`). Στο default mode καλεί τις `ng_cutlinks()`, `ng_unname()` και `ng_unref()` και όταν πρέπει να αναιρέσετε τα πράγματα που κάνατε στη `constructor` μέθοδο τότε πρέπει να γίνει `override` της μεθόδου.

### **int newhook(node\_p node, hook\_p hook, const char \*name)**

Σκοπός: Επικύρωση της σύνδεσης ενός αγκίστρου και αρχικοποίηση τυχόν πόρων ανά `hook`. Ο κόμβος θα πρέπει να επαληθεύσει ότι το όνομα του αγκίστρου είναι πράγματι ένα από τα ονόματα αγκίστρων που υποστηρίζονται από αυτόν τον τύπο κόμβου. Η μοναδικότητα του ονόματος θα έχει ήδη επαληθευτεί. Εάν το άγκιστρο απαιτεί πληροφορίες ανά `hook`, αυτή η μέθοδος θα πρέπει να αρχικοποίηση ανάλογα το `hook->private`. Στο default mode η σύνδεση του `hook` γίνεται πάντα αποδεκτή.

### **hook\_p findhook(node\_p node, const char \*name)**

Σκοπός: Εύρεση ενός συνδεδεμένου `hook` σε αυτόν τον κόμβο. Δεν είναι απαραίτητο να υπερισχύσει αυτή η μέθοδος, εκτός αν ο κόμβος υποστηρίζει μεγάλο αριθμό αγκίστρων, όπου η γραμμική αναζήτηση θα ήταν πολύ αργή. Στο default mode πραγματοποιεί γραμμική αναζήτηση στη λίστα των `hook` που είναι συνδεδεμένα σε αυτόν τον κόμβο και όταν ο κόμβος σας υποστηρίζει μεγάλο αριθμό ταυτόχρονα συνδεδεμένων αγκίστρων ο κόμβος τότε πρέπει να γίνει `override` της μεθόδου.

### **int connect(hook\_p hook)**

Σκοπός: Τελική επαλήθευση της σύνδεσης του `hook`. Αυτή η μέθοδος δίνει στον κόμβο μια τελευταία ευκαιρία να επικυρώσει ένα πρόσφατα συνδεδεμένο άγκιστρο. Για

παράδειγμα, ο κόμβος μπορεί πραγματικά να ενδιαφέρεται για το με ποιον είναι συνδεδεμένος. Εάν αυτή η μέθοδος επιστρέψει σφάλμα, η σύνδεση διακόπτεται. Στο default mode η σύνδεση του hook γίνεται αποδεκτή.

#### **int rcvdata(hook\_p hook, struct mbuf \*m, meta\_p meta)**

Σκοπός: Λήψη ενός εισερχόμενου πακέτου δεδομένων σε ένα συνδεδεμένο άγκιστρο. Ο κόμβος είναι υπεύθυνος για την απελευθέρωση του mbuf αν επιστρέψει σφάλμα ή αν επιθυμεί να απορρίψει το πακέτο δεδομένων. Αν και δεν συμβαίνει προς το παρόν, στο μέλλον θα μπορούσε να συμβεί ότι μερικές φορές το m == NULL (για παράδειγμα, αν υπάρχει μόνο ένα meta προς αποστολή), οπότε οι τύποι κόμβων θα πρέπει να χειρίζονται αυτή την πιθανότητα. Στο default απορρίπτει το πακέτο δεδομένων και τα μεταδεδομένα και αν δεν σκοπεύετε να απορρίψετε όλα τα λαμβανόμενα πακέτα δεδομένων τότε πρέπει να κάνετε override την μέθοδο.

#### **int rcvdataq(hook\_p hook, struct mbuf \*m, meta\_p meta)**

Σκοπός: Αναμονή ενός εισερχόμενου πακέτου δεδομένων για λήψη σε ένα συνδεδεμένο hook. Ο κόμβος είναι υπεύθυνος για την απελευθέρωση του mbuf αν επιστρέψει σφάλμα ή αν επιθυμεί να απορρίψει το πακέτο δεδομένων.

Η πρόθεση εδώ είναι ότι ορισμένοι κόμβοι μπορεί να θέλουν να στέλνουν δεδομένα χρησιμοποιώντας έναν μηχανισμό ουράς αντί ενός λειτουργικού μηχανισμού. Αυτό απαιτεί τη συνεργασία του τύπου του κόμβου λήψης, ο οποίος πρέπει να υλοποιήσει αυτή τη μέθοδο για να κάνει κάτι διαφορετικό από την rcvdata(). Στο default καλεί τη μέθοδο rcvdata() και πρέπει να κάνετε override την μέθοδο μόνο αν έχετε λόγο να μεταχειρίζεστε διαφορετικά τα εισερχόμενα δεδομένα 'queue' από τα εισερχόμενα δεδομένα 'non-queue'.

