ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΒΕΛΤΙΣΤΟΥ ΣΧΕΔΙΑΣΜΟΥ ΔΙΚΤΥΩΝ - NETMODE

Ενσωμάτωση εικονικών δικτύων σε φυσικά δικτυακά υποστρώματα με χρήση τεχνικών αποδοτικής ανάθεσης πόρων και ευριστικής χαρτογράφησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΒΑΣΙΛΕΙΟΥ Π. ΚΟΤΡΩΝΗ

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΒΕΛΤΙΣΤΟΥ ΣΧΕΔΙΑΣΜΟΥ ΔΙΚΤΥΩΝ - NETMODE

Ενσωμάτωση εικονικών δικτύων σε φυσικά δικτυακά υποστρώματα με χρήση τεχνικών αποδοτικής ανάθεσης πόρων και ευριστικής χαρτογράφησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΒΑΣΙΛΕΙΟΥ Π. ΚΟΤΡΩΝΗ

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουνίου 2011.

.....	.....	.....
Μάγκλαρης Βασίλειος	Παπαβασιλείου Συμεών	Καλογεράς Δημήτριος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Διδάκτωρ Ε.Μ.Π.

Αθήνα, Ιούνιος 2011

.....

ΒΑΣΙΛΕΙΟΣ Π. ΚΟΤΡΩΝΗΣ

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

Περίληψη

Η εικονικοποίηση δικτύων αποτελεί μία δυναμική μέθοδο ενσωμάτωσης εικονικών δικτυακών δομών σε φυσικές υποδομές. Παρέχει τη δυνατότητα ταυτόχρονης και παράλληλης λειτουργίας πολλαπλών αρχιτεκτονικών, connection-oriented υπηρεσιών και ετερογενών δικτυακών πειραμάτων σε μία κοινή διαμοιραζόμενη φυσική πλατφόρμα διασύνδεσης.

Μία σημαντική πρόκληση που αντιμετωπίζει η μέθοδος είναι το πρόβλημα της ενσωμάτωσης των εικονικών γράφων στον κοινό φυσικό. Η αποδοτική απεικόνιση των εικονικών κόμβων και ζεύξεων πάνω στο υπόστρωμα, μεριμνώντας για την τήρηση των περιορισμών των στοιχείων, την καλύτερη δυνατή κατανομή των πόρων και την πλήρη αξιοποίηση της διαθέσιμης χωρητικότητας είναι ένα NP-δύσκολο πρόβλημα. Για το λόγο αυτό μεγάλο ποσοστό της έρευνας γύρω από αυτό το θέμα επικεντρώνεται στη σχεδίαση και χρήση ευριστικών αλγορίθμων με στόχο την εύρεση μίας αποδεκτής και οικονομικής λύσης.

Στην εργασία αυτή χρησιμοποιείται ως βάση και τροποποιείται ένας αλγόριθμος της διεθνούς βιβλιογραφίας ο οποίος προγραμματίζεται εξαρχής σε γλώσσα C. Στηρίζεται στην ευριστική ανίχνευση του ισομορφισμού μεταξύ υπογράφων και χαρτογραφεί τους κόμβους και τις ζεύξεις στην ίδια φάση επιτυγχάνοντας υψηλή τάξη συγχρονισμού της απεικόνισης, αποδοτικές χαρτογραφήσεις και πολύ μεγάλη ταχύτητα λειτουργίας. Προσομοιώνεται ουσιαστικά με βάση τον αλγόριθμο αυτό ένα γρήγορο σύστημα διαχείρισης και εξυπηρέτησης πολλαπλών, δυναμικά αφικνούμενων αιτήσεων εικονικών δικτύων, εξάγοντας συμπεράσματα για την αποδεκτότητα και αποδοτικότητα των χαρτογραφήσεων που παράγει σε πραγματικό χρόνο υπό συνθήκες ποικίλου φόρτου. Η προσομοίωση στηρίζεται στην χρονοδρομολόγηση γεγονότων.

Η εργασία αυτή είναι δομημένη ως εξής:

Αρχικά στο μέρος I γίνεται μία εισαγωγή στην τεχνική της εικονικοποίησης δικτύων, στα πλεονεκτήματά της, στα προβλήματα που αντιμετωπίζει, στην υλοποίησή της και στο βασικό θέμα της εργασίας. Ακολούθως στο μέρος II, γίνεται μία αναφορά σε ανάλογες εργασίες και δημοσιεύσεις που είναι σχετικές με το θέμα και στις μεθόδους και λύσεις οι οποίες προτάθηκαν σε αυτές. Κατόπιν στο μέρος III δίνεται μία μαθηματική περιγραφή του μοντέλου του δικτύου (εικονικού και φυσικού) και του ίδιου του προβλήματος ενσωμάτωσης. Στο μέρος IV παρουσιάζεται ο πλήρης αλγόριθμος που χρησιμοποιήθηκε για την επίλυση του προβλήματος χρησιμοποιώντας ευριστικές μεθόδους. Επίσης προτείνεται για εξέταση ένας θεωρητικός αλγόριθμος εύρεσης μονοπατιού για πολλαπλές ροές και δύο αλγόριθμοι για την θεωρητική ανάκαμψη από φυσικές αστοχίες κόμβων και ζεύξεων. Το μέρος V περιέχει τα στοιχεία που αφορούν το περιβάλλον προσομοίωσης του βασικού αλγορίθμου και συγκεντρωτικά τα αποτελέσματα της προσομοίωσης, ακολουθούμενα από σχόλια και συμπεράσματα σχετικά με αυτά. Τέλος στο μέρος VI γίνεται μία συνολική ανασκόπηση της συνεισφοράς της εργασίας στην έρευνα γύρω από το θέμα της ενσωμάτωσης εικονικών δικτύων και διατυπώνονται κάποιες προτάσεις για μελλοντική έρευνα στον γενικότερο τομέα της εικονικοποίησης δικτύων.

Επιπλέον, στο παράρτημα που βρίσκεται στο τέλος της εργασίας περιέχεται ο σχολιασμένος κώδικας σε C που χρησιμοποιήθηκε για τη διεξαγωγή της προσομοίωσης του αλγορίθμου.

Λέξεις κλειδιά

εικονικοποίηση, εικονικό δίκτυο, υπόστρωμα, ενσωμάτωση, χαρτογράφηση, κατανομή πόρων, ισομορφισμός, προσομοίωση

Abstract

Network virtualization is a dynamic method of embedding virtual networks in physical infrastructures. It provides the capability of simultaneous and parallel function of multiple architectures, connection-oriented services and heterogeneous network experiments on a common shared physical networking platform.

An important challenge which this method faces is the problem of embedding the virtual graphs in the common physical graph. The efficient mapping of virtual nodes and links on the substrate network, while taking into account the constraints of the components, the optimal resource allocation and the full exploitation of the available substrate capacity is an NP-hard problem. Therefore, a substantial percentage of research around this topic is focused on the design and use of heuristic algorithms so as to find an acceptable and efficient solution.

This dissertation uses as its basis an existing algorithm of the international bibliography, which is adjusted and reprogrammed from the beginning in C. The algorithm is based on the heuristic search of the isomorphism between subgraphs and maps the nodes and links during the same stage, yielding a high level of mapping synchronization, efficient mappings and a relatively high running speed. Essentially, a fast system which manages and serves multiple, dynamically arriving virtual network requests is simulated. Observations regarding its function lead to the deduction of numerous conclusions about the acceptability and efficiency of the mappings which are produced real-time and under diverse request loads. The simulation is based on event routing.

This thesis is organized as follows:

Firstly, part I contains an introduction to the technique of network virtualization, its advantages, the problems it faces, its implementation and the basic topic of the thesis. Sequentially, part II sums up similar research efforts and papers which are relative to the matter and presents some methods and solutions which they propose. In part III, a mathematical model of the networks (virtual and physical) and the formulation of the problem of virtual network embedding are presented. Part IV contains the algorithm which is used for the solution of the problem using heuristic techniques. Moreover three theoretical algorithms are presented: one regarding path-finding for multiple flows and two others which in theory face the problem of recovery from physical node and link failures. Part V describes the environment of the simulation of the basic algorithm and presents the experimental results extracted by the simulation, accompanied by useful comments and conclusions. Finally, part VI gives an overview of the contribution of this thesis to research around the topic of virtual network embedding and formulates some proposals of future research in the generic field of network virtualization.

Additionally, the appendix which is placed at the end of the thesis contains the commented C code which was manufactured and used for the needs of the simulation of the algorithm.

Key words

virtualization, virtual network, substrate, embedding, mapping, resource allocation, isomorphism, simulation

Ευχαριστίες

Ευχαριστώ θερμά την οικογένειά μου για τη στήριξη που μου παρείχε καθ' όλη τη διάρκεια συγγραφής της εργασίας, τον επιβλέποντα καθηγητή μου κύριο Βασίλειο Μάγκλαρη, καθηγητή Ε.Μ.Π. για την ανάθεση και επίβλεψη της διπλωματικής και τον κύριο Λεωνίδα Λυμπερόπουλο, διδάκτορα του Imperial College of London και συνεργάτη του εργαστηρίου NETMODE για την τεχνική βοήθεια που προσέφερε.

Βασίλειος Π. Κοτρώνης

Αθήνα, 9η Ιουνίου 2011

Πίνακας περιεχομένων

Πίνακας εικόνων

I. Εισαγωγή

I.1 *Το γενικό πρόβλημα του Internet*

Το Internet αποτελεί αναμφίβολα ένα ιδιαίτερα σημαντικό κομμάτι της καθημερινότητας του σύγχρονου ανθρώπου, ως ένας γρήγορος και εύκολα προσβάσιμος τρόπος επικοινωνίας, ανταλλαγής δεδομένων και παροχής ηλεκτρονικών υπηρεσιών. Οι εξαιρετικές δυνατότητες προσαρμογής και επέκτασης που το χαρακτηρίζουν οδήγησαν στην ευρεία χρήση του και στην θεώρησή του ως παγκόσμιου αγαθού. Όμως ακριβώς αυτό το χαρακτηριστικό της ευρείας χρήσης του και της συνεχούς, σχεδόν εκθετικής αύξησης των χρηστών που μετέχουν σε αυτό και επιθυμούν ικανοποιητική ποιότητα σε ετερογενείς υπηρεσίες απειλεί να οδηγήσει στον κορεσμό των πόρων του και στην αδυναμία αποδοτικής αξιοποίησης της διαθέσιμης χωρητικότητάς του. Επίσης, ως οντότητα το Internet δεν διευκολύνει την ταχεία και μεγάλη σε εύρος απορρόφηση νέων τεχνολογιών όπως για παράδειγμα τις διαφοροποιημένες υπηρεσίες και την πολυδιανομή πάνω από IP, καθώς οιεσδήποτε μεταβολές της υπάρχουσας αρχιτεκτονικής είναι εξαιρετικά δύσκολο να πραγματοποιηθούν. Αυτό οφείλεται στην ύπαρξη πολλαπλών μετόχων και εμπορικών παραγόντων με συγκρουόμενους στόχους και πολιτικές, δηλ. η εμπορική φύση του Internet είναι ένα από τα κυριότερα αίτια της ακαμψίας που το χαρακτηρίζει επί του παρόντος.

Για την αντιμετώπιση των προβλημάτων που αντιμετωπίζει το Internet, έχει γίνει μία αποσύνθεση των εμπλεκόμενων ζητημάτων σε μία προσπάθεια κατανεμημένης επίλυσης. Τα σημαντικότερα βήματα που έχουν γίνει ή βρίσκονται καθοδόν προς εφαρμογή είναι τα ακόλουθα:

- Διευθυνσιοδότηση➔ αρχικά IPV4 με class addressing, στη συνέχεια CIDR με τη συνέργεια της τεχνικής NAT και τελικά πρόταση IPV6 για παροχή τεράστιου αριθμού IP διευθύνσεων και πραγματικό end-to-end connectivity.

- Scalability➔ ιεραρχική θεώρηση δικτύων, διαχωρισμός σε domains και AS δρομολόγηση εντός-εκτός domains➔ rip, ospf, igp, bgp
  (➔συνεργασία διαχειριστών σε παγκόσμιο επίπεδο)

- ασφάλεια από άκρο σε άκρο➔ TLS, SSH, VPN, Tunneling κτλ.

- αποδοτική κατανομή και εξασφάλιση πόρων➔ ανοιχτό θέμα προς έρευνα: τεχνικές αποδοτικής κατανομής πόρων, γραμμικός προγραμματισμός κτλ.

- εικονικοποίηση δικτύων➔νέα τεχνολογία που συνδυάζει όλα τα ανωτέρω και οδηγεί σταδιακά στη μεταμόρφωση του σύγχρονου Internet

I.2 *Η εικονικοποίηση δικτύων ως πιθανή λύση*

Η εικονικοποίηση δικτύων υπόσχεται να ξεπεράσει τις αδυναμίες του σύγχρονου Internet και να λειτουργήσει ως μία νέα, ηγετική τεχνολογία που παρέχει δυνατότητες δημιουργίας πολλαπλών εικονικών αρχιτεκτονικών δικτύων (virtual networks) πάνω στο ίδιο φυσικό δίκτυο-υπόστρωμα (substrate).

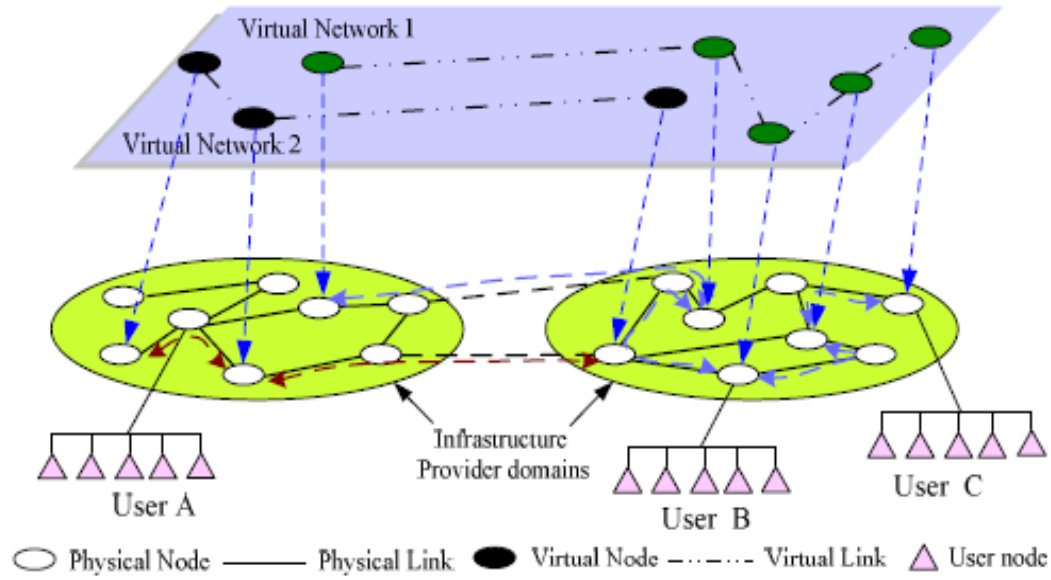
Σε ένα περιβάλλον εικονικοποίησης δικτύων, πολλαπλοί πάροχοι υπηρεσιών (service providers-SPs) έχουν την ικανότητα να δημιουργούν ετερογενή εικονικά δίκτυα (virtual networks-VNs) για να προσφέρουν πλήρως παραμετροποιήσιμες end-to-end υπηρεσίες. Για την επίτευξη αυτών των στόχων μισθώνουν διαμοιραζόμενους πόρους από έναν ή περισσότερους παρόχους υποδομής (infrastructure providers-InPs), χωρίς αξιοσημείωτες επενδύσεις σε φυσικές υποδομές.
Με αυτόν τον τρόπο μία φυσική υποδομή νοικιάζεται κατά τμήματα για να υποστηρίξει την παράλληλη λειτουργία πολλαπλών και ετερογενών εικονικών δικτυακών δομών που επικάθονται πάνω στη φυσική τοπολογία (overlay αρχιτεκτονική).

Ουσιαστικά, η εικονικοποίηση καθιστά εφικτή και διαχειρίσιμη την ενσωμάτωση πολλαπλών δικτύων διαφορετικών απαιτήσεων και τοπολογίας πάνω σε μία ενιαία φυσική πλατφόρμα διασύνδεσης. Συνεπώς εξελίσσει την κλασσική IP δικτύωση που βασίζεται στην μεταφορά πληροφορίας «βέλτιστης προσπάθειας», κατανέμοντας δίκαια και αποδοτικά το απαιτούμενο εύρος ζώνης και γενικά τους αναγκαίους πόρους για ποικίλα εικονικά δίκτυα. Παρέχει έτσι αξιόπιστες υπηρεσίες πολλαπλών συνδέσεων από άκρο σε άκρο (end-to-end). Αναβαθμίζει ένα φυσικό IP δίκτυο προσφέροντας στο μη αξιόπιστο διαδικτυακό «σύννεφο» τη δυνατότητα προσανατολισμού προς σύνδεση, με όλα τα πλεονεκτήματα που αυτό συνεπάγεται.

Πρακτικά λοιπόν, η εικονικοποίηση προσφέρει έναν τρόπο εγκαθίδρυσης και εξασφάλισης ποικίλων υπηρεσιών βασισμένων σε σύνδεση καθώς και διεξαγωγής ετερογενών διαδικτυακών πειραμάτων ταυτόχρονα σε μία κοινή υποδομή. Κάθε εικονικό δίκτυο που ενσωματώνεται για τη λειτουργία της υπηρεσίας ή του πειράματος έχει εγγυημένα τους πόρους που χρειάζεται κάθε στιγμή χωρίς να στηρίζεται στην απρόβλεπτη συμπεριφορά του «σύννεφου» του Internet. Η φυσική υποδομή είναι διαφανής για τον χρήστη, ο οποίος βλέπει μόνο το εικονικό δίκτυο και τους τρόπους επικοινωνίας που μπορεί να αξιοποιήσει με τη χρήση του δικτύου, χωρίς να ενδιαφέρεται για το πώς αυτό υλοποιείται στον φυσικό κόσμο.

Καθίσταται λοιπόν προφανές ότι παρέχεται η δυνατότητα εξέλιξης του σημερινού Internet με τη σταδιακή μετατροπή του σε έναν κόσμο πολλαπλών, ετερογενών, παράλληλων, αξιόπιστων δικτύων που εγκαθίστανται και καταργούνται σύμφωνα με τις ανάγκες των χρηστών και χαρακτηρίζονται από τις αρχές της ασφάλειας και της αποδοτικότητας.

Με τις δυνατότητες που παρέχει η παραπάνω μέθοδος καταπολεμώνται θεωρητικά και πρακτικά πολλά από τα προβλήματα που πηγάζουν από την ακαμψία και την απουσία ευελιξίας του Internet. Το παρακάτω σχήμα παρουσιάζει γραφικά τον τρόπο συνύπαρξης ετερογενών εικονικών δικτύων για την παράλληλη εξυπηρέτηση πολλαπλών χρηστών πάνω από το ίδιο domain-segmented υπόστρωμα:



Σχήμα 1: Παράλληλη ενσωμάτωση πολλαπλών εικονικών δικτύων πάνω σε ένα φυσικό υπόστρωμα (σχήμα από [2])

I.3 *Πιθανές μελλοντικές εφαρμογές εικονικοποίησης δικτύων*

Αφού η εικονικοποίηση επιτρέπει σε πολλές, ετερογενείς οντότητες να μοιράζονται με ασφάλεια ένα ενιαίο φυσικό δίκτυο, μπορεί ως μέθοδος να εφαρμοστεί αποτελεσματικά στις ακόλουθες περιπτώσεις (κυρίως για την εξυπηρέτηση εμπορικών οργανισμών), όπως αναφέρεται και στο [3]:

- Διάκριση δικτύων εταιρικών υπαλλήλων σε «μολυσμένα» και «καθαρά» όπου: «μολυσμένα» ➔ ο υπολογιστής του υπαλλήλου έχει περιορισμένη πρόσβαση επειδή πιθανόν είναι μολυσμένος με κάποιον επικίνδυνο ιό ή απλώς αντιμετωπίζει κάποιο πρόβλημα συμβατότητας και άρα πρέπει να συνδεθεί εδώ για αποκατάσταση.
«καθαρά» ➔ ο υπολογιστής του υπαλλήλου αφού έχει «εξυγιανθεί» με εφαρμογές αποκατάστασης του «βρώμικου» δικτύου και διαπιστωθεί ότι η κατάσταση υγείας του είναι αποδεκτή μπορεί να προσπελάζει απρόσκοπτα τις εφαρμογές όπου έχει δικαιοδοσία.

- Διάκριση εταιρικών δικτύων σε δίκτυα αναξιόπιστων επισκεπτών και δίκτυα αξιόπιστων συνεργατών, ώστε τα σημαντικά για τη λειτουργία της επιχείρησης πρόσωπα να έχουν πρόσβαση στις εφαρμογές που αφορούν την επιχείρηση ενώ οι υπόλοιποι να απολαμβάνουν περιορισμένα προνόμια. Αυτό επιτυγχάνεται με τον χωρισμό του φυσικού δικτύου σε κατάλληλα εικονικά διαμερίσματα.

- Απομόνωση ειδικών συσκευών και εφαρμογών : πολλές συσκευές οι οποίες παραδοσιακά συνδέονταν σε κλειστά δίκτυα όπως τα ATM και οι κάμερες παρακολούθησης CCTV, τώρα τείνουν να μεταφέρονται σε δίκτυα IP ώστε να είναι δυνατή η απομακρυσμένη πρόσβαση σε αυτές από εξουσιοδοτημένο προσωπικό. Με την εικονικοποίηση, τα διαμερίσματα όπου ανήκουν αυτές οι συσκευές μπορούν να προσπελάζονται με ασφάλεια από τους κατάλληλους ανθρώπους κατόπιν πιστοποίησης.

- Εξισορρόπηση φόρτου δικτύου τόσο μακροπρόθεσμα όσο και βραχυπρόθεσμα, αφού αλλάζει η έννοια του ελέγχου του δικτύου : αρκούν κατάλληλες ρυθμίσεις των παραμέτρων των εικονικών δικτύων βάσει των δυνατοτήτων του φυσικού υποστρώματος (ευκολότερο από την αλλαγή των παραμέτρων του ίδιου του φυσικού δικτύου) για ανταπόκριση σε ανεπιθύμητες μεταβολές του φορτίου.

- Δημιουργία εργασιακών τμημάτων μέσα σε μία επιχείρηση με ασφάλεια χωρίς να χάνεται η δυνατότητα συνεργασίας : κάθε τμήμα διαθέτει το δικό του προστατευμένο εικονικό διαμέρισμα και μπορεί εύκολα να ελεγχθεί η ροή δεδομένων τόσο εντός ενός τμήματος όσο και ανάμεσα σε διαφορετικά τμήματα (ασφάλεια και συνεργασία).

- Φιλοξενούμενα δίκτυα οντοτήτων μέσα σε άλλες οντότητες : δίνεται η δυνατότητα δημιουργίας ξεχωριστών εικονικών δικτύων για κάθε υπηρεσία (εμπορική ή άλλου είδους) πάνω στην ίδια υποδομή ("virtual over physical"), χωρίς να υπάρχει σύγχυση των υπηρεσιών και με εξασφάλιση της ιδιωτικής ανάθεσης πόρων και της ασφάλειας των ανταλλασσόμενων πληροφοριών.

- Οποιαδήποτε δικτυακή εφαρμογή ή υπηρεσία (εμπορική ή μη) απαιτεί την παράλληλη λειτουργία εικονικών δικτύων πάνω σε μία κοινή δικτυακή βάση, εξασφαλίζοντας αποδοτική ανάθεση των απαιτούμενων πόρων λειτουργίας στις επιμέρους εικονικές διασυνδεόμενες δομές.

I.4 *Σχεδιαστικές απαιτήσεις εικονικοποίησης δικτύων*

Οι παραπάνω εφαρμογές και δυνατότητες είναι πραγματικά εντυπωσιακές.
Όμως η εικονικοποίηση δικτύων απαιτεί σχεδιαστικά τα ακόλουθα βασικά στοιχεία ώστε να μπορεί να υλοποιήσει τα όσα υπόσχεται ως τεχνολογία:

- ***Ευελιξία-προσαρμοστικότητα***

Κάθε εικονικό δίκτυο πρέπει να έχει τη δυνατότητα ευέλικτης και προσαρμόσιμης απεικόνισης πάνω στο υπόστρωμα. Ο πάροχος υπηρεσίας δεν πρέπει να περιορίζεται ούτε ως προς την τοπολογία τους εικονικού δικτύου που θέλει να ενσωματωθεί, ούτε ως προς τις λειτουργίες δρομολόγησης, ελέγχου και διαχείρισης που επιθυμεί να χαρακτηρίζουν το δίκτυο αυτό. Επίσης πολύ σημαντικό είναι ο πάροχος της φυσικής υποδομής να μπορεί να αντιμετωπίσει γρήγορα τυχόν αστοχίες του υποστρώματος ώστε να μην διαταραχθεί η λειτουργία των ήδη απεικονισθέντων εικονικών δικτύων. Η ανάκαμψη από αστοχία προϋποθέτει την εύκολη και διαφανή για τον πάροχο και τον χρήστη αλλαγή της απεικόνισης του εικονικού δικτύου σε διαφορετικά λειτουργικά τμήματα του υποστρώματος μέχρι να επιδιορθωθεί η βλάβη.
Ανάλογες απαιτήσεις εγείρονται κατά την ενσωμάτωση πολλαπλών εικονικών δικτύων με στόχο την βέλτιστη αξιοποίηση της χωρητικότητας, καθώς κάθε καινούρια απεικόνιση δεν πρέπει να διαταράσσει τις ήδη υπάρχουσες, αλλά πρέπει να υπάρχει μέριμνα για την πραγματοποίηση διαφανών μεταβολών (διαχωρισμός μονοπατιού, μετανάστευση μονοπατιού) που δεν τοποθετούν εκτός λειτουργίας κάποιο δίκτυο.

- ***Διαχειρισιμότητα***

Είναι επιθυμητός ο end-to-end έλεγχος και διαχείριση του δικτύου χωρίς η μετάβαση ανάμεσα στα φυσικά όρια των διαχειριστικών περιοχών (AS domains) να επηρεάζει τη λειτουργία των εικονικών δικτύων. Για την επίτευξη αυτού του στόχου απαιτείται διαχωρισμός των λειτουργιών διαχείρισης των παρόχων φυσικής υποδομής και των παρόχων υπηρεσιών (πάνω από εικονικές δικτυακές δομές). Δηλ. είναι απαραίτητη η υποστήριξη αυτής της διαχειριστικής διαφοροποίησης ώστε ο πάροχος υπηρεσίας να δίνει την δυνατότητα στον τελικό χρήστη της υπηρεσίας του να αντιμετωπίζει διαφανώς την φυσική υποδομή που φιλοξενεί το εικονικό δίκτυο πρόσβασης.

- ***Scalability***

Η δυνατότητα κλιμάκωσης είναι πολύ σημαντική γιατί αποτελεί άμεση απαίτηση της ταυτόχρονης και παράλληλης λειτουργίας πολλαπλών και ετερογενών (π.χ. ως προς το μέγεθος) εικονικών δικτύων, όταν αυτά συνυπάρχουν πάνω από μία κοινή υποδομή. Κάθε πάροχος υποδομής οφείλει να μπορεί να κλιμακώνει την προσφορά πόρων (μέχρι κάποιο όριο συμφόρησης φυσικά) και να φροντίζει ώστε αυτοί να κατανέμονται κατάλληλα και αποδοτικά για την κάλυψη των αιτήσεων των παρόχων υπηρεσίας. Σημαντικό είναι η (ανα)κατανομή των πόρων που αποσκοπεί στη βέλτιστη χρησιμοποίηση να μην επιδρά δυσμενώς στην λειτουργία και την ποιότητα υπηρεσίας των υπαρχόντων εικονικών δικτύων.

- ***Ασφάλεια***

Φυσικά, σημαντική παράμετρος για την λειτουργία των εικονικών δικτύων είναι η κάλυψη των βασικών απαιτήσεων ασφάλειας (αυθεντικοποίηση, εμπιστευτικότητα, ακεραιότητα, μη αποποίηση ευθύνης), ώστε τα δεδομένα που διακινούνται εντός των δικτύων αυτών να είναι πραγματικά μη προσπελάσιμα από κάποιον κακόβουλο χρήστη. Επίσης πρέπει να υπάρχει σαφής απομόνωση μεταξύ των δικτύων αφού οι πόροι που κατανέμονται σε αυτά οφείλουν να είναι «ιδιωτικοί» ακόμα και αν βρίσκονται στις ίδιες φυσικές θέσεις. Δηλ. πρέπει να υπάρχει σαφής απομόνωση λογικών και φυσικών πόρων. Απαιτείται λοιπόν ανοχή σε σφάλματα και επιθέσεις για κάθε εικονικό δίκτυο. Σε περίπτωση που μία βλάβη (ηθελημένη ή μη) παρουσιαστεί σε κάποιο από αυτά πρέπει τα υπόλοιπα να μπορούν να συνεχίζουν την λειτουργία τους απρόσκοπτα. Για την εξασφάλιση των δικτύων μπορούν να χρησιμοποιηθούν είτε ήδη υπάρχοντα πρωτόκολλα ασφάλειας ανώτερων επιπέδων με κατάλληλες τροποποιήσεις, είτε να εισαχθεί η έννοια της ασφάλειας ως λειτουργίας εγγυημένης λόγω της απομόνωσης και του διαχωρισμού που λαμβάνει χώρα σε κατώτερα επίπεδα της δικτυακής στοίβας πρωτοκόλλων (π.χ. επίπεδο 2). Αυτό είναι ένα ανοιχτό θέμα προς έρευνα.

- ***Ετερογένεια***

Τα εικονικά δίκτυα διαφέρουν σε πολλαπλά χαρακτηριστικά: το μέγεθος, στις απαιτήσεις υπηρεσίας, τους αλγορίθμους δρομολόγησης, προώθησης, διαχείρισης και έλεγχου που τα διέπουν κτλ. Διαφοροποίηση παρατηρείται και στην υποκείμενη δικτυακή υποδομή: οι ζεύξεις μπορεί να υλοποιούνται με χάλκινα καλώδια, οπτικές ίνες ή ασύρματους διαύλους, οι φυσικοί δικτυακοί κόμβοι μπορεί να είναι κατασκευασμένοι από διαφορετικούς κατασκευαστές και να εφαρμόζουν εικονικοποίηση με διαφορετικούς τρόπους. Ένας πάροχος υπηρεσίας πρέπει να έχει τη δυνατότητα να σχηματίσει ένα εικονικό δίκτυο χωρίς να επηρεάζεται από την τομεοποίηση, τα πρωτόκολλα λειτουργίας και τα ζητήματα συμβατότητας που χαρακτηρίζουν το φυσικό δίκτυο. Γι' αυτό το λόγο, πρέπει το φυσικό υπόστρωμα να

έχει τη δυνατότητα γεφύρωσης όλων των δυνατών χασμάτων ετερογένειας που ίσως παρατηρηθούν και να υποστηρίζει πολλαπλές τεχνολογίες και πρωτόκολλα.

- ***Παραμετροποιησιμότητα μέσω προγραμματισμού***

Απαιτείται δυνατότητα λογικού προγραμματισμού των στοιχείων ενός εικονικού δικτύου ώστε να παρέχεται ευελιξία και δυνατότητα προσαρμογής σε μεταβαλλόμενες συνθήκες. Για παράδειγμα, πρέπει να υπάρχει δυνατότητα δυναμικής εναλλαγής μεταξύ διαφορετικών δικτυακών πρωτοκόλλων και επιλογής της ανά περίσταση κατάλληλης προγραμματιστικής λύσης για την υλοποίηση μίας υπηρεσίας. Αυτή είναι και απαραίτητη προϋπόθεση για την εφαρμογή ευέλικτων δικτυακών πειραμάτων με ενσωματωμένες δυνατότητες παραμετροποίησης.

- ***Κληροδότηση χαρακτηριστικών***

Η εικονικοποίηση προωθεί τη μεταβολή της φύσης του Internet, αλλά όχι φυσικά την καταστροφή του. Μία τέτοια τεχνολογία πρέπει να ενσωματωθεί ομαλά στο υπάρχον Διαδίκτυο, χωρίς να αλλαχθούν ή καταργηθούν υπάρχουσες τεχνολογίες, υπηρεσίες και εφαρμογές. Τα χαρακτηριστικά τους πρέπει να είναι κληροδοτήσιμα στα νέα εικονικά δίκτυα, δηλ. η νέα τεχνική δικτύωσης οφείλει να κληρονομήσει βασικά στοιχεία του Διαδικτύου. Αυτή η κληροδότηση είναι δυνατή εάν το Διαδίκτυο αντιμετωπιστεί ως ένα γιγάντιο εικονικό δίκτυο το οποίο συνυπάρχει με τις νέες εικονικές δικτυακές δομές.

➔Προφανώς οι παραπάνω απαιτήσεις είναι σε πολλές περιπτώσεις αλληλοσυγκρουόμενες και ο μηχανικός που σχεδιάζει και χρησιμοποιεί την τεχνολογία της εικονικοποίησης πρέπει να βρίσκει ανά περίπτωση τον τρόπο επίτευξης του επιθυμητού trade-off για την τήρηση ενός ικανοποιητικού QoS.

I.5 *Βάση εικονικοποίησης δικτύων*

Η εικονικοποίηση δικτύων στηρίζεται στην εκμετάλλευση των δυνατοτήτων των υπολογιστικών και δικτυακών συστημάτων, καθώς μέσα από την διαρκή αναβάθμιση του hardware (κυρίως) και του software (δευτερευόντος) έχει επιτευχθεί η δυνατότητα διαχωρισμού των φυσικών συσκευών σε πολλαπλές εικονικές συσκευές.

Σε επίπεδο κόμβων λοιπόν αναφέρονται ενδεικτικά οι ακόλουθες τεχνικές εικονικοποίησης και διαμοιρασμού των πόρων των κόμβων, όπως στο [4]:

- Πλήρης εικονικοποίηση όπου κάθε κόμβος τρέχει το δικό του στιγμιότυπο λειτουργικού συστήματος (VMWare Server, kernel-based VM)
- Εικονικοποίηση φιλοξενούμενου λειτουργικού συστήματος όπου ορισμένοι από τους πόρους του λειτουργικού απομονώνονται για τη χρήση τους από ένα εικονικό τερματικό-διεργασία (Linux VServers, FreeBSD Jails, Solaris Zones, OpenVZ)
- Παραεικονικοποίηση, που απαιτεί τροποποίηση του φιλοξενούμενου λειτουργικού και χρησιμοποιεί APIs (XEN, Denali)
- Εικονικοποίηση υλικού

Εκτός των άλλων αξιοποιούνται οι τεχνολογικές εξελίξεις στον τομέα των πολλαπλών πυρήνων, των μνημών, των πολυκάναλων διαύλων (buses), του παράλληλου προγραμματισμού και γενικότερα στον τομέα της αρχιτεκτονικής υπολογιστών.

Σε επίπεδο ζεύξεων μεταξύ των κόμβων των δικτύων υπάρχει η δυνατότητα υλοποίησης εικονικών ζεύξεων είτε με εικονικές σήραγγες (virtual tunnels) είτε με Ethernet over GRE over IP tunnels. Επίσης όσον αφορά τη χωρητικότητα ζεύξης γίνεται εκμετάλλευση των συνεχώς εξελισσόμενων μέσων και τεχνικών μετάδοσης που μπορούν να εξασφαλίσουν ευρυζωνική μεταφορά δεδομένων σε πολύ υψηλές ταχύτητες και με σημαντική αξιοπιστία.

Πρακτικά λοιπόν μπορεί να πραγματοποιηθεί πλέον μία διαδικασία συνδυασμού πόρων υλικού-λογισμικού και δικτυακής λειτουργικότητας σε μία ενιαία και διαχειρίσιμη μέσω λογισμικού οντότητα: το εικονικό δίκτυο. Η εικονικοποίηση δικτύων εμπλέκει εικονικοποίηση πλατφόρμας, συχνά συνδυαζόμενη με εικονικοποίηση πόρων.
Η διαδικασία κατηγοριοποιείται με τους ακόλουθους τρόπους ([5]):

- εξωτερική, συνδυάζοντας πολλαπλά δίκτυα ή τμήματα δικτύων σε μία εικονική μονάδα
- εσωτερική, παρέχοντας λειτουργικότητα που προσομοιώνει την δικτυακή στους υποδοχείς λογισμικού ενός ενιαίου συστήματος

Η εργασία ασχολείται μόνο με την πρώτη περίπτωση (εξωτερική εικονικοποίηση) δηλ. την δημιουργία εικονικών δικτυακών οντοτήτων πάνω από πολλαπλά domains και την αποδοτική ανάθεση πόρων σε αυτές.

Επομένως, σύμφωνα με τις σημερινές δυνατότητες συνδεσιμότητας και διαχείρισης των υπολογιστικών συστημάτων , μπορούν να εφαρμοστούν λογικές παραλληλίας και κατανομής λειτουργιών, ξεφεύγοντας από πρακτικές συγκεντρωτικών αρχιτεκτονικών.

➔Συνέπεια είναι η δυνατότητα δημιουργίας εικονικών δικτύων, τα οποία ως δομές περιγράφονται παρακάτω.

I.6 *Το εικονικό δίκτυο*

I.6.1 *Περιγραφή εικονικού δικτύου*

Ένα εικονικό δίκτυο γενικά αποτελείται από ένα σύνολο εικονικών κόμβων και εικονικών ζεύξεων που συνδέουν τους κόμβους μεταξύ τους. Οι εικονικοί κόμβοι μπορεί να είναι εικονικά τερματικά, εικονικοί δρομολογητές κτλ.. Οι κόμβοι αυτοί φιλοξενούνται από διαφορετικούς φυσικούς κόμβους ανάλογου τύπου, οι οποίοι είναι διασυνδεδεμένοι μέσω φυσικών μονοπατιών του υποκείμενου φυσικού δικτύου. Τα μονοπάτια αυτά αντιστοιχούν σε εικονικές ζεύξεις.

Η λογική αντιστοίχισης φυσικών-εικονικών δικτυακών στοιχείων για την υλοποίηση του εικονικού δικτύου διέπεται από τις ακόλουθες αρχές:

- Κάθε εικονικός κόμβος αντιστοιχίζεται μόνο σε ένα φυσικό
- Ένας φυσικός κόμβους μπορεί να φιλοξενεί πολλαπλούς εικονικούς κόμβους
- Κάθε εικονική ζεύξη αντιστοιχίζεται είτε σε ένα φυσικό μονοπάτι διαδοχικών φυσικών ζεύξεων είτε σε ένα σύνολο ροών που κατανέμονται σε πολλαπλά φυσικά μονοπάτια
- Μία φυσική ζεύξη μπορεί να φιλοξενεί πολλαπλές εικονικές ζεύξεις αποτελώντας τμήμα των ανάλογων φυσικών απεικονίσεων.

Σε κάθε περίπτωση τηρούνται οι περιορισμοί που διέπουν τα στοιχεία του υποκείμενου φυσικού δικτύου και ικανοποιούνται οι απαιτήσεις των εικονικών δικτυακών στοιχείων.

I.6.2 *Τύποι εικονικών δικτύων*

Τα εικονικά δίκτυα συναντώνται συνήθως στις ακόλουθες μορφές ([5]):

- Εικονικά τοπικά δίκτυα (Virtual LANs-VLANs), βασιζόμενα σε φυσικά LANs. Ένα VLAN μπορεί να δημιουργηθεί μέσω της διαμέρισης ενός φυσικού LAN σε πολλαπλά λογικά LANs (υποδίκτυα) με τη χρήση ενός VLAN ID. Εναλλακτικά, πολλαπλά φυσικά LANs μπορούν να λειτουργούν σαν ένα ενιαίο λογικό LAN. Το διαμερισμένο δίκτυο μπορεί να χρησιμοποιεί έναν μοναδικό router, ή πολλαπλά VLANs μπορούν να χρησιμοποιούν πολλαπλούς routers ακριβώς όπως πολλαπλά φυσικά LANs μπορούν. Ένα VLAN μπορεί να είναι ενσωματωμένο και υλοποιημένο μέσα σε ένα VPN.

- Εικονικά ιδιωτικά δίκτυα (Virtual Private Networks-VPNs), που αποτελούνται από πολλαπλά απομακρυσμένα τερματικά σημεία, συνήθως δρομολογητές που λειτουργούν ως πύλες VPN πελατών λογισμικού. Αυτά τα σημεία συνδέονται λογικά με κάποιου είδους σήραγγα πάνω από κάποιο άλλο δίκτυο, χρησιμοποιώντας φυσικά κατάλληλα πρωτόκολλα προώθησης της κίνησης και διασφάλισης της ανταλλασσόμενης πληροφορίας. Δύο τέτοια σημεία αποτελούν ένα ‘Από άκρο σ’ άκρο VPN’ (‘Point to Point VPN-PTP VPN). Σύνδεση περισσότερων των δύο σημείων τοποθετώντας ένα πλέγμα από εικονικές σήραγγες δημιουργεί ένα ‘VPN πολλαπλών σημείων’ (‘Multipoint VPN’)

- Εικονική ιδιωτική LAN υπηρεσία (Virtual Private LAN Service-VPLS), που είναι μία εξειδίκευση ενός VPN πολλαπλών σημείων. Το VPLS διαιρείται σε Διαφανείς Υπηρεσίες LAN (Transparent LAN Services-TLS) και Υπηρεσίες Εικονικής Σύνδεσης Ethernet (Ethernet Virtual Connection Services-EVLS). Μία TLS στέλνει ό,τι λαμβάνει, άρα παρέχει γεωγραφική απομόνωση αλλά όχι υποδικτύωση VLAN. Μία EVCS προσθέτει στα πακέτα VLAN IDs, οπότε παρέχει και γεωγραφική απομόνωση και δυνατότητα υποδικτύωσης VLAN.

Ένα κοινό παράδειγμα ενός εικονικού δικτύου που είναι βασισμένο σε εικονικές συσκευές είναι το δίκτυο εντός ενός VMWare ESX εικονικού host που υλοποιείται με τη χρήση εικονικών μεταγωγέων (virtual switches). Ανάλογα δίκτυα μπορούν να χρησιμοποιούν τόσο το βασικό πρωτόκολλο Ethernet όσο και πρωτόκολλα εικονικοποίησης όπως το IEEE 802.1Q για VLAN.

Ι.7 *Βασικό πρόβλημα ενσωμάτωσης εικονικών δικτύων*

Η εικονικοποίηση εμπλέκει πολλαπλά προβλήματα τα οποία πρέπει να επιλυθούν για την ομαλή εισαγωγή της στο σύγχρονο Internet. Ένα θεμελιώδες πρόβλημα που ανακύπτει στην διαδικασία της ενσωμάτωσης εικονικών δικτύων πάνω σε μία φυσική υποδομή είναι η βέλτιστη ανάθεση και διαμοιρασμός των πόρων που παρέχονται από ένα δεδομένο φυσικό IP δίκτυο. Οι υποκείμενοι πόροι πρέπει να χρησιμοποιούνται με σύνεση. Συνεπώς, απαιτούνται προηγμένες τεχνικές απεικόνισης των εικονικών δικτύων πάνω στο υπόστρωμα ώστε να αποφεύγονται φαινόμενα κατασπατάλησης πόρων και απόρριψης αιτήσεων εγκατάστασης νέων εικονικών δικτύων λόγω κακής εκμετάλλευσης της διαθέσιμης φυσικής χωρητικότητας.

Η απεικόνιση-ενσωμάτωση (embedding) αυτή είναι πρακτικά η ανάθεση κάθε εικονικού κόμβου σε συγκεκριμένο φυσικό κόμβο και αντίστοιχα κάθε εικονικής ζεύξης σε ένα ή περισσότερα μονοπάτια/ροές του φυσικού δικτύου, έτσι ώστε η λογική τοπολογία του εικονικού δικτύου να «επικάθεται» πάνω στην φυσική τοπολογία λαμβάνοντας τους πόρους που χρειάζεται για την λειτουργία του.

Ως τύποι φυσικών κόμβων μπορούν να θεωρηθούν, χωρίς βλάβη της γενικότητας, τερματικά και δρομολογητές που μπορούν να φιλοξενήσουν ταυτόχρονα πολλούς εικονικούς κόμβους αντίστοιχων όμως τύπων. Ένας εικονικός κόμβος καταλαμβάνει μία «φέτα»(slice) του αντίστοιχου φυσικού κόμβου όπου έχει απεικονιστεί και τους αντίστοιχους φυσικούς πόρους που απαιτεί. Αντίστοιχα μία εικονική ζεύξη καταλαμβάνει slices πολλαπλών φυσικών ζεύξεων. Πάνω σε μία φυσική ζεύξη μπορούν να συνυπάρχουν πολλές εικονικές ζεύξεις οι οποίες μπορούν να εκτείνονται ακόμα και σε ολόκληρα φυσικά μονοπάτια. Αν ειδωθεί ως συνολική οντότητα, το slice ορίζει την ανάθεση των κόμβων και ζεύξεων ενός εικονικού δικτύου και αποτελεί πρακτικά το στιγμιότυπο του εικονικού στο φυσικό δίκτυο.

Είναι προφανές ότι η χωρητικότητα, ως ένα συγκεκριμένο όριο λειτουργίας των φυσικών κόμβων και ζεύξεων (που έχει πολλές μορφές: υπολογιστική ισχύς, εύρος ζώνης, μέγιστος αριθμός εικονικών κόμβων ανά φυσικό κτλ.) είναι περιορισμένη και άρα η χαρτογράφηση-ανάθεση-ενσωμάτωση των εικονικών δικτύων υπόκειται σε αυστηρούς κανόνες.