#### **int disconnect(hook\_p hook)**

Σκοπός: Ειδοποίηση στον κόμβο ότι αποσυνδέεται ένα hook . Ο κόμβος θα πρέπει να απελευθερώσει τυχόν πόρους ανά άγκιστρο που διατέθηκαν κατά τη διάρκεια της connect(). Παρόλο που αυτή η συνάρτηση επιστρέφει int, στην πραγματικότητα θα πρέπει να επιστρέψει void επειδή η τιμή επιστροφής αγνοείται, η αποσύνδεση του hook δεν μπορεί να μπλοκαριστεί από έναν κόμβο. Αυτή η συνάρτηση θα πρέπει να ελέγχει αν το τελευταίο hook έχει αποσυνδεθεί (hook->node->numhooks == 0) και αν ναι, να καλεί

την `ng_rmnode()` για να αυτοκαταστραφεί, όπως συνηθίζεται. Αυτό βοηθά στην αποφυγή εντελώς μη συνδεδεμένων κόμβων που παραμένουν στο σύστημα μετά το τέλος της εργασίας τους. Στο default mode δεν κάνει τίποτα ενώ πρέπει να κάνετε override την μέθοδο σχεδόν πάντα .

### **int mod\_event(module\_t mod, int what, void \*arg)**

Σκοπός: Χειρισμός των συμβάντων φόρτωσης και εκφόρτωσης του τύπου κόμβου. Σημειώστε ότι και τα δύο συμβάντα αντιμετωπίζονται μέσω αυτής της μιας μεθόδου, η οποία διακρίνεται από το what που είναι είτε MOD\_LOAD είτε MOD\_UNLOAD. Η παράμετρος arg είναι ένας δείκτης στη δομή struct ng\_type που ορίζει τον τύπο κόμβου. Αυτή η μέθοδος δεν θα κληθεί ποτέ για το MOD\_UNLOAD όταν υπάρχουν κόμβοι αυτού του τύπου αυτή τη στιγμή. Επί του παρόντος, το Netgraph θα προσπαθήσει να κάνει MOD\_UNLOAD έναν τύπο κόμβου μόνο όταν κληθεί ρητά η kldunload. Στο default mode δεν κάνει τίποτα απλά αν δεν παρακαμφθεί, οι MOD\_LOAD και MOD\_UNLOAD θα τρέξουν κανονικά. Επίσης πρέπει να κάνετε override την μέθοδο μόνο αν ο τύπος σας χρειάζεται να κάνει οποιαδήποτε ειδική για τον τύπο αρχικοποίηση ή κατανομή πόρων κατά τη φόρτωση, ή να αναιρέσει οποιαδήποτε από αυτά κατά την εκφόρτωση.

---

## **Κεφαλίδες**

Τα αρχεία κεφαλίδας παίζουν σημαντικό ρόλο στη ομαλή λειτουργία του Netgraph καθώς ορίζουν τις δομές δεδομένων που χρησιμοποιεί το Netgraph για να αναπαραστήσει κόμβους, συνδέσεις, και μηνύματα ελέγχου. Ακόμη περιέχουν δηλώσεις των συναρτήσεων, ορισμούς μακροεντολών καθώς και ορισμούς διαφόρων σταθερών.

Υπάρχουν δύο αρχεία κεφαλίδας που περιλαμβάνουν όλοι οι τύποι κόμβων[2][4].

Το netgraph.h ορίζει τις βασικές δομές του Netgraph. Αυτές οι δομές χρησιμοποιούνται για την οργάνωση και τη διαχείριση των κόμβων και των συνδέσεων μέσα στο δίκτυο. Η μνήμη που καταναλώνουν αυτές οι δομές απελευθερώνεται μέσω της κλήσης της

συνάρτησης `ng_unref()`, η οποία μειώνει τον μετρητή αναφοράς των δομών και όταν αυτός φτάσει στο μηδέν, η μνήμη αποδεσμεύεται.