Η πρόκληση είναι, τηρουμένων αυτών των κανόνων που είναι δεδομένοι και διέπουν το δοθέν φυσικό δίκτυο, να γίνει η ανάθεση έτσι ώστε οι φυσικοί πόροι να αξιοποιούνται με βέλτιστο τρόπο και να επιτρέπουν σε όσο το δυνατόν μεγαλύτερο αριθμό αιτήσεων εικονικών δικτύων να γίνονται δεκτές και να υλοποιούνται παράλληλα από την υποκείμενη υποδομή.
Αυτό είναι και το πρόβλημα το οποίο καλείται να λύσει αυτή η εργασία, έχοντας ως βάση αντίστοιχες εργασίες που έχουν δημοσιευθεί στο παρελθόν ([1], [6], [7], [8]).

Απαιτείται λοιπόν αλγόριθμος ο οποίος απεικονίζει τους εικονικούς κόμβους και τις εικονικές ακμές στα αντίστοιχα πραγματικά στοιχεία του φυσικού δικτύου, με τέτοιο τρόπο ώστε το ίδιο φυσικό δίκτυο να έχει τη δυνατότητα να φιλοξενήσει όσο το δυνατόν περισσότερα εικονικά δίκτυα που να χρησιμοποιούν όσο το δυνατόν καλύτερα τους πόρους του.

➔Όμως το παραπάνω πρόβλημα ενέχει πολλούς δύσκολα επιλύσιμους περιορισμούς που παρουσιάζονται στην ακόλουθη ενότητα.

I.8 *Γενικοί περιορισμοί ενσωμάτωσης εικονικών δικτύων*

Παρά τις προσπάθειες που έχουν γίνει για την επίλυση του προβλήματος, οι παρακάτω τέσσερις παράγοντες καθιστούν το πρόβλημα της VN ενσωμάτωσης εξαιρετικά δυσχερές ([8]).

I.8.1 *Περιορισμοί κόμβων και ακμών*

Κάθε αίτηση εγκαθίδρυσης VN εμπεριέχει περιορισμούς πόρων, όπως : απαίτηση CPU και μνήμης στους κόμβους και μέγιστος αριθμός εικονικών κόμβων που μπορούν να «τρέχουν» ταυτόχρονα πάνω στον ίδιο φυσικό, απαίτηση εύρους ζώνης ανά ζεύξη, μέγιστος αριθμός φυσικών ζεύξεων/εικονική ζεύξη (σε όρους καθυστέρησης λόγω αλμάτων-hops στο μονοπάτι) και πιθανότητα λάθους στην μετάδοση bits όσον αφορά τις ακμές-ζεύξεις.

Το καθαρά υπολογιστικό πρόβλημα το οποίο μας περιορίζει είναι το ότι η ενσωμάτωση VN σε φυσικά δίκτυα, με στόχο την βέλτιστη αξιοποίηση πόρων και με περιορισμούς-κανόνες ως προς τους κόμβους και τις ακμές, οδηγεί αποσυντιθέμενη στο NP-δύσκολο multi way separator πρόβλημα ([9]), ακόμα και αν όλες οι αιτήσεις είναι γνωστές εκ των προτέρων. Επίσης ακόμα και αν όλοι οι εικονικοί κόμβοι έχουν απεικονιστεί προκαταβολικά, η διαδικασία απεικόνισης εικονικών ακμών σε φυσικά μονοπάτια με περιορισμούς ως προς το εύρος ζώνης είναι ακόμα NP-δύσκολη στο σενάριο της αδιαχώριστης ροής.

Έχει αποδειχθεί λοιπόν ότι το γενικό πρόβλημα ενσωμάτωσης είναι υπολογιστικά μη επιλύσιμο σε πολυωνυμικό χρόνο αφού ανήκει στην κλάση των NP-δύσκολων προβλημάτων. Για την εύρεση «καλών» λύσεων (κοντά στην βέλτιστη) έχουν προταθεί κλασσικοί ευριστικοί αλγόριθμοι που είτε εκμεταλλεύονται την ισχύ των σημερινών υπολογιστικών συστημάτων (επιθέσεις ωμής δύναμης, εξομοιούμενη ανόπτηση κτλ. [9]) είτε στηρίζονται σε περισσότερο εκλεπτυσμένες μεθόδους αντιμετώπισης όπως οι αλγόριθμοι MCF σε συνδυασμό με αρχές του γραμμικού προγραμματισμού διαιρώντας την ανάθεση σε δύο φάσεις: ανάθεση κόμβων και κατόπιν ανάθεση ακμών ([7]). Ορισμένες στοχεύουν σε συντονισμό των δύο φάσεων για μεγαλύτερη αποτελεσματικότητα ([6]), όμως καταφεύγουν στη δημιουργία επαυξημένων φυσικών γράφων για την επίλυση με αποτέλεσμα τη μεγαλύτερη χρονική επιβάρυνση που εισάγεται κατά το τρέξιμο των αλγορίθμων.

I.8.2 *Έλεγχος αποδεκτότητας*

Οι πόροι του φυσικού δικτύου είναι πεπερασμένοι, άρα αναγκαστικά κάποιες αιτήσεις ενσωμάτωσης VN θα απορριφθούν ή θα αναβληθούν, ώστε να μην παραβιάσουν τις ελάχιστες απαιτήσεις των υπαρχόντων VNs. Αυτό είναι βασική παράμετρος που πρέπει να ληφθεί υπόψη, καθώς η θεμελιώδης αρχή της εικονικοποίησης είναι ότι η εισαγωγή νέων VN στην κοινή φυσική τοπολογία δεν διαταράσσει τη λειτουργία των ήδη απεικονισθέντων VN. Αποδεκτές πρέπει να γίνονται μόνο οι αιτήσεις που μπορούν να υποστηριχθούν από το υποκείμενο φυσικό δίκτυο.

Από τη στιγμή όμως που κάποιο VN εγκαθιδρύεται-ενσωματώνεται, το φυσικό δίκτυο του παραχωρεί τους αντίστοιχους πόρους (κόμβων και ακμών) οι οποίοι μπορούν να θεωρηθούν πλέον ιδιωτικοί όσον αφορά την πλευρά του κόσμου «έξω» από το εικονικό δίκτυο. Το πώς οι διάφοροι φυσικοί πόροι θα διαμοιραστούν δίκαια και αποτελεσματικά ανάμεσα στα VNs, χωρίς η λειτουργία του ενός να επιδρά δυσμενώς στη λειτουργία του άλλου είναι ζήτημα διαχείρισης του φυσικού δικτύου (τεχνικές χρονοπρογραμματισμού κτλ.).

I.8.3 *Αιτήσεις online*

Οι VN αιτήσεις δεν είναι γνωστές εκ των προτέρων. Μπορούν να φθάνουν δυναμικά, και εφόσον γίνουν αποδεκτές, να παραμένουν για άγνωστο χρονικό διάστημα στο δίκτυο μέχρι την αναχώρησή τους. Αυτή η ιδιότητα των VN είναι απόρροια της αντιμετώπισης «Το Διαδίκτυο ως υπηρεσία» (“Internet as a service”), δηλ. τα VNs αιτούνται πόρους για την λειτουργία δικτυακών υπηρεσιών οι οποίες μπορούν να δημιουργηθούν κατ’ απαίτηση και να διαρκέσουν μη προβλέψιμο χρονικό διάστημα. Το πρόβλημα λοιπόν γίνεται πολύ δύσκολο καθώς η ορατότητα του αλγορίθμου προς τα εμπρός είναι πολύ μικρή λόγω της απρόβλεπτης εξέλιξης των αιτήσεων.

I.8.4 *Ποικίλες τοπολογίες*

Οι τοπολογίες των VN δικτύων μπορούν να έχουν οποιαδήποτε μορφή, πέραν των κλασικών hub and spoke (διαμοιραστής-τερματικά) και tree (δεντρική δομή, π.χ. για multicasting). Ο αλγόριθμος πρέπει να μπορεί να διαχειρίζεται αιτήσεις κάθε δυνατής τοπολογίας όσο πιο αποδοτικά γίνεται, και να είναι ιδιαίτερα αποτελεσματικός στην διαχείριση συγκεκριμένων κοινών τοπολογιών όπως αυτές που προαναφέρθηκαν.

I.9 *Συνολική πρόκληση ενσωμάτωσης εικονικών δικτύων*

Βάσει των παραπάνω διαπιστώσεων προκύπτει συμπερασματικά η συνολική πρόκληση που συνεπάγεται η παγκόσμια εφαρμογή της τεχνολογίας της εικονικοποίησης δικτύων:

➔Η δημιουργία ενός συγκεντρωτικού ή κατανεμημένου, ιεραρχικού συστήματος VN ενσωμάτωσης το οποίο επιτελεί τις εξής λειτουργίες:

- Online υποδοχή και εξυπηρέτηση δυναμικά αφικνούμενων αιτήσεων ενσωμάτωσης ετερογενών εικονικών δικτύων, διαφορετικών τοπολογιών και ποικίλων απαιτήσεων ποιότητας υπηρεσίας από διαφορετικούς παρόχους υπηρεσίας.
- Αποτελεσματική διαχείριση των αιτήσεων σύμφωνα με συγκεκριμένες πελατειακές πολιτικές ελέγχου αποδεκτότητας και προτεραιοτήτων.
- Ενσωμάτωση όσο το δυνατόν περισσότερων εικονικών δικτύων με όσο το δυνατόν αποδοτικότερη χρησιμοποίηση της διαθέσιμης χωρητικότητας του υποστρώματος (αποδοτική χρήση ολόκληρου του υποστρώματος χωρίς τον περιορισμό των domains).
- Όλα τα παραπάνω πρέπει να γίνονται λαμβάνοντας υπόψη τη μεγιστοποίηση τους κέρδους ανταγωνιστικών πλευρών: των παρόχων υπηρεσίας και των παρόχων υποδομής.

➔ Μία τέτοια προσέγγιση συστήματος προσπαθεί να επιτύχει η τρέχουσα εργασία της οποίας το βασικό θέμα παρουσιάζεται εκτενώς παρακάτω.

I.10 *Το βασικό θέμα της εργασίας*

Στην τρέχουσα εργασία παρουσιάζεται και υλοποιείται με κάποιες τροποποιήσεις ένας ήδη υπάρχων αλγόριθμος ο οποίος συνδυάζει ευριστικές μεθόδους με αλγορίθμους MCF, με συγχώνευση των φάσεων ανάθεσης κόμβων και ακμών για καλύτερα αποτελέσματα και με χρήση τεχνικών βελτιστοποίησης ώστε η έξοδος του αλγορίθμου να είναι κοντά στη βέλτιστη λύση.

Λαμβάνονται υπόψη όλοι οι περιορισμοί που χαρακτηρίζουν την φυσική υποδομή και λαμβάνεται μέριμνα για την περίπτωση όπου ο αλγόριθμος αδυνατεί να ενσωματώσει το εικονικό δίκτυο και άρα αποκρίνεται αρνητικά στην αίτηση (η οποία τότε εισέρχεται σε ουρά αναμονής). Οι αιτήσεις μπορούν να καταφτάνουν δυναμικά, και για την προσομοίωση υιοθετείται το πιθανοτικό μοντέλο αφίξεων Poisson. Η λογική της προσομοίωσης είναι οδηγούμενη από τα γεγονότα (event-driven).

Η τοπολογία που έχει υποτεθεί είναι αυθαίρετη όσον αφορά τη διασύνδεση μεταξύ των routers και αστεριού όσον αφορά τη διασύνδεση τερματικών-routers δηλ. ανταποκρίνεται πλήρως στη εικόνα ενός σύγχρονου φυσικού IP δικτύου. Φυσικά η τοπολογία αυτή δεν είναι δεσμευτική αλλά χρησιμοποιήθηκε για την κάλυψη των απαιτήσεων της προσομοίωσης.

Φυσικά δεν αμελήθηκε η ύπαρξη switches αλλά πολύ απλά το πρόβλημα μπορεί κάλλιστα να λυθεί με τον δοθέν αλγόριθμο αν αντιμετωπίσουμε τα routers ως switches, αφού δεν αλλάζει κάτι ουσιαστικό στην απεικόνιση παρά μόνον ίσως οι απαιτήσεις των κόμβων (➔επιλύεται με κατάλληλη παραμετροποίηση).

Γενικά αυτό είναι και ένα παράπλευρο αποτέλεσμα της εργασίας αυτής: θα παραχθεί κώδικας ο οποίος μπορεί εύκολα να παραμετροποιηθεί αν απαιτηθεί η προσθήκη επιπλέον τύπων κόμβων, η τροποποίηση ή εισαγωγή μετρικών πόρων του δικτύου κτλ. Επειδή δηλ. είναι πολύ γενικός και γράφτηκε σε γλώσσα C για υψηλή ταχύτητα προσομοίωσης και για δυνατότητα παραμετροποίησης ακόμα και σε χαμηλό επίπεδο προσφέρει μεγάλη ελευθερία αλλαγών χωρίς να απαιτηθούν δραματικές αλλαγές στη φιλοσοφία του.

Τελικός στόχος είναι ο αλγόριθμος να μπορεί να δουλέψει επιτυχώς σε συνθήκες πραγματικού χρόνου με αποτελεσματικότητα και αποδοτικότητα. Στη λύση έχουν ληφθεί υπόψη και κάποια στοιχεία οικονομοτεχνικής ανάλυσης, καθώς είναι σαφές ότι οι αιτήσεις υλοποίησης εικονικών δικτύων γίνονται από παρόχους υπηρεσιών που απευθύνονται σε έναν πάροχο φυσικής δικτυακής υποδομής. Κάθε πλευρά θέλει να ελαχιστοποιήσει το κόστος που την αφορά και να μεγιστοποιήσει τις απολαβές της.

Συγκεκριμένα γίνεται αναφορά στην πλευρά του παρόχου υπηρεσίας, που επιδιώκει μέγιστο κέρδος με το ελάχιστο δυνατό κόστος και από τον οποίον πηγάζει η υπηρεσία ενσωμάτωσης. Ο πάροχος υπηρεσίας νοικιάζει πόρους του φυσικού δικτύου από τον πάροχο υποδομής και απολαμβάνει αντίστοιχα κέρδη από την χρήση της εκάστοτε υπηρεσίας από τους πελάτες. Τα κόστη του αφορούν την κράτηση των φυσικών πόρων ώστε να εξυπηρετηθούν οι εικονικές απαιτήσεις.

ΙΙ. Παρελθούσα έρευνα σχετικά με το πρόβλημα της ενσωμάτωσης

Το πρόβλημα της ενσωμάτωσης εικονικών δικτύων αποτελεί γενίκευση των ήδη ερευνηθέντων προβλημάτων της ενσωμάτωσης Εικονικών Ιδιωτικών Δικτύων (VPNs) σε μία διαμοιραζόμενη τοπολογία παρόχου καθώς και της χαρτογράφησης μιας εικονικής πειραματικής δικτυακής δομής (testbed), [10],[11].

Όμως, ένα VPN χαρακτηρίζεται από περιορισμούς που αφορούν μόνο τις απαιτήσεις εύρους ζώνης στις ζεύξεις, χωρίς περιορισμούς στους πόρους των κόμβων. Έτσι το πρόβλημα της ενσωμάτωσης VPN και τους σχεδιασμού των αναγκαίων αλγορίθμων ανάγεται στην εύρεση κατάλληλων μονοπατιών για τα ζεύγη πηγή/προορισμός, χωρίς να εξετάζονται τα τερματικά άκρα ξεχωριστά.

Από την άλλη πλευρά, αλγόριθμοι όπως ο assign [11] που χρησιμοποιείται στο Emulab testbed [12] λαμβάνουν υπόψη τόσο περιορισμούς ζεύξεων όσο και περιορισμούς κόμβων, αλλά επιτρέπουν μόνο αποκλειστική χρήση κόμβων: διαφορετικοί εικονικοί κόμβοι απαγορεύεται να μοιράζονται έναν κοινό φυσικό κόμβο.

Είναι όμως προφανές ότι στην εικονικοποίηση δικτύων εμπλέκονται πολλαπλοί περιορισμοί χωρητικότητας και τοποθέτησης τόσο στους εικονικούς κόμβους όσο και στις εικονικές ζεύξεις, ενώ φυσικοί κόμβοι και ακμές μπορούν να φιλοξενούν συνολικά πολλά και ετερογενή εικονικά δίκτυα. Η ισχυρή αυτή απαίτηση συντελεί στην κατάταξη του προβλήματος στα NP-hard.

Οι ερευνητές λοιπόν άρχισαν να στοχεύουν στην μείωση της πολυπλοκότητας του προβλήματος και στην εύρεση έξυπνων ευριστικών λύσεων.
Για το λόγο αυτό περιόρισαν το πρόβλημα σε διαφορετικές διαστάσεις που διευκολύνουν την επίλυσή του με τους ακόλουθους τρόπους (όλες οι θεωρήσεις είναι στατικές και όχι δυναμικές):

- Θεώρηση της offline έκδοσης του προβλήματος, χωρίς ανάγκη αντιμετώπισης δυναμικά αφικνούμενων αιτήσεων ενσωμάτωσης και real-time κατανομής και ανάθεσης πόρων ([13], [14]).

- Μη θεώρηση είτε των απαιτήσεων κόμβων είτε των απαιτήσεων ζεύξεων για απλοποίηση των περιορισμών ([14], [15]).

- Υπόθεση άπειρης χωρητικότητας κόμβων και ζεύξεων έτσι ώστε να εξαλειφτεί η ανάγκη ελέγχου αποδεκτότητας εικονικών αιτήσεων ([13], [14], [15]).

- Επικέντρωση σε συγκεκριμένες τοπολογίες εικονικών δικτύων που είναι αρκετά συνήθεις και βοηθούν την αποδοτική επίλυση όπως τοπολογία αστέρα (διαμοιραστής-τερματικά) και δέντρου [14]. Οι τοπολογίες αυτές προκύπτουν με εξειδικευμένες τεχνικές αποσύνθεσης της αρχικής αυθαίρετης τοπολογίας.

- Υπόθεση υποστήριξης τεχνικών μετανάστευσης κόμβων και ζεύξεων και διαχωρισμού μονοπατιών από το υπόστρωμα [17], σε συνδυασμό με τον αλγόριθμο κ-συντομότερων μονοπατιών για την απεικόνιση των ζεύξεων. Η ύπαρξη αυτών των δυνατοτήτων αποδεικνύεται ότι οδηγεί σε αποδοτικότερες χαρτογραφήσεις.

- Εφαρμογή τεχνικών και αλγορίθμων που έχουν επιτυχημένα υλοποιηθεί σε πραγματικά δίκτυα μεταφοράς κίνησης (π.χ. ηλεκτρικά), θεωρώντας κόμβους πρόσβασης (access nodes) και κόμβους «ραχοκοκαλιάς» (backbone nodes), [13]. Η κίνηση κατανέμεται σε ζεύγη πηγής-προορισμού τα οποία αντιμετωπίζονται ως ανεξάρτητες οντότητες (commodities). Για την επίλυση του προβλήματος εφαρμόζονται λογικές εμπνευσμένες από τα δίκτυα μεταφοράς, όπως οι αλγόριθμοι MCF (Multi-Commodity Flow) ([16], [13], [6], [7]) γραμμικού προγραμματισμού ή τετραγωνικού προγραμματισμού μεικτού ακεραίου. Ειδικά σε αυτόν τον τομέα (MCF) έχουν προταθεί πολλοί τρόποι επίλυσης του μαθηματικού προβλήματος όπως για παράδειγμα η χρήση στρατηγικής ενεργού συνόλου [17].

Ειδική αναφορά στο πρόβλημα του virtual testbed γίνεται στο [9] όπου περιέχονται πολλές από τις θεωρητικές προσεγγίσεις στην ανάθεση εικονικών κόμβων σε φυσικούς κόμβους, με σημαντικότερες τις ακόλουθες:

- Επιθέσεις ωμής δύναμης με χρήση έξυπνων ευριστικών για δυνατότητα υπαναχώρησης (backtracking) ή χρήση της τεχνικής της προσομοιούμενης ανόπτησης (simulated annealing). Η πρώτη μέθοδος ενώ είναι γρήγορη, απαιτεί ευφυΐα για πρόσδοση scalability ενώ δεν μπορεί να αποφανθεί σε πολυωνυμικό χρόνο για την μη ύπαρξη λύσης. Η δεύτερη μέθοδος από την άλλη δεν αξιοποιεί επιτυχώς τυχόν πληροφορίες τομεοποίησης των δικτύων (domains) ενώ σε πολλές περιπτώσεις υποφέρει από βραδεία σύγκλιση.
- Προσεγγιστικοί αλγόριθμοι που αποφαίνονται σε πολυωνυμικό χρόνο για την ύπαρξη ή μη λύσης οι οποίοι στηρίζονται σε τεχνικές διαχωρισμού γράφων. Ανάλογες τεχνικές είναι οι αραιές κοπές (sparse cuts) που οδηγούν στην εύρεση k-way separators και στην επίλυση προβλημάτων MCF. Το μειονέκτημα τους είναι ότι είναι πιο αργοί σε σύγκριση με τους ευριστικούς αλγορίθμους εφόσον βέβαια υπάρχει εφικτή λύση του προβλήματος.

Γενικά, όλοι οι αλγόριθμοι ενσωμάτωσης ανάγονται εν τέλει σε δύο βασικές φάσεις:

1. Ανάθεση εικονικών κόμβων σε φυσικούς με χρήση άπληστων ευριστικών μεθόδων, π.χ. ανάθεση εικονικών κόμβων υψηλότερων απαιτήσεων επεξεργασίας σε φυσικούς κόμβους του υποστρώματος με περισσότερους διαθέσιμους πόρους, ([7], [14]).

2. Ανάθεση εικονικών ζεύξεων σε φυσικά μονοπάτια με χρήση:

- συντομότερου μονοπατιού π.χ. Dijkstra στην περίπτωση αδιαχώριστης ροής (single-path unsplittable flow), [14] αλλά με χρήση μοναδικής μετρικής για τις απαιτήσεις της εικονικής ζεύξης (π.χ. εύρος ζώνης ή καθυστέρηση ή το μη επαρκές γινόμενό τους). Το [18] προτείνει όμως τρόπο εφαρμογής του αλγορίθμου στην περίπτωση δύο μετρικών (εύρος ζώνης και καθυστέρηση) έτσι ώστε να ικανοποιούνται επιτυχώς και οι δύο περιορισμοί που σχετίζονται με αυτές.

- MCF αλγορίθμων στην περίπτωση διαχωρίσιμων ροών (splittable flows), [6], [7], [8].

Μάλιστα έχουν προταθεί κατανεμημένοι αλγόριθμοι όπως στο [19] που ταυτόχρονα χαρτογραφούν εικονικούς κόμβους και ζεύξεις χωρίς την ύπαρξη κεντρικού ελεγκτή χρησιμοποιώντας κατανεμημένες λογικές, αλλά η αποδοτικότητα και η κλιμάκωση τους δεν είναι ακόμα συγκρίσιμη με αντίστοιχους συγκεντρωτικούς αλγορίθμους.

Προσεγγίσεις προγραμματισμού ακεραίου και μεικτού ακεραίου έχουν εφαρμοστεί σε πολυάριθμα προβλήματα ανάθεσης πόρων και βελτιστοποίησης στην περιοχή των δικτύων. Στο [20], οι συγγραφείς προτείνουν ένα μοντέλο προγραμματισμού ακεραίου για την επίλυση του προβλήματος υπολογισμού VPN δέντρων που σχετίζεται με την ανάθεση εύρους ζώνης σε εικονικά ιδιωτικά δίκτυα VPN. Τεχνικές τυχαιοποιημένης στρογγυλοποίησης για τη χαλάρωση των περιορισμών σε προβλήματα γραμμικού προγραμματισμού και την ανάπτυξη προσεγγιστικών αλγορίθμων προτάθηκαν για πρώτη φορά στο [21].
Στο [6] συγκεκριμένα αναπτύσσεται ένα μοντέλο προγραμματισμού μεικτού ακεραίου για την επίλυση του online προβλήματος ενσωμάτωσης εικονικών δικτύων χωρίς να περιορίζεται ο χώρος του προβλήματος υποθέτοντας άπειρους πόρους για το υπόστρωμα ή συγκεκριμένες τοπολογίες. Ο προσεγγιστικός αλγόριθμος που αναπτύσσεται εφαρμόζεται σε επαυξημένο γραφικά υπόστρωμα έτσι ώστε να τηρούνται και οι περιορισμοί γεωγραφικής τοποθέτησης των εικονικών κόμβων και άρα δέχεται αυξημένο σε μέγεθος δίκτυο σε σχέση με το αρχικό πρόβλημα. Επίσης σημαντικό είναι ότι η αναφορά αυτή είναι η πρώτη που αποδεικνύει ότι είναι επιθυμητός ο συγχρονισμός ανάμεσα στις φάσεις ανάθεσης κόμβων και ζεύξεων για αποδοτικότερες απεικονίσεις.

Στο [1], το οποίο αποτελεί και τη βάση της παρούσας εργασίας, παρουσιάζεται ευριστικός αλγόριθμος ο οποίος στηρίζεται στην ανίχνευση ισομορφισμού μεταξύ υπογράφων με υπαναχώρηση (backtracking) και χαρτογραφεί τους εικονικούς κόμβους και τις εικονικές ζεύξεις στην ίδια φάση. Αποδεικνύεται ότι αυτή η μέθοδος καταλήγει σε καλύτερες χαρτογραφήσεις και είναι γρηγορότερη από την προσέγγιση δύο φάσεων, κυρίως για μεγάλα σε μέγεθος εικονικά δίκτυα με υψηλή κατανάλωση πόρων που είναι δυσκολότερο να απεικονιστούν. Το μειονέκτημά της είναι ότι σε περίπτωση μη ύπαρξης λύσης, η απόκριση μη επιλυσιμότητας είναι πολύ αργή λόγω της φύσης του ευριστικού αλγορίθμου. Το φαινόμενο αυτό αντιμετωπίζεται μερικώς με χρήση έξυπνων heuristics.

Μία εκτενής παρουσίαση των γενικών προκλήσεων που ανακύπτουν στην ανάθεση πόρων στον τομέα της εικονικοποίησης δικτύων γίνεται στο [4], όπου αναφέρονται τα ακόλουθα:

- Ανασκόπηση των πιο πρόσφατων τεχνικών ανάθεσης πόρων για τα εικονικά δίκτυα όπου περιγράφονται ποικίλες στατικές και δυναμικές προσεγγίσεις για επίλυση του προβλήματος.

- Πρόταση μίας νέας αντικειμενικής συνάρτησης για χαρτογράφηση των εικονικών δικτύων.

- Μία περιγραφή και ανάλυση των προβλημάτων ανάθεσης και κατανομής πόρων σε επίπεδο συστήματος βασισμένη σε αυτόνομα συστήματα ή σε συστήματα που στηρίζονται στη θεωρία αυτομάτου ελέγχου με ανάδραση.

- Παρουσίαση υπαρχόντων συστημάτων υλοποίησης της διαχείρισης πόρων στα πλαίσια της εικονικοποίησης δικτύων, όπως είναι το PlanetLab [22], το VINI [23] και το Emulab [12].

- Αναγνώριση ορισμένων σημαντικών ερευνητικών προκλήσεων.

Τέλος, στο [2], προτείνεται μία αρχιτεκτονική χαρτογράφησης εικονικών δικτύων που είναι βασισμένη σε έναν ιεραρχικό, πολύ-ζωνικό (multi-domain) μηχανισμό διαχείρισης. Ο μηχανισμός αυτός επιτρέπει σε ξεχωριστές διαχειριστικές οντότητες υπεύθυνες για την ατομική ανάθεση πόρων εντός της ζώνης επιρροής τους να συνεργάζονται μέσω ενός προτεινόμενου πρωτοκόλλου επικοινωνίας ώστε να εκτελούν παράλληλα την ενσωμάτωση των εικονικών δικτύων. Η διαδικασία ενσωμάτωσης υπακούει σε συγκεκριμένους κανόνες ιεραρχίας, με τον αριθμό των συμμετεχουσών οντοτήτων και το ποσό της ανταλλασσόμενης διαχειριστικής πληροφορίας να μειώνεται κατά τη μετάβαση σε ανώτερα επίπεδα
(αφαίρεση bottom-up).

Αυτό το ιεραρχικό διαχειριστικό πλαίσιο εργασίας σε συνδυασμό με την αποσύνθεση των τοπολογιών των εικονικών δικτύων και τη μετέπειτα σύζευξή τους πάνω από πολλαπλά domain επιτυγχάνει τα ακόλουθα:

- ευνοεί την κλιμάκωση (scalability)
- ελαττώνει την πολυπλοκότητα
- βελτιώνει την αποδοτικότητα
- ικανοποιεί την απαίτηση της κατανεμημένης-παράλληλης ενσωμάτωσης εικονικών δικτύων στο διαμοιραζόμενο υπόστρωμα σε πραγματικό χρόνο (real-time).

➔Είναι λοιπόν φανερό ότι το ζήτημα της αποδοτικής κατανομής πόρων πολλαπλών εικονικών δικτύων ώστε αυτά να ενσωματωθούν σε ένα φυσικό δικτυακό υπόστρωμα έχει απασχολήσει πολλούς ερευνητές και έχει ελκύσει προς επένδυση μεγάλα ποσά χρόνου και χρήματος.

➔Η εργασία αυτή διαθέτει λοιπόν μία ισχυρή ερευνητική βάση αναφοράς και αποτελεί άλλον έναν κρίκο στην ερευνητική αλυσίδα με σκοπό την επίτευξη μίας αποδοτικής και οικονομικής λύσης του γενικού προβλήματος ενσωμάτωσης εικονικών δικτύων.

III. Μοντέλο VN ενσωμάτωσης και διατύπωση προβλήματος

III.1 *Βασικοί ορισμοί*

Παρακάτω περιγράφονται οι γενικοί ορισμοί των βασικών οντοτήτων που εμπλέκονται στο πρόβλημα καθώς και οι εξειδικεύσεις αυτών (όπου χρειάζονται) για την παραγωγή του συγκεκριμένου μοντέλου που θα προσομοιωθεί:

III.1.1 *Ορισμός 1 ➔ Δίκτυο (Network)*

Ένα δίκτυο περιγράφεται από έναν μη-κατευθυνόμενο γράφο G = (N, L, C), όπου N είναι ένα σύνολο κόμβων και L είναι ένα σύνολο ακμών. Κάθε κόμβος ή ακμή $e \in N \cup L$ είναι συσχετισμένα με ένα σύνολο από περιορισμούς
$C(e) = \{C_1(e),\ldots,C_m(e)\}$.
Σύμφωνα με τα παραπάνω, ένα φυσικό δίκτυο (Physical Network-PN) θα περιγράφεται από τον γράφο $G^P = (N^P, L^P, C^P)$, ενώ ένα εικονικό δίκτυο (Virtual Network-VN) από τον γράφο $G^S = (N^V, L^V, C^V)$.

➔Εξειδίκευση ορισμού για γράφο υποστρώματος:

Γράφος του φυσικού δικτύου υποστρώματος (Substrate Graph)

Η δομή αυτή πρακτικά περιέχει τα σύνολα των φυσικών κόμβων και των φυσικών ακμών. Οι φυσικοί κόμβοι μπορούν να είναι είτε τερματικά είτε routers. Κάθε φυσικός κόμβος-τερματικό χαρακτηρίζεται από τη διαθέσιμη cpu και τον αριθμό των εικονικών τερματικών (terminal slices) που μπορεί να σηκώσει. Κάθε φυσικός κόμβος-router χαρακτηρίζεται μόνο από τον αριθμό των εικονικών routers που μπορεί να σηκώσει (router slices), καθώς υποτίθεται ότι η εικονική δρομολόγηση δαπανά αμελητέους πόρους όσον αφορά την υπολογιστική ισχύ και την μνήμη των φυσικών routers. Προφανώς είναι λογικό να θεωρηθεί ότι οι φυσικοί routers μπορούν να διαμεριστούν σε πολύ μεγαλύτερο αριθμό slices από ό,τι τα φυσικά τερματικά, καθώς αυτό συμφωνεί με τη φύση της εικονικής δρομολόγησης και της δικτυακής δομής που χαρακτηρίζει την πλειονότητα των σημερινών φυσικών δικτύων : αυθαίρετη τοπολογία σύνδεσης μεταξύ routers και αστερωτή τοπολογία διασύνδεσης πολλαπλών τερματικών γύρω από κάθε router (1 router-gateway/terminal, multiple terminals/router). Επίσης: routers→ βάρος σε δρομολόγηση, ενώ terminals→ βάρος σε τρέξιμο πολλαπλών βαριών εφαρμογών.

Τέλος, οι φυσικές ακμές χαρακτηρίζονται από το διαθέσιμο bandwidth τους. Μελλοντική (προαιρετική) επέκταση θα μπορούσε να είναι η ένταξη φυσικών κόμβων και ακμών σε διαφορετικά domains ώστε να υπάρχει κάποιου είδους πολιτική διαχείρισης των απεικονίσεων, καθώς το σε ποιο domain ανήκει ένας φυσικός κόμβος ή μία ακμή επηρεάζει στην πραγματικότητα θέματα κόστους και απολαβής και φυσικά την ίδια τη δυνατότητα επιτυχούς χαρτογράφησης.

➔ Εξειδίκευση ορισμού για γράφο εικονικού δικτύου:

Γράφος του εικονικού δικτύου (VN Graph)

Η δομή αυτή είναι παρόμοια με εκείνη του υποστρώματος (παρόμοια δομή, διαχωρισμός κόμβων σε routers-terminals), αλλά φυσικά υποτίθεται ότι είναι πολύ μικρότερη σε μέγεθος από το υπόστρωμα.

Επίσης κάθε terminal έχει μία απαίτηση cpu ενώ οι routers δεν έχουν κάποια απαίτηση, εκτός βέβαια από την ανάγκη ύπαρξης διαθέσιμου φυσικού λογικού router για την απεικόνιση κατ' αναλογία με τα terminals. Επίσης, πρόσθετοι προαιρετικοί περιορισμοί είναι: η απαίτηση κάθε εικονικού κόμβου να απεικονιστεί σε έναν συγκεκριμένο φυσικό ή σε συγκεκριμένη απόσταση από αυτόν και η απαίτηση ενός εικονικού τερματικού να ενσωματωθεί στην φυσική ομάδα τερματικών που σχετίζεται με ένα συγκεκριμένο φυσικό router-gateway.
Εκτός από αυτό, έχουμε και ένα πρόσθετο περιορισμό για τα links (πέραν του Bandwidth) : την απαίτηση max_hops, που δηλώνει το μέγιστο ανεκτό αριθμό hops που μπορεί να περιλαμβάνει το αντίστοιχο φυσικό μονοπάτι-απεικόνιση της virtual link, δηλ. είναι ένας όρος μέγιστης ανεκτής απόστασης-καθυστέρησης.

III.1.2 *Ορισμός 2 → Απεικόνιση-Χαρτογράφηση εικονικού δικτύου (Virtual Network Mapping-VNM)*

Η VNM απεικόνιση ενός εικονικού δικτύου $G^V = (N^V, L^V, C^V)$ πάνω σε ένα φυσικό δίκτυο $G^P = (N^P, L^P, C^P)$ ορίζεται ως η απεικόνιση του G^V πάνω σε ένα υποσύνολο του G^P, έτσι ώστε κάθε εικονικός κόμβος να απεικονίζεται ακριβώς σε έναν φυσικό κόμβο (ίδιου όμως τύπου : terminal→terminal, router→router) και κάθε εικονική ζεύξη πάνω σε ένα (αδιαχώριστη ροή) ή περισσότερα (διαχωρίσιμη ροή) ακυκλικά μονοπάτια του φυσικού δικτύου : $M : G^V \rightarrow (N^P, P^P)$,
όπου P^P είναι ένα υποσύνολο όλων των ακυκλικών μονοπατιών του G^P.
Μία VNM απεικόνιση M χαρακτηρίζεται ως «έγκυρη» εάν ισχύουν τα ακόλουθα:

- όλοι οι περιορισμοί (κόμβοι + ζεύξεις) του εικονικού δικτύου G^V ικανοποιούνται
- συγκεκριμένα, για κάθε εικονική ακμή $l^V = (s^V, t^V) \in L^V$ υπάρχει ένα μονοπάτι $p(s^P, t^P) \in P^P$ με $M(s^V) = s^P$ και $M(t^V) = t^P$ του οποίου το bottleneck bandwidth υπερβαίνει το απαιτούμενο bandwidth της εικονικής ζεύξης και το μήκος σε αριθμό φυσικών ζεύξεων είναι μικρότερο ή ίσο με το απαιτούμενο max_hops, στο σενάριο της αδιαχώριστης ροής. Ανάλογα για την περίπτωση της διαχωρίσιμης ροής, υπάρχει ένα σύνολο μονοπατιών-ροών PF^P, υποσύνολο του συνόλου των ακυκλικών μονοπατιών του G^P, όπου κάθε φυσική ροή ικανοποιεί την απαίτηση max_hops και το άθροισμα των bandwidths που μεταφέρουν οι ροές αυτές ικανοποιεί την απαίτηση bandwidth της εικονικής ζεύξης.

Μία VNM απεικόνιση μπορεί να αποσυντεθεί σε ανάθεση κόμβων και ανάθεση ζεύξεων ως ακολούθως:

- Ανάθεση-απεικόνιση κόμβων : $M_N : N^V \rightarrow N^P$
- Ανάθεση-απεικόνιση ζεύξεων : $M_L : L^V \rightarrow P^P$

Συγκεκριμένα, στην εργασία προσομοιώθηκε μόνο το σενάριο της αδιαχώριστης ροής, ενώ στο μέρος IV.5.1 προτείνεται και ένας θεωρητικός αλγόριθμος δημιουργίας ροών που ικανοποιούν τους περιορισμούς στο σενάριο της διαχωρίσιμης ροής για μελλοντική έρευνα.

III.1.3 *Ορισμός 3 ➔ Κόστη χαρτογράφησης εικονικού δικτύου (VNM Costs) - πλευρά παρόχου υπηρεσίας*

Έστω ένα εικονικό δίκτυο G^V με περιορισμούς $C^V = \{C_1^V,\ldots,C_m^V\}$, ένα φυσικό δίκτυο G^P με περιορισμούς $C^P = \{C_1^P,\ldots,C_m^P\}$ και μία απεικόνιση $M(G^V)$ του G^V πάνω στο G^P. Το συνολικό κόστος της απεικόνισης $M(G^V)$ δίνεται από :

$$\text{cost}(M(G^V)) = \sum_{i=1}^{n} a_i * \text{cost}_i(M(G^V))$$

Ο ορισμός των συναρτήσεων κόστους-περιορισμών ποικίλουν και εξαρτώνται από τη φύση του αντίστοιχου περιορισμού. Οι περιορισμοί και οι αντίστοιχες συναρτήσεις κόστους λήφθηκαν υπόψη στην εργασία είναι η ακόλουθοι:

- Εύρος ζώνης / ζεύξη : $\text{cost}_1(M(G^V)) = \sum_{l \in LV} C_1^V(l) * length(M(l))$
- Υπολογιστική ισχύς / τερματικό : $\text{cost}_2(M(G^V)) = \sum_{\substack{n \in NV, \\ n->term}} C_2^V(n)$
- Αριθμός slices / τερματικό : $\text{cost}_3(M(G^V)) = \sum_{\substack{n \in NV, \\ n->term}} C_3^V(n)$
- Αριθμός slices / router : $\text{cost}_4(M(G^V)) = \sum_{\substack{n \in NV, \\ n->router}} C_4^V(n) + \sum_{\substack{n \in NP, \\ n->router \\ transit}} C_4^P(n)$

Όσον αφορά τους συντελεστές κόστους ανά πόρο a_i, i=1,..,n, n=4 στην εργασία λήφθηκαν όλοι ίσοι με 1 για διευκόλυνση. Φυσικά αυτές οι παράμετροι κόστους μπορούν να τροποποιηθούν κατά βούληση ή ακόμα και να συσχετιστούν όχι απλά με τον πόρο όπου αντιστοιχούν, αλλά και με το συγκεκριμένο δικτυακό στοιχείο που τον φιλοξενεί (π.χ. διαφοροποίηση κόστους/bandwidth για διαφορετικές ζεύξεις ή κόστος/CPU για διαφορετικά τερματικά). Σημείωση: transit routers➔ ενδιάμεσοι φυσικοί routers που χρησιμοποιούνται για τη συντήρηση των φυσικών μονοπατιών όπου απεικονίζονται οι εικονικές ζεύξεις.

III.1.4 *Ορισμός 4 ➔ Αίτηση εικονικού δικτύου (Virtual Network Request-VNR) – Ουρές αιτήσεων (VNR queues)*

Μία αίτηση ενσωμάτωσης εικονικού δικτύου $r_i = (G_i^V, a_i, l_i)$ αποτελείται από ένα εικονικό δίκτυο G_i^V, έναν χρόνο άφιξης a_i και μία διάρκεια ζωής l_i. Ο χρόνος άφιξης είναι ο χρόνος στον οποίο η VNR εισέρχεται στο σύστημα και πρέπει να απεικονιστεί πάνω στο PN και η διάρκεια ζωής δείχνει την χρονική περίοδο που θα διαρκέσει η ενσωμάτωση και παρουσία του εικονικού δικτύου πάνω στο φυσικό υπόστρωμα. Επειδή δεν μπορούν όλες οι VNR να ικανοποιούνται κατευθείαν με το που αφικνούνται καθώς μπορεί να μην είναι δυνατή εκείνη την στιγμή η ανάθεση των πόρων του δικτύου στο εικονικό, τα VN's που περιέχονται στις VNR's μπορεί να ξεκινήσουν αργότερα. Αυτή η καθυστέρηση άφιξης-έναρξης λειτουργίας μπορεί προαιρετικά να εκληφθεί ως ένα επιπλέον κόστος για τον πάροχο υπηρεσίας.

Οι αιτήσεις με το που καταφθάνουν εισέρχονται σε μία ουρά αναμονής όπου και περιμένουν την ικανοποίησή τους. Αν λήξουν χωρίς απεικόνιση, διαγράφονται από την ουρά. Αν απεικονιστούν, ανατίθενται οι πόροι που τους αναλογούν, διαγράφονται από την ουρά αναμονής και μεταβαίνουν σε μία άλλη ουρά για εξυπηρέτηση,

περιμένοντας τον τερματισμό της υπηρεσίας. Όταν αυτό συμβεί, οι πόροι απελευθερώνονται και οι αιτήσεις διαγράφονται και από την συγκεκριμένη ουρά.
Περισσότερες λεπτομέρειες για την πολιτική διαχείρισης των αιτήσεων και των ουρών αναμονής περιγράφονται στο μέρος V όπου αναπτύσσονται λεπτομέρειες της προσομοίωσης.

III.1.5 *Ορισμός 5 ➔ Εναπομείναντας γράφος (Residual Graph)*

Έστω ότι δίνεται ένα φυσικό δίκτυο G^P, ένα εικονικό δίκτυο G^V και μία απεικόνιση του G^V πάνω στο G^P. Παίρνουμε τον εναπομείναντα γράφο $G_{res}{}^P$ του G^P αφαιρώντας τις χωρητικότητες κάθε εικονικού κόμβου και ζεύξης του G^V από τις χωρητικότητες των φυσικών κόμβων και ακμών του G^P όπου αυτά είναι απεικονισμένα. Επίσης, για κάθε εικονικό κόμβο αφαιρούμε αριθμητικά ένα slice από τον φυσικό κόμβο όπου αυτός έχει απεικονιστεί (αφού τώρα πια το slice έχει αφιερωθεί στον εικονικό κόμβο).

Όσον αφορά τους φυσικούς κόμβους, αν θεωρήσουμε τυχαία έναν πόρο $c_i(n^P)$ ενός φυσικού κόμβου n^P (ο πόρος μπορεί να είναι CPU, Memory, Slice number), τότε η εναπομείνασα χωρητικότητα του φυσικού κόμβου μετά την διεκπεραίωση ενός αριθμού χαρτογραφήσεων ορίζεται ως:

$$R_{Ni}(n^P) = c_i(n^P) - \sum_{\forall n^V \uparrow n^P} c_i(n^V) \ ,$$

όπου x↑y συμβολίζει ότι ο εικονικός κόμβος x έχει απεικονιστεί στον φυσικό κόμβο y (υπενθύμιση: ένας φυσικός κόμβος μπορεί να φιλοξενεί πολλαπλούς εικονικούς, υπό τον όρο ο καθένας από αυτούς να ανήκει σε διαφορετικό εικονικό δίκτυο).
Φυσικά, λόγω της πολλαπλότητας των πόρων η εναπομείνασα χωρητικότητα ενός κόμβου μπορεί να εκφραστεί διανυσματικά με συνιστώσες τις χωρητικότητες που αντιστοιχούν σε κάθε τύπο πόρου: $\mathbf{R_N}=(R_{N1,..,} R_{Nm})$)

Όσον αφορά τις φυσικές ζεύξεις, ο μοναδικός πόρος που έχει υποτεθεί είναι το bandwidth $b(l^P)$ μίας φυσικής ζεύξης l^P. Η εναπομείνασα χωρητικότητα της ζεύξης μετά την διεκπεραίωση ενός αριθμού χαρτογραφήσεων ορίζεται ως:

$$R_L(l^P) = b(l^P) - \sum_{\forall l^V \uparrow l^P} b(l^V) \ ,$$

όπου x↑y συμβολίζει ότι η φυσική ζεύξη y ανήκει στο φυσικό μονοπάτι στο οποίο έχει απεικονιστεί η εικονική ζεύξη x.

Τέλος, μπορούμε να βασιστούμε στον παραπάνω ορισμό και να ορίσουμε την εναπομείνασα χωρητικότητα ενός φυσικού μονοπατιού P που ανήκει στο σύνολο μονοπατιών P^P, ως εξής: $R_L(P) = \min_{l^P \in P} R_L(l^P)$

Φυσικά τα ακριβώς αντίθετα βήματα και υπολογισμοί πραγματοποιούνται στην περίπτωση της αποδέσμευσης πόρων.