Το `ng_message.h` ορίζει δομές και μακροεντολές σχετικές με το χειρισμό μηνυμάτων ελέγχου και συγκεκριμένα ορίζει το `struct ng_mesg` που κάθε μήνυμα ελέγχου έχει ως πρόθεμα. Ακόμη χρησιμοποιείται για όλα τα γενικά μηνύματα ελέγχου, τα οποία έχουν typecookie `NGM_GENERIC_COOKIE` δηλαδή προκαθορισμένα μηνύματα που λειτουργούν για οποιοδήποτε κόμβο, καθώς υποστηρίζονται άμεσα από το ίδιο το πλαίσιο και παρατίθενται παρακάτω

Πίνακας παραρτήματος 2.1. Γενικά μηνύματα ελέγχου

Όνομα μηνύματος	Σκοπός
<code>NGM_SHUTDOWN</code>	Αποσύνδεση όλων των <code>hooks</code> του κόμβου και αφαίρεση του κόμβο
<code>NGM_MKPEER</code>	Δημιουργία νέου κόμβου και σύνδεση με αυτόν
<code>NGM_CONNECT</code>	Σύνδεση του άγκιστρου ενός κόμβου σε έναν άλλο κόμβο
<code>NGM_NAME</code>	Εκχώρηση ονόματος στον κόμβο
<code>NGM_RMHOOK</code>	Διακόπτει μια σύνδεση μεταξύ του κόμβου και ενός άλλου κόμβου
<code>NGM_NODEINFO</code>	Λήψη πληροφοριών σχετικά με τον κόμβο
<code>NGM_LISTHOOKS</code>	Λήψη λίστας όλων των συνδεδεμένων <code>hooks</code> στον κόμβο μας
<code>NGM_LISTNAMES</code>	Λήψη λίστας όλων των κόμβων που έχουν όνομα
<code>NGM_LISTNODES</code>	Λήψη λίστας όλων των κόμβων που είτε έχουν όνομα είτε δεν έχουν

<b>NGM_LISTTYPES</b>	Λήψη λίστας όλων των εγκατεστημένων κόμβων
<b>NGM_TEXT_STATUS</b>	Λήψη μιας αναγνώσιμης από τον άνθρωπο αναφοράς κατάστασης για κάποιο κόμβο
<b>NGM_BINARY2ASCII</b>	Μετατροπή ενός μηνύματος ελέγχου από δυαδικό σε ASCII.
<b>NGM_ASCII2BINARY</b>	Μετατροπή μηνύματος ελέγχου από ASCII σε δυαδικό.

## Παράρτημα 3. Επεξήγηση σημαντικών τύπων κόμβων

### *ng\_ether*

Ο κόμβος ether είναι αρμόδιος για να αλληλοεπιδρά ένα ήδη υπάρχων interface Ethernet με το υποσύστημα Netgraph. Επιτρέπει τον χειρισμό και την παρακολούθηση της κυκλοφορίας Ethernet που διέρχεται από τη διασύνδεση(interface). Όταν το module ng\_ether φορτωθεί μέσα στον πυρήνα, τότε δημιουργείται αυτόματα ένας κόμβος για κάθε διασύνδεση μέσα στο σύστημα

Η συνδεσιμότητα τους κόμβου μέσα στο δίκτυο γίνεται με την βοήθεια τριών αγκίστρων:

1)Lower: Σύνδεση με το κατώτερο επίπεδο συνδέσμου της συσκευής. Μέσα από αυτό το άγκιστρο περνάνε όλα τα εισερχόμενα πακέτα απευθείας από την συσκευή εκτός από αυτά που απευθύνονται άμεσα στα ανώτερα επίπεδα για προώθηση και επεξεργασία από το πυρήνα(kernel) μέσω του upper άγκιστρο.

2)Upper: Σύνδεση με το πρωτόκολλα των ανώτερων επίπεδων. Μέσα από αυτό το άγκιστρο περνάνε όλα τα εξερχόμενα πακέτα εκτός από αυτά που μεταδίδονται από την συσκευή ,άρα απευθύνονται στο lower άγκιστρο.

3)Orphans: Ιδια λειτουργία με το lower άγκιστρο, αλλά ως εισερχόμενα πακέτα έχει μόνο τα πακέτα που δεν έχουν αναγνωριστεί από το upper άγκιστρο δηλαδή έχουν απορριφθεί εν συντομίᾳ και το orphan άγκιστρο δίνει μια δεύτερη ευκαιρία.

### ***ng\_iface***

Ο κόμβος iface είναι ταυτόχρονα ένας κόμβος Netgraph και ένα εικονικό δικτυακό interface . Όταν δημιουργείται ένας κόμβος iface, εμφανίζεται μια διασύνδεση(interface ng0, ng1 κ.λπ.) που είναι προσβάσιμη μέσω της εντολής ifconfig. Όταν ένας κόμβος απενεργοποιείται, η αντίστοιχη διασύνδεση αφαιρείται και το όνομα της διασύνδεση γίνεται διαθέσιμο για επαναχρησιμοποίηση από μελλοντικούς κόμβους iface. Επίσης η διασύνδεση που δημιουργείται μπορεί να διαμορφωθεί ως broadcast ή point-to-point μέσω της ifconfig. Η διαμόρφωση μπορεί να αλλάξει μόνο όταν η διασύνδεση(interface) είναι εκτός λειτουργίας. Η προεπιλεγμένη λειτουργία είναι point-to-point