III.2 *Τελική έκφραση VNM προβλήματος βάσει ορισμών*

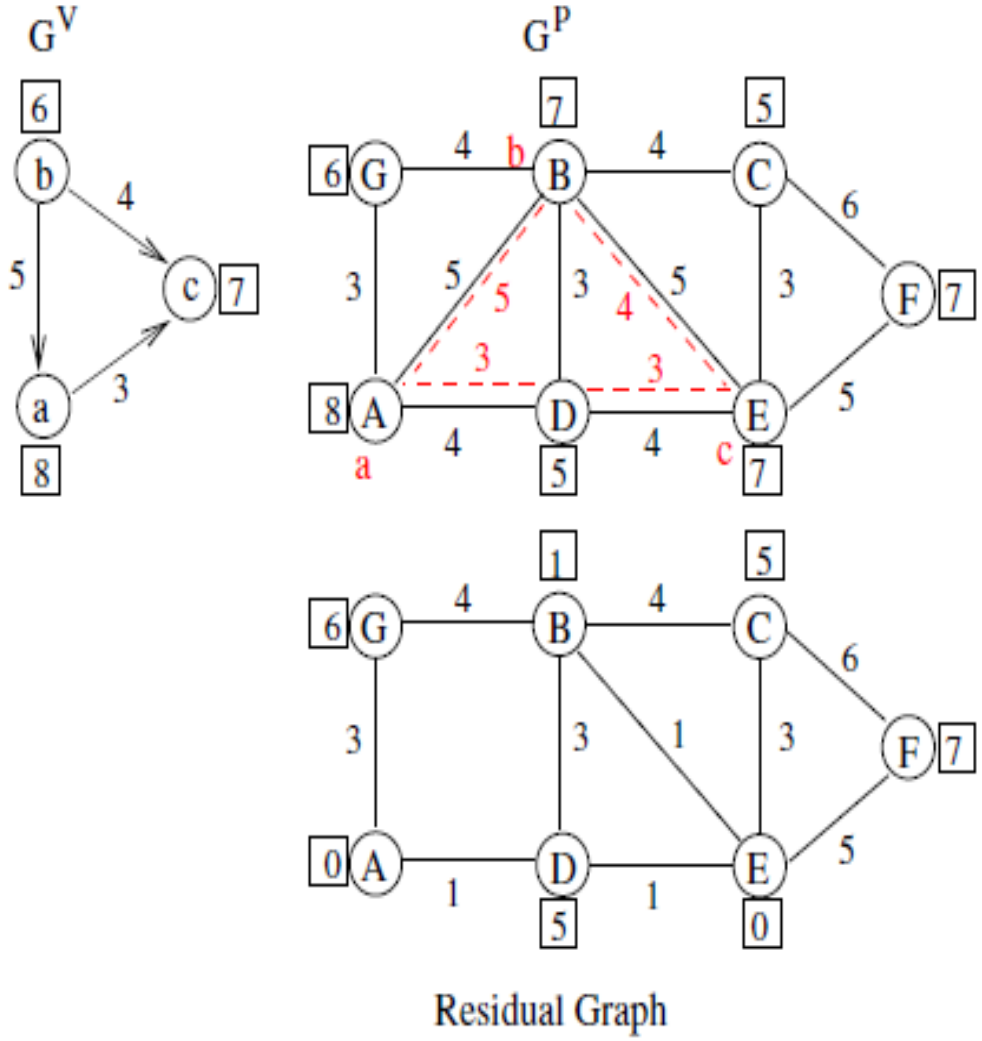
➔Σχεδιασμός και υλοποίηση αλγορίθμου ενσωμάτωσης σε ένα δεδομένο και τυχαία παραγόμενο υπόστρωμα substrate, πολλαπλών εικονικών δικτύων VNs που περιέχονται σε δυναμικά αφικνούμενες και μεταβλητής διάρκειας εικονικές αιτήσεις VNRs, με στόχο την παραγωγή αποδοτικών και οικονομικών απεικονίσεων VNMs, που μεγιστοποιούν την εναπομείνασα-residual χωρητικότητα του φυσικού υποστρώματος και ελαχιστοποιούν τα κόστη χαρτογράφησης VNM costs για την πλευρά του παρόχου υπηρεσίας.

III.3 *Ένα αρχικό παράδειγμα ενσωμάτωσης*

Ένα παράδειγμα VNM ενσωμάτωσης και εύρεσης αποδεκτής λύσης στο βασικό πρόβλημα περιγράφεται παρακάτω και είναι παρμένο από το [1]:

Ζητούμενο➔απεικόνιση του εικονικού γράφου GV στο φυσικό γράφο GP

Αποδεκτή απεικόνιση-λύση➔
κόμβοι: a->A, b->B, c->E
ζεύξεις: (a-c)->(A-D-E), (a-b)->(A-B), (b-c)->(B-E)



Σχήμα 2: Παράδειγμα ενσωμάτωσης εικονικού δικτύου σε υπόστρωμα (από [1])

Φυσικά ο εικονικός γράφος δεν είναι αναγκαίο να διαθέτει κατευθυνόμενες ακμές (μπορεί να είναι full-duplex όπως στην εργασία). Οι εναπομείναντες πόροι (CPU, bandwidth) του φυσικού γράφου φαίνονται στο σχήμα. Προσοχή πρέπει να δοθεί στο γεγονός ότι απεικόνιση εικονικής ζεύξης σε φυσικό μονοπάτι συνεπάγεται δέσμευση του απαραίτητου bandwidth από κάθε ζεύξη που περιλαμβάνεται σε αυτό.

IV. Ο Αλγόριθμος

IV.1 *Η πηγή του αλγορίθμου*

Ο αλγόριθμος που προτιμήθηκε προς ανάπτυξη είναι ο vnmFlib (σχήμα 3) και παρουσιάστηκε για πρώτη φορά στο [1]. Οι λόγοι προτίμησής του καθώς και η βάση του αναλύονται παρακάτω.

IV.1.1 *Γιατί προτιμήθηκε ο αλγόριθμος vnmFlib*

Ο αλγόριθμος που υιοθετήθηκε χρησιμοποιεί ευριστική υπαναχώρηση (backtracking) και βασίζεται σε μία μέθοδο ανίχνευσης ισομορφισμού υπογράφων, χαρτογραφώντας τους εικονικούς κόμβους και τις εικονικές ζεύξεις στην ίδια φάση. Το πλεονέκτημα αυτής της μονοφασικής προσέγγισης είναι το ότι οι περιορισμοί των κόμβων και των ζεύξεων λαμβάνονται υπόψη σε κάθε βήμα της χαρτογράφησης. Όταν ανιχνευτεί μία κακή απόφαση χαρτογράφησης-απεικόνισης που οδηγεί σε αδιέξοδο, μπορεί απλά να αναιρεθεί με υπαναχώρηση στην τελευταία έγκυρη απόφαση χαρτογράφησης. Σε αντίθεση, ανάλογες μέθοδοι δύο σταδίων είναι αναγκασμένες να επαναχαρτογραφούν όλες τις ζεύξεις το οποίο είναι πολύ δαπανηρό σε όρους χρόνου εκτέλεσης ([7]). Επίσης, με τον vnmFlib επιτρέπεται και ενθαρρύνεται η χρήση έξυπνων heuristics που μειώνουν το πεδίο αναζήτησης του αλγορίθμου και βελτιώνουν σημαντικά το χρόνο εκτέλεσης.

IV.1.2 *Θεωρητική βάση vnmFlib*

Το NP-πλήρες πρόβλημα ανίχνευσης ισομορφισμού υπογράφων [24] μπορεί να αναχθεί ελαττούμενο στο VNM (Virtual Network Mapping) πρόβλημα αναθέτοντας έναν μοναδικό περιορισμό καθυστέρησης ίσο με 1 σε κάθε φυσική και εικονική ζεύξη. Ο περιορισμός καθυστέρησης ικανοποιείται μόνο αν η καθυστέρηση του φυσικού μονοπατιού δεν ξεπερνά την καθυστέρηση της εικονικής ζεύξης που έχει απεικονισθεί πάνω σε αυτό. Μία απεικόνιση VNM ενός εικονικού γράφου G^V πάνω στο φυσικό δικτυακό υπόστρωμα G^P απεικονίζει κάθε εικονικό κόμβο n^V που ανήκει στο N^V σε έναν κόμβο n^P που ανήκει στο N^P, και κάθε εικονική ζεύξη l^V που ανήκει στο L^V σε ένα φυσικό μονοπάτι p που ανήκει στο P^P. Για να δημιουργηθεί μία έγκυρη VNM πρέπει η απαίτηση καθυστέρησης να ικανοποιείται. Άρα πρέπει κάθε l^V να έχει απεικονισθεί σε ένα p μήκους$\leq$1 και συνεπώς η τελική VNM αποτελεί έναν γραφικό ισομορφισμό του G^V στο G^P, αφού κάθε εικονική ζεύξη είναι απεικονισμένη σε ακριβώς μία φυσική και κάθε εικονικός κόμβος σε ακριβώς έναν φυσικό $\Leftrightarrow$ 1-1 αντιστοιχία εικονικών-φυσικών στοιχείων. Επομένως, είναι λογικό το ότι χρησιμοποιήθηκε η παραπάνω βάση του αλγορίθμου για την επίλυση του VNM προβλήματος.

IV.1.3 *Επέκταση vnmFlib σε σχέση με τη βάση του*

Ο αλγόριθμος που χρησιμοποιήθηκε αποτελεί επέκταση του vFlib αλγορίθμου ταιριάσματος γράφων [25]. Η κύρια διαφορά είναι ότι ο vnmFlib επιτρέπει την χαρτογράφηση των εικονικών ζεύξεων σε φυσικά μονοπάτια μήκους μικρότερου από μία προκαθορισμένη τιμή ε-max_hops σε όρους αλμάτων (hops).
Φυσικά αν ε=1 τότε αναγόμαστε στο πρόβλημα βάσης, δηλ. αναζητείται ένας επιτυχημένος ισομορφισμός μεταξύ του εικονικού και του φυσικού γράφου και άρα

μία ορθή αναπαραγωγή του εικονικού network pattern πάνω στο φυσικό substrate network. Μία άλλη κύρια διαφορά είναι ότι ο vnmFlib ελέγχει τους περιορισμούς δικτύου σε κάθε βήμα χαρτογράφησης. Οι περιορισμοί αυτοί είναι για τους φυσικούς κόμβους: CPU για τερματικά και αριθμός slices διαμοιρασμού για όλους τους κόμβους και για τις φυσικές ζεύξεις το bandwidth.
Αντίστοιχα οι εικονικοί κόμβοι απαιτούν CPU ως τερματικά και οι εικονικές ζεύξεις bandwidth και μέγιστο αριθμό hops για απεικόνιση. Προαιρετικές μελλοντικές απαιτήσεις: προτιμητέα θέση φυσικού κόμβου απεικόνισης ή μέγιστη απόσταση από αυτήν, προτιμητέο σύνολο τερματικών πίσω από επιθυμητή gateway.

IV.2 *Περιγραφή βασικού αλγορίθμου*

Ο αλγόριθμος προσπαθεί να χτίσει σταδιακά μία έγκυρη λύση VNM προσθέτοντας κλιμακωτά κόμβους και ζεύξεις του G^V σε έναν αρχικά κενό υπογράφο G^V_{sub} του G^V.
Κατά τη διάρκεια της διαδικασίας χαρτογράφησης ο αλγόριθμος καθιστά ελέγχει ότι η απεικόνιση $M(G^V_{sub})$ είναι μία έγκυρη VNM του G^V_{sub} πάνω στον G^P.
Σε περίπτωση χαρτογραφικού αδιέξοδου λόγω μίας κακής επιλογής απεικόνισης αξιοποιείται η δυνατότητα backtracking στην τελευταία valid απόφαση.
Ο αλγόριθμος τερματίζει όταν το G^V_{sub} καλύπτει πλήρως το G^V και επιστρέφει τελικά τη $M(G^V_{sub})$, που είναι μία έγκυρη VNM του G^V_{sub} πάνω στο G^P.

Το βασικό σώμα του αλγορίθμου παρουσιάζεται παρακάτω:

Algorithm 1 vnmFlib(G^V_{sub}, $M(G^V_{sub})$, G^V, G^P)

Require: a VN G^V, a PN G^P and a subgraph G^V_{sub} of G^V

```
C = ← genneigh(G^V_sub, G^V, G^P)
for each (n^V, n^P) in C do
    if valid(M(G^V_sub), (n^V, n^P), G^P) then
        create G^V_sub and M(G^V_sub) by adding (n^V, n^P)
        vnmFlib(G^V_sub, M(G^V_sub), G^V, G^P_res)
    end if
    if G^V_sub == G^V then
        return M(G^V_sub)
    end if
end for
```

Σχήμα 3: Ο αλγόριθμος vnmFlib (από [1])

IV.3 *Παραδείγματα εφαρμογής vnmFlib*

IV.3.1 *Παράδειγμα 1*

Εδώ εξηγείται πώς δημιουργήθηκε η απεικόνιση που παρουσιάζεται στο σχήμα 2 (και εξηγείται ανάλογα και στο [1]). Για διευκόλυνση και για ταίριασμα με το παράδειγμα που χρησιμοποιήθηκε στο [1] υποτέθηκε ότι το εικονικό δίκτυο αποτελείται από τρία διασυνδεδεμένα τερματικά a,b,c ενώ το φυσικό δίκτυο από 7 διασυνδεδεμένα τερματικά A,B,C,D,E,F,G. Στο σχήμα δεν παρουσιάζονται τα slices παρά μόνο οι πόροι CPU και BW (συμφωνία με [1]). Η απαίτηση max_hops των εικονικών ζεύξεων ισούται με 2.

Αυτό είναι προφανώς ένα αρχικό παράδειγμα κατανόησης, καθώς στην επίσημη προσομοίωση που εκτελέστηκε χρησιμοποιήθηκαν τοπολογίες τύπου αστεριών τερματικών γύρω από routers-gateways και arbitrary διασύνδεσης routers (ανάλογο παράδειγμα παρουσιάζεται στο επόμενο μέρος με τα αποτελέσματα της προσομοίωσης) και χρησιμοποιήθηκαν κάποιες τροποποιήσεις του vnmFib. Αυτά όμως θα εξηγηθούν στο παράδειγμα 2 αλά και από τον κώδικα του παραρτήματος.

Στο δοσμένο παράδειγμα λοιπόν, η διαδικασία εφαρμογής του αλγορίθμου παρουσιάζεται παρακάτω, όπως αναφέρεται και στο [1]:

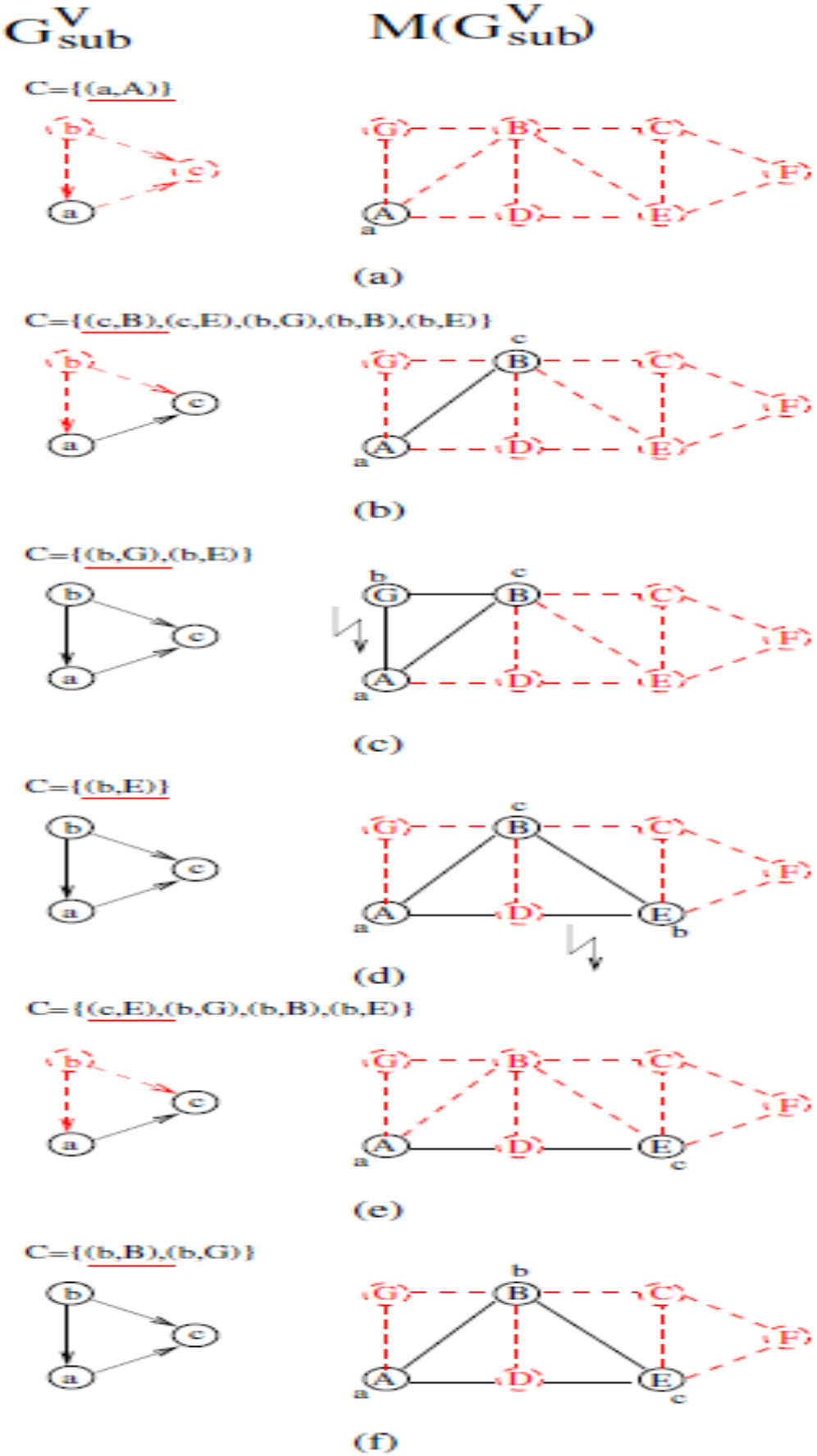
Στο πρώτο βήμα γεννάται ένα σύνολο C από ζεύγη-αντιστοιχίσεις κόμβων (n^V,n^P) με $n^V \in N^V$ και $n^P \in N^P$, από τη συνάρτηση *genneigh()* (σχ.3, γραμμή 1). Ο αλγόριθμος προσθέτει το n^V στο G_{sub}^V και το n^P στο $M(G_{sub}^V)$ αντίστοιχα (σχήμα 4a). Τώρα η συνάρτηση *valid()* (σχ.3, γραμμή 3) ελέγχει αν η συνεπαγόμενη χαρτογράφηση $M(G_{sub}^V)$ είναι έγκυρη. Σε θετική περίπτωση οι κόμβοι προστίθενται στο G_{sub}^V και στο $M(G_{sub}^V)$ (σχ.3, γραμμή 4) και ο vnmFlib καλείται για τον αντίστοιχο εναπομείναντα φυσικό γράφο G_{res}^P και τους καινούριους υπογράφους G_{sub}^V και $M(G_{sub}^V)$ (σχ.3, γραμμή 5). Αλλιώς, η συνθήκη τερματισμού ελέγχεται (σχ.3, γραμμή 7) και αν αποτύχει, το επόμενο ζεύγος κόμβων του C τίθεται υπό εξέταση. Αν επιτύχει τότε επιστρέφεται η έγκυρη πλήρης χαρτογράφηση $M(G_{sub}^V)$ του εικονικού δικτύου (σχ.3, γραμμή 8). Αυτό συμβαίνει και στην εν λόγω περίπτωση όπου η *valid()* επιστρέφει true και άρα ο vnmFlib καλείται με το νεοσχηματισθέν G_{sub}^V και $M(G_{sub}^V)$.

Παράγεται ξανά ένα διάνυσμα ζευγών κόμβων και το πρώτο ζεύγος κόμβων (c,B) προστίθενται στο G_{sub}^V και $M(G_{sub}^V)$ (σχήμα 4b). Τώρα πρέπει να ευρεθεί ένα φυσικό μονοπάτι του G^P για κάθε μία εκ των εικονικών ζεύξεων που συνδέουν τον virtual κόμβο c με τον G_{sub}^V, και να προστεθεί στην χαρτογράφηση εφόσον βέβαια ικανοποιεί όλους τους περιορισμούς. Στο παράδειγμα η μόνη εικονική ζεύξη που συνδέει τον c με τον G_{sub}^V είναι η (a-c) και ο αλγόριθμος προσθέτει το μονοπάτι (A-B) στην χαρτογράφηση καθώς ικανοποιεί τον περιορισμό εύρους ζώνης. Η χαρτογράφηση είναι έγκυρη και ο αλγόριθμος προχωρά με την γένεση ενός καινούριου συνόλου ζευγών κόμβων.

Ακολούθως ο αλγόριθμος απεικονίζει τον virtual κόμβο b στον physical κόμβο G. Αυτήν την φορά η συνάρτηση *valid()* αποτυγχάνει επειδή δεν υπάρχει μονοπάτι από τον G στον A στο G^P με αρκετό εύρος ζώνης (σχήμα 4c). Εξετάζει λοιπόν το επόμενο διαθέσιμο ζεύγος και αποτυγχάνει για τον ίδιο λόγο (σχήμα 4d). Αφού το σύνολο πιθανών απεικονίσεων C έχει πλέον εξαντληθεί, ο αλγόριθμος τσεκάρει τη συνθήκη τερματισμού (σχ.3, γραμμή 7), αποτυγχάνει, εκτελεί ένα βήμα υπαναχώρησης στην τελευταία έγκυρη απεικόνιση (σχήμα 4b) και προσπαθεί τώρα να εφαρμόσει την απεικόνιση (c,E) γεγονός που οδηγεί σε έγκυρη χαρτογράφηση (σχήμα 4e).

Ξανά ο αλγόριθμος υπολογίζει ένα σύνολο ζευγών κόμβων και απεικονίζει το πρώτο (σχήμα 4f). Αυτή τη φορά η χαρτογράφηση είναι έγκυρη και ο αλγόριθμος την επιστρέφει (σχ.3, γραμμή 8), αφού το G_{sub}^V καλύπτει πλήρως το G^V και ο έλεγχος τερματισμού της γραμμής 7 του σχ.3 επιτυγχάνει.

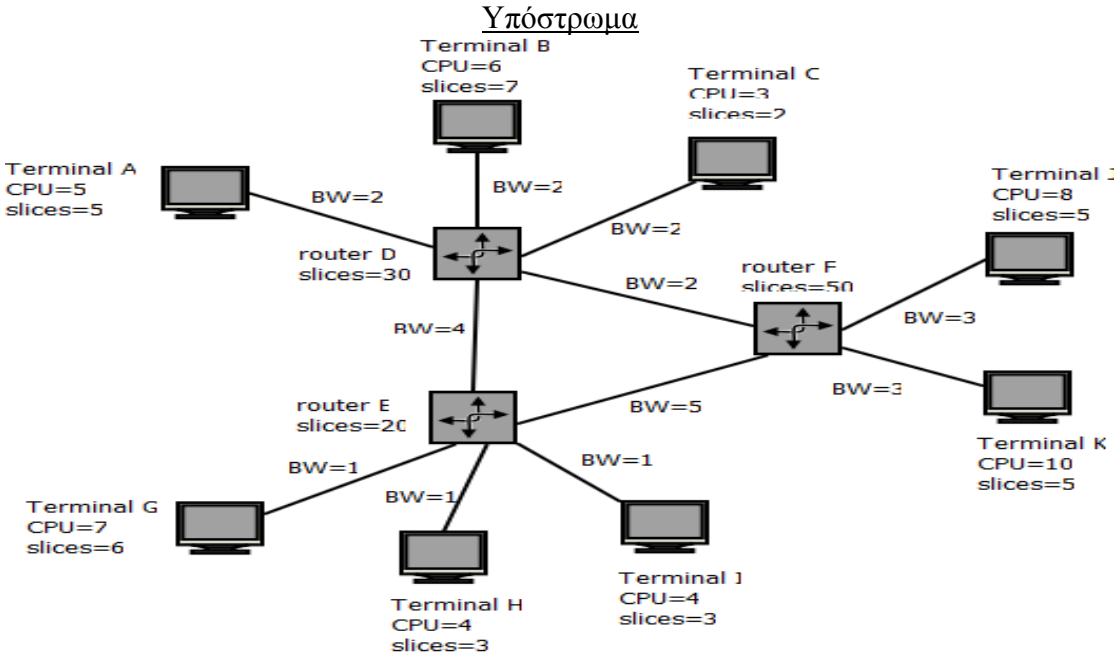
➔Η παραπάνω διαδικασία περιγράφεται αναλυτικά γραφικά στο ακόλουθο σχήμα (4) παρμένο από το [1]:



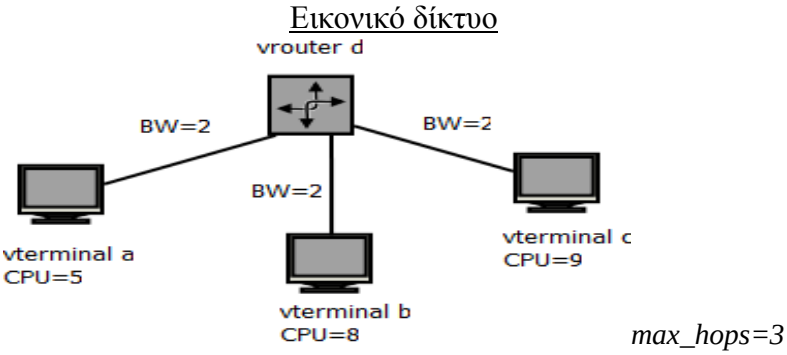
Σχήμα 4: διαδικασία χαρτογράφησης vnmFlib για το παράδειγμα 1 (από [1])

IV.3.2 *Παράδειγμα 2*

Σε αναλογία με το παράδειγμα 1 αλλά με διαχωρισμό των κόμβων αναλόγως τύπου (terminal-router) και με το πρόσθετο σημαντικό constraint των slices αναπτύσσεται το ακόλουθο παράδειγμα απεικόνισης ενός εικονικού δικτύου τερματικών-δρομολογητών σε ένα φυσικό υπόστρωμα τερματικών-δρομολογητών:



Σχήμα 5: Αρχικό υπόστρωμα για το παράδειγμα 2



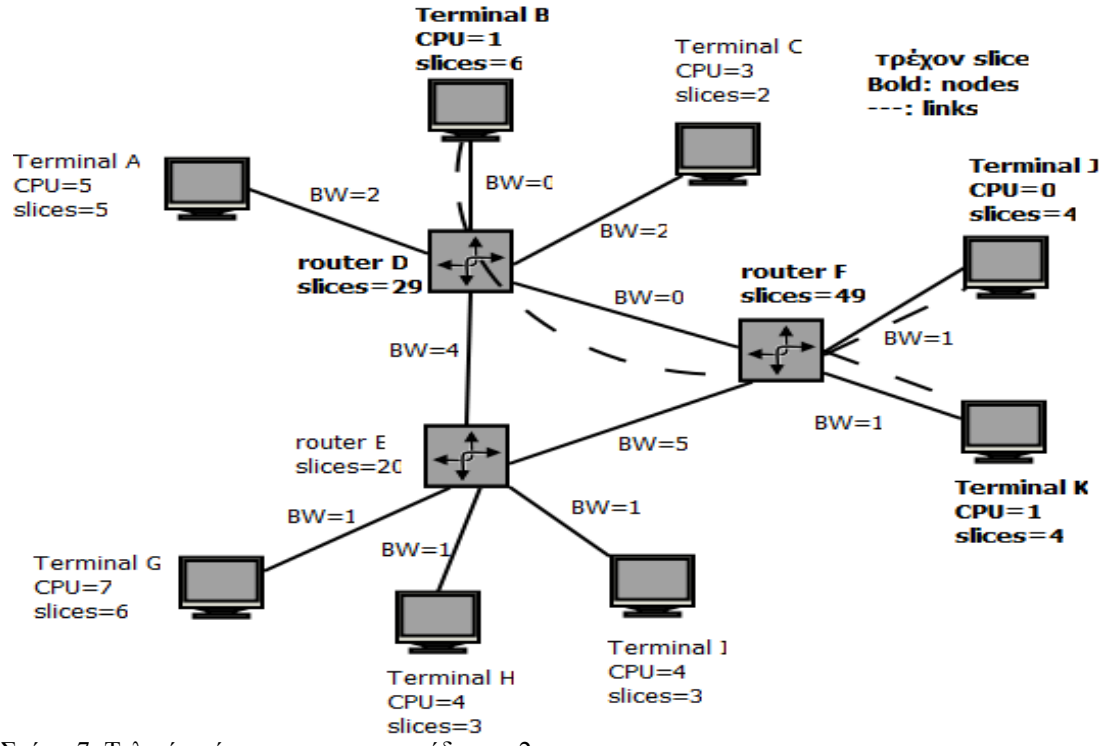
Σχήμα 6: Εικονικό δίκτυο προς ενσωμάτωση για το παράδειγμα 2

➔Βήματα εφαρμογής vnmFlib
(με χρήση optimization και ταξινόμησης ζευγών ώστε ζεύγη με πιο απαιτητικούς εικονικούς κόμβους και πιο πλούσιους φυσικούς να προτιμώνται)

- vnmFlib καλείται
- C={(c,K),(b,K),(b,J),(a,K),(a,J),(a,G)(a,B),(a,A),(d,F)(d,D)(d,E)}
  (όλα τα πιθανά ζεύγη λαμβάνονται υπόψη στο πρώτο στάδιο)
- (c,K) valid
- Creation - adding (c,K)
- vnmFlib καλείται
- C={(d,F)(d,D)(d,E)} (d μόνος γείτονας του c που δεν ανήκει σε G_{sub}^{V})

- (d,F) valid, μονοπάτι (F-K) για απεικόνιση της (c-d)
- Creation - adding (d,F)
- vnmFlib καλείται
- C={(b,J), (a,G)(a,B),(a,A)} (b,a γείτονες των c,d που δεν ανήκουν σε G_{sub}^{V})
- (b,J) valid, μονοπάτι (F-J) για απεικόνιση της (b-d)
- Creation - adding (b,J)
- C={(a,G)(a,B),(a,A)} (a μόνος κόμβος που δεν έχει ακόμη ενταχτεί σε G_{sub}^{V})
- (a,G) not valid (δεν υπάρχει κατάλληλο φυσικό μονοπάτι σύνδεσης!!!)
- (a,B) valid, μονοπάτι (B-D-F) για απεικόνιση της (a-d)
- Creation – adding (a,B)
- vnmFlib καλείται
- άδειο C (όλοι οι εικονικοί κόμβοι έχουν απεικονιστεί)
- Συνθήκη τερματισμού ελέγχεται➔επιτυγχάνει!!!
- Τέλος με επιστροφή παρακάτω έγκυρης χαρτογράφησης:
  Κόμβοι: a→B, b→J, c→K, d→F
  Ζεύξεις: (a-d)→(B-D-F), (b-d)→(J-F), (c-d)→(K-F)

Εναπομείναν υπόστρωμα κατόπιν απεικόνισης
(και παρουσίαση τελικού καταληφθέντος slice)



Σχήμα 7: Τελικό υπόστρωμα για το παράδειγμα 2

Προσοχή: καταλαμβάνονται και slices των ενδιάμεσων routers (δηλ. εκείνων που δεν είναι επίσημα απεικονισμένοι) όμως αυτό είναι αναγκαίο για ορθή δρομολόγηση και συντήρηση φυσικών μονοπατιών!
Επίσης σε περίπτωση θεωρητικής αστοχίας της φυσικής ζεύξης (D-F)➔δυνατότητα χρήσης (D-E-F) ➔όχι παραβίαση max_hops=3. Άλλη περίπτωση: αστοχία φυσικού κόμβου B➔backtracking και προτίμηση κόμβου A.

IV.4 *Λεπτομέρειες αλγορίθμου*

IV.4.1 *Βασικός περιορισμός αλγορίθμου και τρόπος αντιμετώπισης*

Προσοχή πρέπει να δοθεί στο γεγονός ότι ο αλγόριθμος vnmFlib είναι ευριστικός-αναδρομικός. Το πρόβλημα που καλείται να επιλύσει βρίσκοντας μία αποδεκτή οικονομική λύση ενσωμάτωσης είναι NP-hard. Συνεπώς η απόφαση για το αν μία τέτοια λύση υπάρχει (αποκρισιμότητα) μπορεί θεωρητικά να χρειαστεί εκθετικό (σε σχέση με το μέγεθος της εισόδου) χρόνο σε ένα υπολογιστικό σύστημα.

Η χειρότερη θεωρητική περίπτωση είναι η απαίτηση εύρεσης ενός ισομορφισμού (max_hops=1) του εικονικού δικτύου πάνω στο υπόστρωμα, ο οποίος συχνά δεν υπάρχει κυρίως για μεγάλα σε μέγεθος εικονικά δίκτυα. Τότε ο αλγόριθμος θα έπρεπε να διανύσει ολόκληρο το δέντρο αναζήτησης το οποίο έχει πολυπλοκότητα της τάξης του $O(|N^P|!|N^V|)$.

Για το λόγο αυτό στην πράξη πρέπει να τεθεί ένα άνω όριο στον αριθμό των κλήσεων της βασικής αναδρομικής συνάρτησης του αλγορίθμου (όπως και στο [1]), του οποίου η υπέρβαση οδηγεί στο χαρακτηρισμό της χαρτογράφησης ως αποτυχημένης. Θεωρητικά, ένα καλό όριο που προτιμήθηκε στην παρούσα εργασία είναι το $[|N^V|x(|N^V|+1)/2]$ και βασίζεται στην εξής λογική: Για κάθε valid απεικόνιση κόμβου που ευρίσκεται επιτρέπονται τόσα backtracks όσο είναι το τρέχον $|N^{GsubV}|$, δηλ. αν υπάρχουν δύο valid απεικονίσεις μέσα στο VNM, επιτρέπονται δύο backtracks. Άρα μέγιστος αριθμός επιτρεπόμενων backtracks=$(1+2+\ldots+(|N^V|-1))=$ $[|N^V|x(|N^V|-1)/2]$. Άρα μέγιστος αριθμός κλήσεων = $|N^V|$ valid mappings-creations+ $[|N^V|x(|N^V|-1)/2]$ backtracks ➔ max_flib_calls = $[|N^V|x(|N^V|+1)/2]$. Αυτή η δυσμενής περίπτωση παρουσιάζεται συχνά όταν υπάρχει πολύ υψηλός φόρτος εικονικών αιτήσεων (τόσο σε αριθμό όσο και σε απαιτήσεις λειτουργίας) και το υπόστρωμα βρίσκεται πολύ κοντά στα όρια της χωρητικότητάς του. Αυτό βέβαια είναι λογικό λόγω της ευριστικής φύσης του αλγορίθμου, οπότε στις περιπτώσεις αυτές θα ήταν καλύτερο να χρησιμοποιηθούν προηγμένοι αλγόριθμοι MCF για την όσο το δυνατόν αποδοτικότερη αξιοποίηση του υποστρώματος.

Τέλος, όσον αφορά τις απαιτήσεις σε max_hops των εικονικών ζεύξεων πρέπει να υιοθετηθούν λογικές τιμές >1 που εξαρτώνται θεωρητικά από τις συνολικές απαιτήσεις του εικονικού δικτύου, αφού ένα μεγάλο σε μέγεθος εικονικό δίκτυο πρέπει να έχει πιο χαλαρές απαιτήσεις καθυστέρησης για να μπορεί να ενσωματωθεί αποτελεσματικά. Βέβαια, μικρές τιμές για τον περιορισμό max_hops μπορούν να οδηγήσουν σε καλύτερες απεικονίσεις όσον αφορά τα κόστη ενσωμάτωσης αλλά αν επιλεγούν πολύ περιοριστικές μπορούν να αυξήσουν το ποσοστό των απορριφθέντων αιτήσεων εικονικών δικτύων λόγω του προβλήματος μη ύπαρξης ισομορφισμών που αναφέρθηκε παραπάνω.

IV.4.2 *Βασικές συναρτήσεις αλγορίθμου*

Παρακάτω περιγράφονται τέσσερις βασικές συναρτήσεις του αλγορίθμου: η *genneigh()*, η *valid()*, η *create()* και η *undo_creation()*. Οι συναρτήσεις αυτές επεξηγούνται λεπτομερώς ως προς τη λειτουργία τους για διευκόλυνση κατανόησης του κώδικα που χρησιμοποιήθηκε και παρουσιάζεται στο παράρτημα στο τέλος της εργασίας.

IV.4.2.1 *genneigh()*

Η συνάρτηση αυτή χρησιμοποιεί τέσσερις γράφους ως ορίσματα εισόδου: ένα εικονικό δίκτυο (VN) G^V, το υπόστρωμα G^P, έναν υπογράφο G^V_{sub} του G^V και μία απεικόνιση (VNM) $M(G^V_{sub})$ του G^V_{sub} πάνω στο G^P η οποία μπορεί επίσης να ειδωθεί ως γράφος. Παρουσιάζεται στο σχήμα 9.

Στο πρώτο βήμα η συνάρτηση παράγει την προβολή $F\,G^V_{sub}(G^V)$ του G^V_{sub} ως προς το G^V και ελέγχει αν είναι κενή. Στο σημείο αυτό πρέπει να εξηγηθεί το σύνολο $F\,G^V_{sub}(G^V)$ και πώς αυτό παράγεται:

Έστω G=(N,L) ένας κατευθυνόμενος γράφος (αυτή η απαίτηση δεν είναι τελεσίδικη αλλά μπορεί να εφαρμοστεί και σε μη-κατευθυνόμενους γράφους όπως στην εργασία) και $G_{sub}=(N_{sub},L_{sub})$ ένας υπογράφος του G. Τότε το σύνολο:

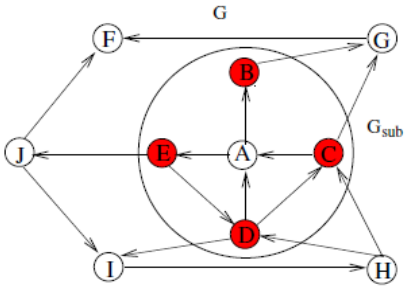
$F\,G^V_{sub}(G^V) = F\,G^{V\ in}_{sub}(G^V) \cup F\,G^{V\ out}_{sub}(G^V)$

όπου: $F\,G^{V\ in}_{sub}(G^V) = \{n_i | (n_i,n_j) \in L \wedge (n_j \in N_{sub} \wedge n_i \notin N_{sub})\}$

$F\,G^{V\ out}_{sub}(G^V) = \{n_j | (n_i,n_j) \in L \wedge (n_i \in N_{sub} \wedge n_j \notin N_{sub})\}$

καλείται η προβολή του G_{sub} σε σχέση με τον G. Δηλ. η προβολή αυτή είναι το σύνολο όλων των κόμβων που είναι προσκείμενοι σε μία ζεύξη που συνδέει τον G_{sub} με τον G.

Ένα παράδειγμα εύρεσης προβολής παρουσιάζεται παρακάτω:



Σχήμα 8: Παράδειγμα εύρεσης προβολής $F\,G^V_{sub}(G^V)$ του G^V_{sub} σε σχέση με το G^V (από [1])

Εδώ έχουμε: N={A,B,C,D,E,F,G,H,I,J}, N_{sub}={A,B,C,D,E}

$F\,G^{V\ in}_{sub}(G^V)$={C,D}

$F\,G^{V\ out}_{sub}(G^V)$={B,C,D,E}

➔$F\,G^V_{sub}(G^V)$={B,C,D,E}

Ίδιο αποτέλεσμα θα παίρναμε και στην περίπτωση full-duplex ζεύξεων.

Algorithm 2 genneigh(G^P, G^V, G^V_{sub}, $M(G^V_{sub})$)

Require: G^P, G^V, G^V_{sub}, $M(G^V_{sub})$

1: **if** $F_{G^V_{sub}}(G^V) = \emptyset$ **then**
2: $\quad C = N^V \times N^P$
3: **else**
4: $\quad C = F_{G^V_{sub}} \times G^P \setminus M_N(G^V_{sub})$
5: **end if**
6: optimize(C)
7: sort(C)
8: return(C)

Σχήμα 9: Η συνάρτηση genneigh() (από [1])

Ας θεωρηθεί ξανά το αρχικό παράδειγμα εφαρμογής του αλγορίθμου. Στο αρχικό βήμα του vnmFlib το G^V_{sub} είναι κενό και άρα το $F G^V_{sub}(G^V)$ είναι επίσης κενό. Συνεπώς ο αλγόριθμος αρχικά υπολογίζει το σύνολο C υποψηφίων απεικονίσεων ως το καρτεσιανό γινόμενο όλων των κόμβων στο G^V και όλων των κόμβων στο G^P. Όμως αυτό το σύνολο μειώνεται σημαντικά ως προς τις διαστάσεις του με χρήση της συνάρτησης *optimize()*. Η συνάρτηση αυτή διαγράφει όλα τα ζεύγη-απεικονίσεις κόμβων που δεν ικανοποιούν την απαίτηση CPU. Προαιρετικά, σε περίπτωση ύπαρξης πρόσθετων περιορισμών για τους εικονικούς κόμβους, η συνάρτηση μπορεί να διαγράψει ζεύγη τα οποία π.χ. δεν ικανοποιούν τον περιορισμό επιθυμητής φυσικής θέσης απεικόνισης του εικονικού κόμβου ή απαίτησης κάποιου άλλου πόρου όπως η μνήμη. Ανάλογες βελτιστοποιήσεις μπορούν να οδηγήσουν σε δραστική συρρίκνωση του χώρου αναζήτησης του αλγορίθμου και σε συντομότερους χρόνους λειτουργίας και απόκρισης ύπαρξης λύσης.

Προσοχή όμως πρέπει να δοθεί στο γεγονός ότι σε αυτό το στάδιο η βελτιστοποίηση μπορεί να εφαρμοστεί μόνο για τις απεικονίσεις κόμβων και όχι ζεύξεων. Στο [1] προτείνεται ο έλεγχος ύπαρξης φυσικών μονοπατιών με συγκεκριμένο μέγιστο μήκος (max_hops), όπου μελλοντικά θα απεικονιστούν οι εικονικές ζεύξεις που θα συνδέσουν τους υποψήφιους εικονικούς κόμβους με τους ήδη χαρτογραφημένους. Αυτός ο έλεγχος δεν έχει νόημα να γίνει εδώ αλλά σε μετέπειτα στάδιο (συνάρτηση *valid())* καθώς το λογικό είναι πρώτα να ελεγχθούν οι κόμβοι και να επιλεγεί κάποια απεικόνιση εικονικού κόμβου σε πρώτη φάση και κατόπιν να γίνει ο έλεγχος ύπαρξης μονοπατιών που θα συνδέουν αυτόν τον κόμβο με τους υπολοίπους σε φυσικό επίπεδο. Προσπάθεια προ-ελέγχου μονοπατιών για βελτιστοποίηση μπορεί να επιφέρει αρνητικά αποτελέσματα ως προς την ταχύτητα καθώς δαπανάται πολύς υπολογιστικός χρόνος για μεγάλο σύνολο πιθανών απεικονίσεων χωρίς ιδιαίτερο κέρδος. Στην εργασία όμως έχει γίνει μία στοιχειώδης βελτιστοποίηση όσον αφορά την επιλογή ζεύξεων εικονικών τερματικών – routers. Συγκεκριμένα, αν ένα υποψήφιο ζεύγος απεικόνισης τερματικών περιέχει ένα εικονικό τερματικό το οποίο συνδέεται με εικονικό router με ζεύξη με BW που υπερβαίνει το BW της ζεύξης του υποψήφιου φυσικού τερματικού και της default GW του, τότε το ζεύγος διαγράφεται αφού είναι βέβαιο ότι το εικονικό τερματικό δεν θα μπορέσει να συνδεθεί φυσικά με τους γείτονές του αν απεικονιστεί στο εν λόγω φυσικό τερματικό.

Μετά την τελική βελτιστοποίηση των ζευγών εικονικός-φυσικός κόμβος, τα ζεύγη ταξινομούνται. Το κριτήριο ταξινόμησης εξαρτάται από το σύνολο περιορισμών των δικτύων, και στην εργασία τέθηκε ως ένας σταθμισμένος μέσος όρος των πόρων των φυσικών κόμβων αφού όλοι αναχθούν σε κάποια κοινή μετρική βάση. Βαρύτητα δόθηκε στον πόρο που είναι σπανιότερος και ακριβότερος για κάθε κόμβο ➔π.χ. στα τερματικά η CPU. Διαισθητικά είναι καλύτερο να χρησιμοποιούμε για χαρτογράφηση φυσικούς κόμβους με υψηλή εναπομείνασα χωρητικότητα πρώτους, όπως επίσης είναι προτιμότερο σε αυτούς τους κόμβους να απεικονίζονται εικονικοί κόμβοι με υψηλότερες απαιτήσεις σε πόρους.

➔Η ταξινόμηση λοιπόν πραγματοποιείται ως εξής:

1. Ταξινομούνται σε πρώτη φάση τα ζεύγη απεικονίσεων τερματικών του C με κριτήριο τόσο το σταθμισμένο μέσο όρο CPU και slices που διαθέτουν τα φυσικά τερματικά όσο και τις απαιτήσεις CPU των εικονικών τερματικών. Ζεύγη που περιέχουν φυσικά τερματικά με περισσότερους πόρους και εικονικά τερματικά με μεγαλύτερες απαιτήσεις τοποθετούνται πρώτα.

Η ταξινόμηση λαμβάνει χώρα σε δύο στάδια: στο $1^{ο}$ με βάση τους φυσικούς πόρους και στο $2^{ο}$ με βάση τις εικονικές απαιτήσεις. Τα στάδια είναι ακολουθιακά ($1^{ο}$→$2^{ο}$). Έτσι προτιμώνται ζεύγη που ικανοποιούν και τις δύο απαιτήσεις (εικονικό και φυσικό τμήμα).

2. Σε δεύτερη φάση ταξινομούνται τα ζεύγη δρομολογητών του C με κριτήριο τον αριθμό των slices που διαθέτουν οι φυσικοί τους κόμβοι. Όσο περισσότερα slices➔ σε τόσο υψηλότερη θέση το αντίστοιχο ζεύγος.

Για την ταξινόμηση μπορούν να χρησιμοποιηθούν τεχνικές μονής μετρικής-κριτηρίου ταξινόμησης όπως η bubblesort, η quicksort κτλ. Στην εργασία χρησιμοποιήθηκε η απλή στην υλοποίηση bubblesort.