Η συνδεσιμότητα τους κόμβου μέσα στο δίκτυο γίνεται με την βοήθεια του ενός μόνου αγκίστρου που αντιστοιχεί σε κάθε υποστηριζόμενο πρωτόκολλο. Τα πρωτόκολλα που υποστηρίζονται επί του παρόντος είναι IP και IPv6 και τα άγκιστρα ονομάζονται αντίστοιχα inet για το IP και inet6 για το IPv6.

### ***ng\_hole***

Ο κόμβος hole χρησιμοποιείται για έλεγχο και αποσφαλμάτωση(test/debug) απορρίπτοντας οποιαδήποτε δεδομένα ή μηνύματα ελέγχου λαμβάνει.

Η συνδεσιμότητα του κόμβου γίνεται με οποιαδήποτε άγκιστρο με μόνο περιορισμό την μοναδικότητά του ονόματος του άγκιστρου.

### ***ng\_echo***

Ο κόμβος echo χρησιμοποιείται για έλεγχο και αποσφαλμάτωση(test/debug) ανακλώντας πίσω στον αποστολέα οποιαδήποτε δεδομένα ή μηνύματα ελέγχου λαμβάνει από τον ίδιο.

Η συνδεσιμότητα του κόμβου γίνεται με οποιαδήποτε άγκιστρο με μόνο περιορισμό την μοναδικότητα του ονόματος του άγκιστρου.

## ***ng\_ipfw***

Ο κόμβος ipfw χρησιμοποιείται για την υλοποίηση της διασύνδεσης μεταξύ των υποσυστημάτων ipfw(IP firewall) και Netgraph. Μόλις φορτωθεί το ipfw στον πυρήνα, δημιουργείται ένας μόνο κόμβος και δεν μπορούν να δημιουργηθούν άλλοι κόμβοι ipfw. Όταν καταστραφεί, ο μόνος τρόπος για να ξαναδημιουργηθεί ο κόμβος είναι η επαναφόρτωση της μονάδας ipfw. Τα πακέτα μπορούν να εισαχθούν στο Netgraph χρησιμοποιώντας είτε την εντολε'η (ipfw add (rule) netgraph (hookname)all from any to any in via fxp0) είτε με τις εντολές ngtee του βοηθητικού προγράμματος ipfw. Εάν δεν υπάρχει ταύτιση με κάποιο άγκιστρο, τα πακέτα απορρίπτονται.

Η συνδεσιμότητα του κόμβου γίνεται με οποιαδήποτε πλήθος άγκιστρων με την προϋπόθεση ότι το όνομα του αγκίστρου αποτελείται από μόνο αριθμητικούς χαρακτήρες.

## ***ng\_bridge***

Ο κόμβος bridge είναι υπεύθυνος για την σύνδεση ενός ή περισσοτέρων Ethernet συνδέσμων(links) όπου κάθε σύνδεση είναι ουσιαστικά ένα άγκιστρο στο οποίο μεταφέρονται ακατέργαστα(raw) πλαίσια Ethernet. Μέσω αυτής της διαδικασίας ο bridge κόμβος γνωρίζει σε ποιο σύνδεσμο αντιστοιχεί κάθε host. Ενώ το hub προωθεί ένα πακέτο σε κάθε σύνδεσμο στο εκάστοτε δίκτυο, το bridge προωθεί αποκλειστικά το πακέτο στον αρμόδιο σύνδεσμο και μόνο γλυτώνοντας έτσι τον φόρτο στους υπόλοιπους συνδέσμους άρα και το κόστος.

Η συνδεσιμότητα του κόμβου με το δίκτυο μπορεί να γίνει θεωρητικά με «άπειρα» άγκιστρα όπου κάθε άγκιστρο ονομάζεται linkX(link0,link1,κλπ) και αναπαριστά ένα bridged σύνδεσμο όπου συνδέονται με τα lower άγκιστρα από έναν κόμβο ng\_ether και στο ίδιο κόμβο ng\_ether συνδέεται στο upper άγκιστρο ο host του δικτύου. Ακόμη υπάρχει η υποπερίπτωση αντί για το συνηθισμένο όνομα linkX να δώσετε το όνομα uplinkX. Η χρήση αυτού του ονόματος αποσκοπεί στο ότι ο κόμβος να διατηρεί μικρούς εσωτερικούς πίνακες διευθύνσεων καθώς δεν διατηρεί διευθύνσεις MAC.