IV.4.2.2 *valid()*

IV.4.2.2.1 *Βασική λειτουργία συνάρτησης*

Η συνάρτηση αυτή ελέγχει το αν η προσθήκη ενός ζεύγους αντιστοίχισης n^V-n^P σε μία έγκυρη VNM (Virtual Network Mapping) καταλήγει πάλι σε μία έγκυρη VNM.
Συγκεκριμένα, ελέγχει αν ικανοποιούνται οι ακόλουθοι κανόνες:

➔Ικανοποίηση απαιτήσεων κόμβου (έλεγχος άκυρων απεικονίσεων κόμβων):

- Ικανοποίηση διαθεσιμότητας απεικόνισης: n^P assigned?
- Ικανοποίηση απαίτησης ίδιου τύπου: n^P→type==n^V→type?
- Ικανοποίηση απαίτησης CPU αν n^P ,n^V τερματικά:
  n^P→residual_CPU>n^V→required_CPU?
- Ικανοποίηση απαίτησης αριθμού slices: n^P→residual_slices_number>0?
- (Προαιρετικά) Ικανοποίηση απαίτησης επιθυμητής θέσης κόμβου απεικόνισης ή λιγότερο αυστηρής απαίτησης μέγιστης απόστασης φυσικού κόμβου απεικόνισης:
  n^V→required_n^P==n^P? or distance(n^V→required_n^P,n^P)<maximum?
- (Προαιρετικά) Ικανοποίηση επιθυμητής θέσης GW(GateWay)
  αν n^V τερματικό: n^V→required_GW==n^P→GW?

➔Ικανοποίηση απαιτήσεων ζεύξης (έλεγχος «σπασμένων» συνδέσεων):

- Για κάθε γείτονα virtual_neighbour του n^V που ανήκει στον G_{sub}^V υπάρχει φυσικό μονοπάτι μήκους<max_hops από τον n^P στον φυσικό κόμβο physical_host που ανήκει στον M(G_{sub}^V) και όπου έχει απεικονιστεί ο virtual_neighbour
- Το αντίστοιχο μονοπάτι πρέπει να ικανοποιεί την απαίτηση BW κάθε εικονικής ζεύξης (n^V-virtual_neighbour), δηλ. θα πρέπει το BW της bottleneck φυσικής ζεύξης που αυτό περιλαμβάνει να είναι μεγαλύτερο ή ίσο του απαιτούμενου BW της εικονικής ζεύξης.

Αν έστω και ένας εκ των παραπάνω κανόνων δεν ισχύει τότε η *valid()* σταματά και επιστρέφει NULL, καθώς μία και μόνο παραβίαση των περιορισμών κόμβου ή ζεύξης επαρκεί για την παραγωγή μίας άκυρης χαρτογράφησης. Σε αντίθετη περίπτωση, δηλ. αν όλοι οι κανόνες ικανοποιούνται, η *valid()* επιστρέφει μία δομή που περιλαμβάνει όλες τις αντιστοιχίσεις των εικονικών ζεύξεων του n^V με τους γείτονές του εντός του G_{sub}^{V} με τα φυσικά μονοπάτια που υπολογίστηκαν ως έγκυρα τόσο από πλευράς μήκους (hops) όσο και από πλευράς BW.

Η εύρεση μονοπατιών που ικανοποιούν και τα δύο constraints (BW+delay) είναι ένα σημαντικό τμήμα της εργασίας καθώς αποτελεί μία από τις βασικότερες λειτουργίες που παρασκηνιακά επιτελεί ο αλγόριθμος. Για το λόγο αυτό περιγράφεται ανεξάρτητα παρακάτω.

IV.4.2.2.2 *Υπολογισμός μονοπατιών διπλής μετρικής (BW+delay)*

Χρησιμοποιήθηκε ένας αλγόριθμος ο οποίος στηρίζεται στο [18] και στον αλγόριθμο του Dijkstra εύρεσης συντομότερου μονοπατιού. Ο αλγόριθμος αυτός είναι συγκεντρωτικός και όχι κατανεμημένος (βέβαια στο [18] περιλαμβάνεται και ένας αντίστοιχος κατανεμημένος) και μπορεί να χρησιμοποιηθεί για δρομολόγηση πηγής με το μονοπάτι να υπολογίζεται κατ' απαίτηση στον κόμβο πηγής κεντρικά. Δοθέντων των απαιτήσεων BW και delay μίας ροής, ο αλγόριθμος μπορεί να βρει ένα μονοπάτι που ικανοποιεί και τις δύο προϋποθέσεις σε περίπτωση που αυτό υπάρχει.

Έστω ένας μη-κατευθυντικός γράφος G=(N,L) με N το σύνολο των κόμβων και L το σύνολο των ζεύξεων που περιλαμβάνει. Σε κάθε $(i,j) \in L$, με $i,j \in N$ ανατίθενται δύο πραγματικοί αριθμοί, b_{ij} ως το διαθέσιμο εύρος ζώνης και d_{ij} ως η καθυστέρηση. Αν ως καθυστέρηση νοείται ο αριθμός των αλμάτων-hops, τότε $d_{ij}=1$. Αν $(i,j) \notin L \rightarrow b_{ij}=0$, $d_{ij}=\infty$.

Δοθέντος λοιπόν ενός μονοπατιού p=(i,j,k,…,,m) προκύπτουν οι ακόλουθοι ορισμοί:

- το εύρος του μονοπατιού = εύρος(p) ορίζεται ως το bottleneck BW του μονοπατιού, δηλ.: εύρος(p)=$\min\{b_{ns}\}$, $\forall n,s \in p \wedge n,s$ ακολουθιακά στο p
- το μήκος του μονοπατιού = μήκος(p) ορίζεται ως η συνολική αθροιστική καθυστέρηση διάσχισης του μονοπατιού, δηλ. : μήκος(p)=$d_{ij}+\ldots+d_{lm}$

Δοθέντων δύο κόμβων i και m του γράφου, και δύο περιορισμών B και D, το πρόβλημα είναι ή εύρεση ενός μονοπατιού p^* που ενώνει τα i,m και για το οποίο ισχύουν: εύρος(p^*)≥B και μήκος(p^*)≤D. Ένα μονοπάτι έχει εύρος όχι μικρότερο από B αν και μόνο αν κάθε ζεύξη του μονοπατιού έχει BW όχι μικρότερο από B. Αυτό συνεπάγεται ότι κάθε ζεύξη με BW<B αποκλείεται να είναι μέρος του μονοπατιού που επιδιώκεται.

Γι' αυτό η εύρεση λαμβάνει χώρα σε δύο στάδια:

- Πρώτον, διαγράφονται όλες οι ζεύξεις με BW<B ώστε για κάθε μονοπάτι p στον εναπομείναντα γράφο να ισχύει: εύρος(p)≥B.
- Δεύτερον, γίνεται προσπάθεια εύρεσης ενός μονοπατιού p που ικανοποιεί: μήκος(p)≤D.

Για να επιτευχθεί αυτό, μπορεί να ευρεθεί μονοπάτι ελαχίστου μήκους. Σε μία αναζήτηση-πέρασμα μπορεί να αποφασιστεί αν ένα τέτοιο μονοπάτι υπάρχει και να ευρεθεί ένα σε θετική περίπτωση.
Έστω ότι το μονοπάτι p^* είναι το μονοπάτι ελαχίστου μήκους D_{min} . Αν $D_{min} \leq D$ τότε το p^* ικανοποιεί κα τους δύο περιορισμούς. Αλλιώς, γίνεται κατανοητό ότι δεν υπάρχει κατάλληλο μονοπάτι αφού οποιοδήποτε άλλο και να επιλεγεί θα έχει μήκος >D αναγκαστικά.

Έστω ότι ο κόμβος s είναι η πηγή και ο κόμβος d ο προορισμός. Ο ακόλουθος αλγόριθμος βρίσκει (ή προσπαθεί να βρει) ένα μονοπάτι μεταξύ s,d με εύρος ≥ B και μήκος ≤ D, εφόσον ένα τέτοιο μονοπάτι υπάρχει. Έστω D_i το εκτιμώμενο μήκος του συντομότερου μονοπατιού από τον κόμβο s στον κόμβο i που ικανοποιεί επιπλέον και τον περιορισμό BW. Τα μονοπάτια που τυχόν ευρίσκονται είναι βέλτιστα ως προς το μήκος και best-effort όσον αφορά την αποδοτική διαχείριση του BW. Τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Για κάθε $i,j \in N$:
Θέσε $d_{ij}=1$, αν $(i,j) \in L$ και $b_{ij}>B$
Θέσε $d_{ij}=\infty$, αν $(i,j) \notin L$ ή $b_{ij}<B$

Βήμα 2: Θέσε $C=\{s\}$, $D_i=d_{si}$ για κάθε $i \neq s$

Βήμα 3: Βρες $k \notin C$ έτσι ώστε $D_k= \min_{i \notin C} D_i$
$C=C \cup \{k\}$.
Αν $D_k>D$, δεν υπάρχει μονοπάτι και ο αλγόριθμος τερματίζει.
Αν το σύνολο C περιέχει τον κόμβο d, ένα μονοπάτι βρέθηκε και ο αλγόριθμος τερματίζει.

Βήμα 4: Για κάθε $i \notin C$, θέσε $D_i=\min[D_i,D_k+d_{ki}]$

Βήμα 5: Πήγαινε στο βήμα 3.

Το βήμα 1 εξαλείφει όλες τις ζεύξεις που δεν ικανοποιούν την απαίτηση BW θέτοντας το μήκος τους ίσο με ∞. Τα βήματα 2-5 βρίσκουν το μονοπάτι ελάχιστου μήκους με χρήση του αλγορίθμου του Dijkstra. Προσοχή: δεν χρειάζεται να ευρεθούν τα μονοπάτια ελαχίστου μήκους από την πηγή σε όλους τους κόμβους. Ο αλγόριθμος μπορεί να τερματίσει είτε όταν ο κόμβος-προορισμός d είναι μόνιμα μαρκαρισμένος ή όταν η καθυστέρηση σε hops υπερβεί το κατώφλι πριν ενταχτεί ο d στο σύνολο C μαρκαρισμένων κόμβων.

Ο αλγόριθμος είναι ελεύθερος από βρόχους-loops καθώς τα μονοπάτια που επιλέγονται είναι τα συντομότερα του ελαττωμένου γράφου. Κάθε βήμα στον παραπάνω αλγόριθμο απαιτεί έναν αριθμό λειτουργιών ανάλογο του |N|, όπου |N| = αριθμός κόμβων του υποστρώματος. Τα βήματα επαναλαμβάνονται στη χειρότερη περίπτωση |N|-1 φορές. Συνεπώς η πολυπλοκότητα του αλγορίθμου είναι στη χειρότερη περίπτωση $O(|N|^2)$, ίδια με του αλγορίθμου του Dijkstra.

IV.4.2.3 *create()*

Η συνάρτηση αυτή καλείται αφού το υποψήφιο ζεύγος εικονικός-φυσικός κόμβος (n^V-n^P) περάσει τον έλεγχο της valid. Λόγω των υπολογισμών που γίνονται στην valid είναι διαθέσιμα όλα τα φυσικά μονοπάτια που θα συνδέσουν τον φυσικό host του εικονικού κόμβου με όλους τους φυσικούς hosts όπου έχουν απεικονιστεί οι εικονικοί γείτονές του που ανήκουν στον υπογράφο G_{sub}^V.

Συνεπώς πραγματοποιούνται τα ακόλουθα βήματα:

- Προστίθεται στον υπογράφο G_{sub}^V ο εικονικός κόμβος n^V του ορθά επιλεγμένου ζεύγους.
- Προστίθενται στον υπογράφο G_{sub}^V όλες οι εικονικές ζεύξεις μεταξύ του εικονικού κόμβου n^V και των γειτόνων του εντός του G_{sub}^V.
- Απεικονίζεται ο εικονικός κόμβος n^V στον φυσικό host με ενημέρωση του $M(G_{sub}^V)$. Με την απεικόνιση δεσμεύονται οι αναγκαίοι πόροι (CPU κτλ.) από τον host και παραχωρείται ένα slice αυτού για τη λειτουργία του εικονικού δικτύου, αν αυτή η παραχώρηση δεν έχει ήδη γίνει σε προηγούμενο στάδιο.
- Απεικονίζονται οι εικονικές ζεύξεις μεταξύ του εικονικού κόμβου και των γειτόνων του εντός του GsubV στα ανάλογα φυσικά μονοπάτια με ενημέρωση του $M(G_{sub}^V)$. Σε κάθε εικονική ζεύξη που περιλαμβάνεται σε κάποιο εκ των φυσικών μονοπατιών γίνεται δέσμευση του αναγκαίου BW για τη λειτουργία της αντίστοιχης εικονικής ζεύξης. Επίσης κάθε ενδιάμεσος (κα όχι ακραίος) κόμβος κάθε φυσικού μονοπατιού παραχωρεί ένα slice αυτού για τη λειτουργία του εικονικού δικτύου, αν αυτή η παραχώρηση δεν έχει ήδη γίνει σε προηγούμενο στάδιο. Αυτή η παραχώρηση είναι απαραίτητη καθώς ναι μεν οι ενδιάμεσοι κόμβοι μπορεί να μην έχουν παραχωρηθεί για απεικόνιση, αλλά είναι απαραίτητοι για τη δρομολόγηση εντός του υποστρώματος και τη διαχείριση των ζεύξεων του εικονικού δικτύου. Οι ακραίοι κόμβοι με την παραπάνω διαδικασία έχουν ήδη παραχωρήσει κάποιο slice για απεικόνιση ή ενδιάμεση δρομολόγηση.

➔Η εκτέλεση των βημάτων που περιγράφονται παραπάνω οδηγούν στην σταδιακή εξέλιξη του G_{sub}^V (προσθήκη εικονικών κόμβων και ακμών) μέχρι την επίτευξη του ισομορφισμού του με τον G^V, σε περίπτωση που αυτός είναι υπαρκτός. Επίσης παράλληλα ενημερώνεται και η δομή χαρτογράφησης $M(G_{sub}^V)$ με τις νέες φυσικές απεικονίσεις.

IV.4.2.4 *undo_creation()*

Η συνάρτηση αυτή καλείται σε περίπτωση αντιμετώπισης κάποιο αδιεξόδου κατά τη διάρκεια της χαρτογράφησης. Με τον όρο αδιέξοδο εννοείται η αδυναμία μετέπειτα εξέλιξης του αλγορίθμου vnmFlib καθώς δεν υπάρχουν πλέον διαθέσιμα υποψήφια ζεύγη απεικονίσεων που ικανοποιούν τις προϋποθέσεις έτσι ώστε να προχωρήσει η ενημέρωση του G_{sub}^V και της $M(G_{sub}^V)$.

Άρα ο vnmFlib τελματώνει και δημιουργείται πρόβλημα συνέχισης της διαδικασίας.

Το αδιέξοδο στη συνήθη περίπτωση είναι αποτέλεσμα κάποιας κακής χαρτογραφικής απόφασης που λήφθηκε σε προηγούμενο βήμα και οδηγεί σε τέλμα.

Για να επιλυθεί λοιπόν το πρόβλημα, γίνεται υπαναχώρηση (backtracking) στην τελευταία έγκυρη χαρτογράφηση, αναιρώντας τη λειτουργία *create()* που έλαβε χώρα για την τελικά «άκυρη» προηγούμενη απεικόνιση n^V-n^P. Αυτή η υπαναχώρηση μπορεί να γίνει αναδρομικά αν συναντηθούν επιπλέον αδιέξοδα, καταλήγοντας πάντα όμως είτε σε έγκυρες χαρτογραφήσεις είτε σε τερματισμό του αλγορίθμου λόγω υπέρβασης του μέγιστου ορίου διαδοχικών κλήσεων.

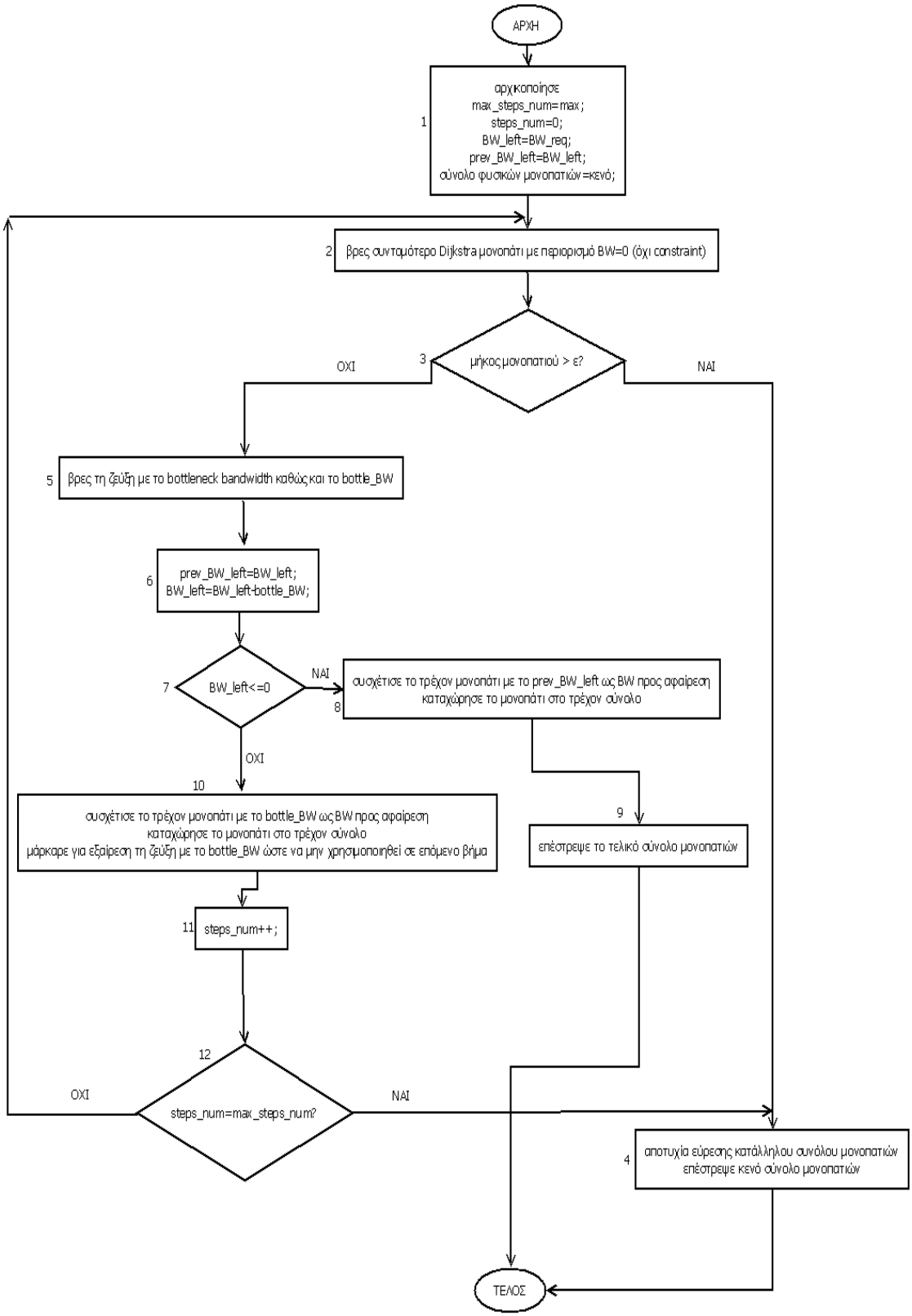
➔Η συνάρτηση *undo_creation()* εξασφαλίζει την ομαλή αποκατάσταση του χαρτογραφικού λάθους αναιρώντας πλήρως τα αποτελέσματα της αμέσως προηγούμενης κλήσης της συνάρτησης *create()* και επιτελώντας τις ακριβώς αντίθετες διαδικασίες σε σχέση με αυτήν. Δηλ. γίνονται τα ακριβώς ανάποδα βήματα με διαδοχική αποδέσμευση όλων των δεσμευθέντων πόρων φυσικών κόμβων και ζεύξεων (CPU, slices, BW) και αντίστοιχης κατάργησης των πραγματοποιηθεισών απεικονίσεων από τη δομή M(G^V_{sub}) και των προσθηκών κόμβων και ζεύξεων από τον G^V_{sub}.

IV.5 *Προτεινόμενοι θεωρητικοί αλγόριθμοι*

Με στόχο την ενθάρρυνση μελλοντικής έρευνας και την επέκταση του βασικού αλγορίθμου ενσωμάτωσης ώστε να αντιμετωπίζονται και περιπτώσεις απεικονίσεων ζεύξεων multi-path (πολλαπλών μονοπατιών-ροών κατ’ αναλογία με MCF αλγορίθμους) καθώς και περιπτώσεις φυσικών αστοχιών σε κόμβους ή ζεύξεις του υποστρώματος προτείνονται οι ακόλουθοι αλγόριθμοι οι οποίοι παρουσιάζονται σε μορφή διαγραμμάτων ροής.

IV.5.1 *Προτεινόμενος* αλγόριθμος *για εύρεση πολλαπλών φυσικών μονοπατιών για μία εικονική ζεύξη*

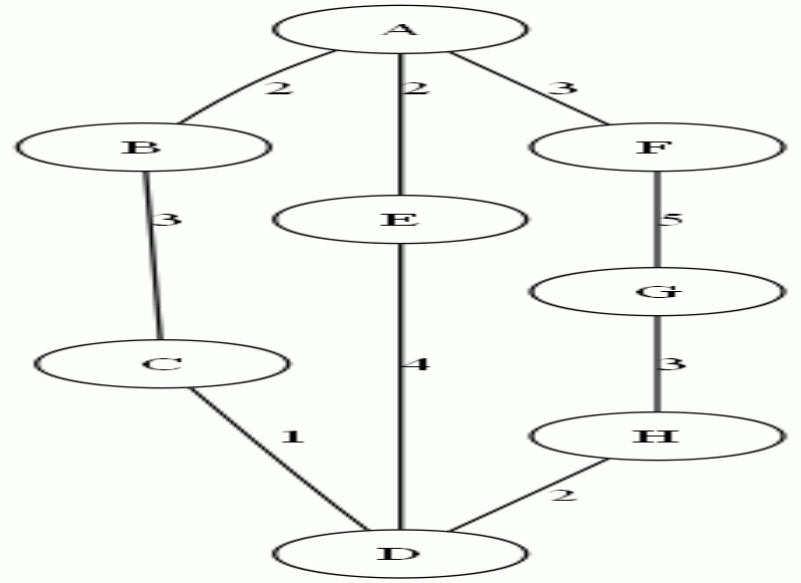
IV.5.1.1 *Διάγραμμα ροής αλγορίθμου εύρεσης πολλαπλών μονοπατιών*



Σχήμα 10: Διάγραμμα ροής αλγορίθμου εύρεσης πολλαπλών μονοπατιών

IV.5.1.2 *Παράδειγμα εφαρμογής αλγορίθμου εύρεσης πολλαπλών μονοπατιών*

Έστω εικονική ζεύξη VL με απαίτηση BW=5 και max_hops=4, με άκρα απεικονισθέντα στους φυσικούς κόμβους A και D, ενώ δίνεται ο παρακάτω αρχικός φυσικός γράφος-υπόστρωμα όπου φαίνονται τα BW των φυσικών ζεύξεων:



Σχήμα 11: Αρχικός φυσικός γράφος-υπόστρωμα για το παράδειγμα εφαρμογής αλγορίθμου εύρεσης πολλαπλών μονοπατιών

Εφαρμογή αλγορίθμου

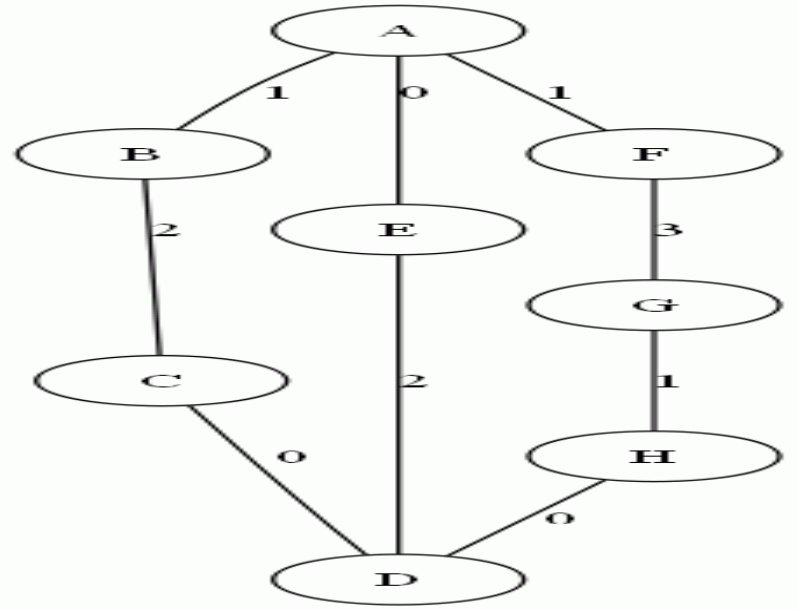
Αρχικοποίηση
max_steps_num=3;
steps_num=0;
BW_left=5;
Prev_BW_left=5;
Path_set={};

Βήμα 1
Βρέθηκε συντομότερο μονοπάτι A-E-D.
hops=2<max_hops➔OK
ζεύξη με bottleneck BW=A-E, bottle_BW=2
prev_BW_left=5
BW_left=3>0
Path_set={(A-E-D)➔BW=2}
A-E μαρκάρεται για αφαίρεση ➔A-E-D όχι πλέον διαθέσιμο

Βήμα 2
Βρέθηκε συντομότερο μονοπάτι A-B-C-D.
hops=3<max_hops➔OK
ζεύξη με bottleneck BW=C-D, bottle_BW=1
prev_BW_left=3
BW_left=2>0
Path_set={(A-E-D)➔BW=2, (A-B-C-D)➔BW=1}
C-D μαρκάρεται για αφαίρεση ➔ A-B-C-D όχι πλέον διαθέσιμο

Βήμα 3
Βρέθηκε συντομότερο μονοπάτι A-F-G-H-D.
hops=4=max_hops➔OK
ζεύξη με bottleneck BW=H-D, bottle_BW=2
prev_BW_left=2
BW_left=0
Path_set={(A-E-D)➔BW=2, (A-B-C-D)➔BW=1, (A-F-G-H-D)➔BW=2}
return Path_set

Ο τελικός φυσικός γράφος παρουσιάζεται παρακάτω:



Σχήμα 12: Τελικός φυσικός γράφος-υπόστρωμα μετά την απεικόνιση της εικονικής ζεύξης για το παράδειγμα εφαρμογής αλγορίθμου εύρεσης πολλαπλών μονοπατιών

Παρατηρείται ότι ενώ στην περίπτωση single-path η απεικόνιση θα ήταν αδύνατη, τώρα η ζεύξη απλώθηκε σε τρία φυσικά μονοπάτια καταλαμβάνοντας τους πόρους των φυσικών ζεύξεων ανάλογα με το bottle-neck κάθε μονοπατιού . Προσοχή όμως στα max_hops: αν είχε απαιτηθεί max_hops=3 (μεγαλύτερη αυστηρότητα) ➔ο αλγόριθμος θα επέστρεφε κενό σύνολο (αδυναμία διαμοιρασμού BW).

Βέβαια ο αλγόριθμος δε επιδρά πάντα θετικά, καθώς θα μπορούσαν τα παραπάνω αξιοποιηθέντα από μία μόνο εικονική ζεύξη μονοπάτια να καταληφθούν από τρεις εικονικές ζεύξεις τριών διαφορετικών εικονικών δικτύων που έχουν τα άκρα τους στους A,D και BW=1, κάτι που τώρα δεν μπορεί να συμβεί. Άρα, διαισθητικά καταλαβαίνουμε ότι ο αλγόριθμος αυτός ευνοεί την single-VN απεικόνιση για πολύ απαιτητικά σε BW VN και ενδείκνυται για εφαρμογή σε περιβάλλοντα πολλαπλών VN κυρίως σε ακραίες περιπτώσεις όπου το υπόστρωμα βρίσκεται στα όριά του και πρέπει να κάνει ό,τι καλύτερο μπορεί για να δεχθεί μία νέα απαιτητική αίτηση, αυξάνοντας το ποσοστό αποδεκτότητάς του καθώς σε διαφορετική περίπτωση (single-path) δεν θα μπορούσε να ικανοποιήσει «βαριές» αιτήσεις.

Φυσικά η τελική συμπεριφορά του αλγορίθμου και το αν αυτός ωφελεί ή επιβαρύνει το προσομοιωθέν σύστημα μπορεί να αποτελέσει αντικείμενο μελλοντικής έρευνας, σε συνδυασμό με τη χρήση αποδοτικότερων τεχνικών κατανομής εύρους ζώνης σε πολλαπλά μονοπάτια με εφαρμογή λογικών MCF δικτύων μεταφοράς.

IV.5.2 *Αλγόριθμοι ανάκαμψης από αστοχίες φυσικού κόμβου ή φυσικής ζεύξης*

IV.5.2.1 *Πώς το backtracking ευνοεί την ανάκαμψη από αστοχίες φυσικού κόμβου ή φυσικής ζεύξης*

Ο αλγόριθμος vnmFlib προσφέρει με μικρές τροποποιήσεις τη δυνατότητα ανάκαμψης από φυσική αστοχία κόμβου ή ζεύξης με επιμέρους διαφορετική χαρτογράφηση, καθώς μπορεί να εφαρμοστεί εύκολα στον τροποποιημένο (π.χ. μετά την πτώση κόμβου ή ζεύξης) εικονικό υπογράφο χρησιμοποιώντας για backtracking απεικονίσεις κοντινές στο σημείο αστοχίας, ώστε η διαταραχή να περιοριστεί στη γειτονιά της αστοχίας. Αν η αναπροσαρμογή τελικά αποτύχει μπορεί να γίνει γρήγορα εξ' ολοκλήρου καινούρια χαρτογράφηση, αν φυσικά το υπόστρωμα μπορεί να την υποστηρίξει. Δύο αλγόριθμοι που αντιστοιχούν σε πτώση κόμβου και ζεύξης παρουσιάζονται παρακάτω σε μορφή διαγραμμάτων ροής. Δεν δίνονται επιπλέον παραδείγματα καθώς τα διαγράμματα είναι σαφή και δεν αλλάζει κάτι σημαντικά σε σχέση με τον κύριο αλγόριθμο που παρουσιάστηκε εκτενώς παραπάνω, απλώς αντιμετωπίζεται ένα θεωρητικό πρόβλημα αστοχίας.
Δηλ. παρουσιάζονται διαγραμματικά τρόποι με τους οποίους θα μπορούσε το υλοποιηθέν σύστημα-αλγόριθμος να αντιμετωπίσει κάποια αστοχία με σεβασμό της αρχής της απομόνωσης των εικονικών δικτύων και της εξασφάλισης της ποιότητας υπηρεσίας. Και οι δύο επιπλέον αλγόριθμοι μπορούν να εφαρμοστούν ανεξάρτητα για κάθε επηρεασθέν εικονικό δίκτυο.
Βέβαια, για πληρότητα της αναφοράς πρέπει να διευκρινιστεί τι εννοείται με τον όρο αστοχία καθορίζοντας κάποιες από τις περιπτώσεις εκδήλωσής της.

Όσον αφορά τον φυσικό κόμβο, μπορεί να έχει συμβεί κάτι από τα εξής και να παρουσιάζεται εκτός λειτουργίας για κάποιο/α εκ των εικονικών δικτύων που σχετίζονται με αυτόν (ή για όλα αν η βλάβη είναι σοβαρή):

- Απενεργοποίηση κόμβου για λόγους συντήρησης
- Απότομη διακοπή παροχής ενέργειας προς κόμβο➔κόμβος "down"
- Σφάλμα σύνδεσης του κόμβου με το υπόλοιπο δίκτυο➔αποκοπή κόμβου
- Μόλυνση κόμβου με ιό και αυτόματη απομόνωσή του για αποφυγή εξάπλωσης κακόβουλου λογισμικού σε ολόκληρο το εικονικό δίκτυο και το υπόστρωμα
- Αστοχία στην διαχείριση κάποιων slices του κόμβου τα οποία καθίστανται πλέον μη διαθέσιμα για τα εικονικά δίκτυα που τον χρησιμοποιούσαν.

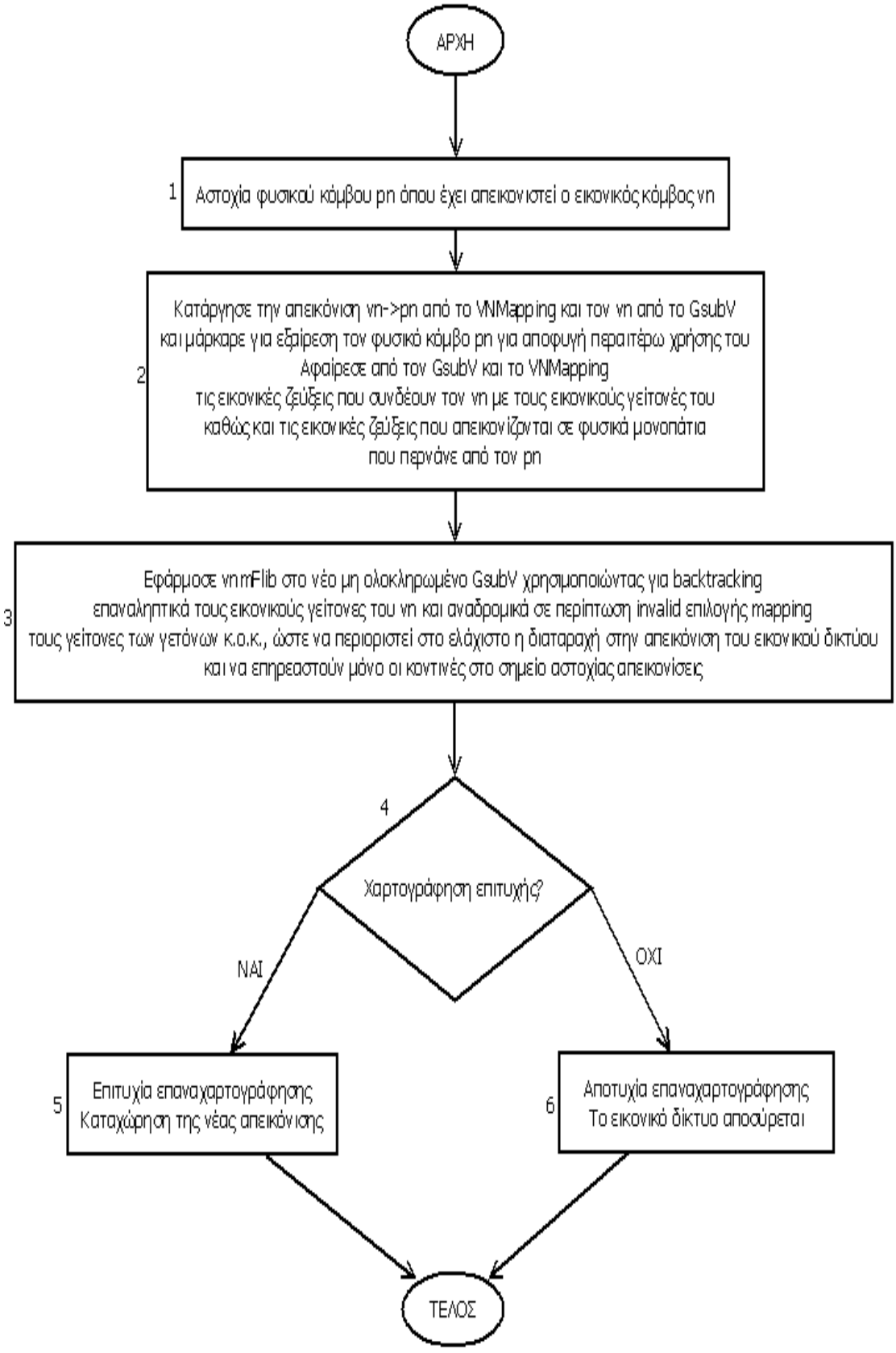
Όσον αφορά τη φυσική ζεύξη, μπορεί να έχει συμβεί κάτι από τα εξής (επηρεάζεται κάθε εικονική ζεύξη που αντιστοιχεί σε φυσικό μονοπάτι που διέρχεται από αυτήν):

- Διακοπή ζεύξης για αναβάθμιση-αντικατάσταση
- Φυσική βλάβη σε περίπτωση ενσύρματης ζεύξης όπως κοπή οπτικής ίνας ή καλωδίου χαλκού
- Σε περίπτωση ασύρματης ζεύξης➔πτώση ζεύξης (outage) λόγω εξασθένησης ισχύος, ύπαρξης παρεμβολών, ενισχυμένου θορύβου, μετακίνησης άκρων ζεύξης κτλ.

➔Υποθέτοντας λοιπόν ότι έχει συμβεί κάτι από τα παραπάνω που βγάζει εκτός κάποιον κόμβο ή ζεύξη του υποστρώματος, μπορούν να χρησιμοποιηθούν οι ακόλουθοι αλγόριθμοι για μία γρήγορη και όσο το δυνατόν λιγότερο επώδυνη ανάκαμψη από τη βλάβη.

IV.5.2.2 *Προτεινόμενος αλγόριθμος για ανάκαμψη από αστοχία φυσικού κόμβου*

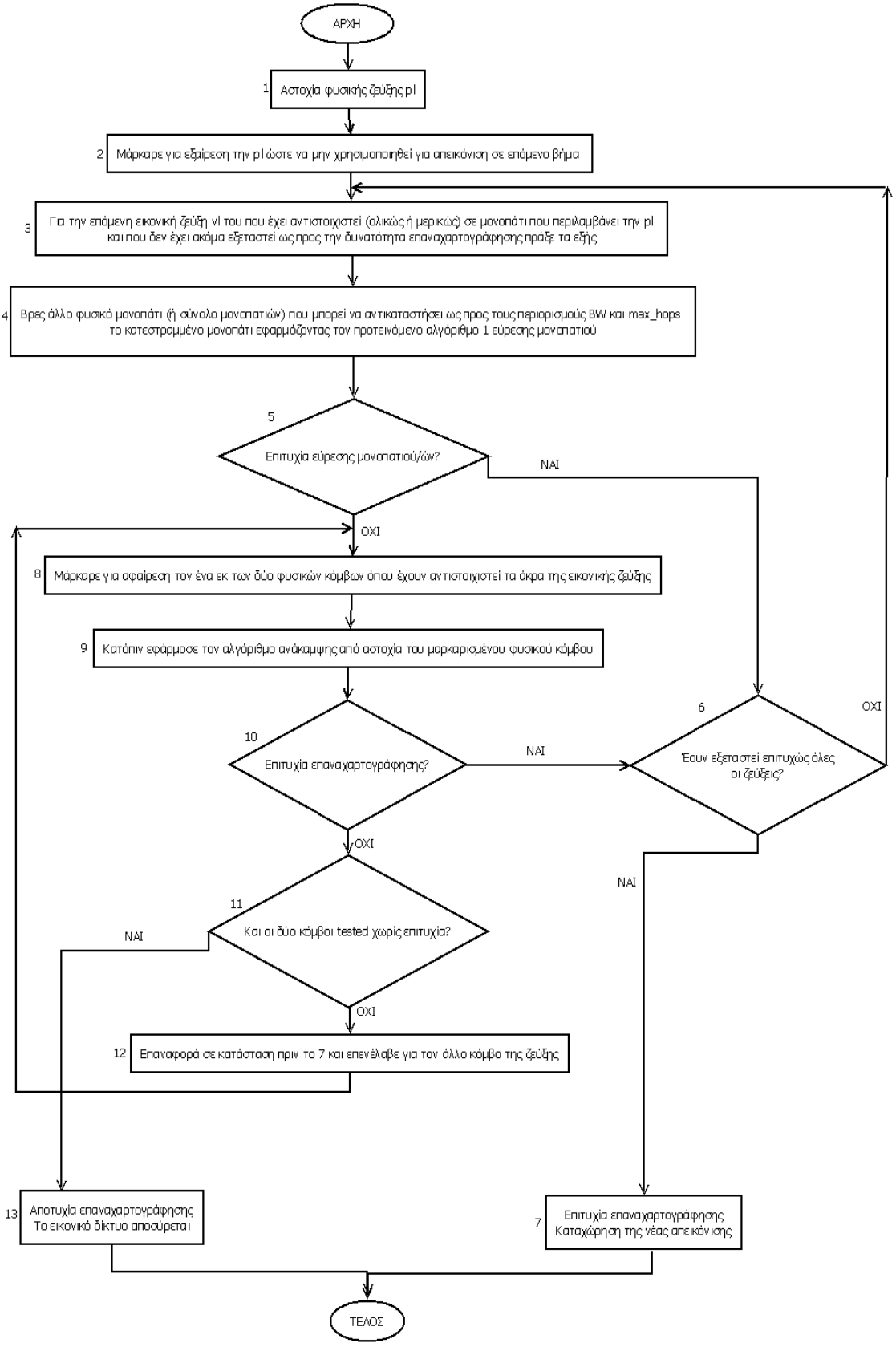
Ανάκαμψη από αστοχία φυσικού κόμβου



Σχήμα 13: Διάγραμμα ροής αλγορίθμου ανάκαμψης από αστοχία φυσικού κόμβου

IV.5.2.2.2 *Προτεινόμενος αλγόριθμος για ανάκαμψη από αστοχία φυσικής ζεύξης*

Ανάκαμψη από αστοχία φυσικής ζεύξης



Σχήμα 14: Διάγραμμα ροής αλγορίθμου ανάκαμψης από αστοχία φυσικής ζεύξης

V. Πειραματικά αποτελέσματα και σχόλια

V.1 *Βασικά στοιχεία και περιβάλλον προσομοίωσης*

Παρακάτω παρατίθενται τα βασικά στοιχεία που αφορούν την διεξαχθείσα προσομοίωση και το περιβάλλον όπου υλοποιήθηκε.

V.1.1 *Υπολογισμός ποιότητας χαρτογράφησης ανά αίτηση VNR*

Τα δίκτυα που χρησιμοποιούνται για την πειραματική αξιολόγηση είναι συσχετισμένα με τέσσερα κύρια κόστη όπως αναφέρθηκε και στο μέρος III.1.3:

- Εύρος ζώνης / ζεύξη : $\text{cost}_1(\text{M}(\text{G}^\text{V})) = \sum_{l \in LV} C_1^V(l) * length(M(l))$
- Υπολογιστική ισχύς / τερματικό : $\text{cost}_2(\text{M}(\text{G}^\text{V})) = \sum_{\substack{n \in NV, \\ n->term}} C_2^V(n)$
- Αριθμός slices / τερματικό : $\text{cost}_3(\text{M}(\text{G}^\text{V})) = \sum_{\substack{n \in NV, \\ n->term}} C_3^V(n)$
- Αριθμός slices / router : $\text{cost}_4(\text{M}(\text{G}^\text{V})) = \sum_{\substack{n \in NV, \\ n->router}} C_4^V(n) + \sum_{\substack{n \in NP, \\ n->router \\ transit}} C_4^P(n)$

Έτσι το συνολικό κόστος μίας απεικόνισης ορίζεται ως:

$\text{cost}(\text{M}(\text{G}^\text{V})) = \sum_{i=1}^{n} a_i * \cos t_i(M(G^V))$, με α_i=1, i=1,..,n, n=4

Αφού πάντα καταλαμβάνεται ανά επιτυχή απεικόνιση ένα slice ενός φυσικού host για κάθε εικονικό κόμβο (router/terminal) και για τους ενδιάμεσους routers, ισχύει:
$C_3^V(n)$=1, n→virtual terminal, $C_4^V(n)$=1, n→virtual router, $C_4^P(n)$=1, n→ physical router in transit. Επίσης: $C_1^V(l)$=BW που απαιτεί η εικονική ζεύξη l

$C_2^V(n)$=CPU που απαιτεί το εικονικό τερματικό n.

Για να εκφράσουμε την ποιότητα μίας απεικόνισης εισάγεται ο όρος «αναλογία κέρδους προς κόστος» (Revenue-to-Cost-Ratio⇔R/C), ως ο λόγος του κέρδους του παρόχου υπηρεσίας προς το κόστος που πληρώνει στον πάροχο υποδομής.

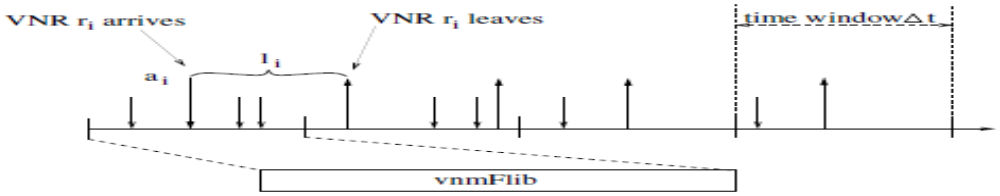
Το κέρδος του παρόχου ανά απεικόνιση εκφράζεται ως εξής, θεωρώντας ίδιους συντελεστές α_i όπως στο κόστος: $\text{R}(\text{G}^\text{V}) = \alpha_1 * \sum_{l \in LV} C_1^V(l) + \sum_{i=2}^{n} a_i * \cos t_i(M(G^V))$

Αυτό δείχνει ότι ο πάροχος επωφελείται από απεικονίσεις ζεύξεων σε όσο το δυνατόν συντομότερα μονοπάτια, δηλ. σε μονοπάτια που καταλαμβάνουν ελάχιστες σε αριθμό φυσικές ζεύξεις(➔νοικιάζει λιγότερο αθροιστικά BW από τον πάροχο υποδομής).
Προσοχή: για να μετρηθεί η επίδραση των μακρών μονοπατιών μόνο ως προς τη σπατάλη BW (όπως στο [1]) και όχι ως προς τη σχετικά μικρή σπατάλη slices των ενδιάμεσων routers, τα κόστη αυτών ενσωματώθηκαν ως είχαν στο revenue. Ούτως ή άλλως οι ενδιάμεσοι φυσικοί routers διαθέτουν στην προσομοίωση πολύ μεγάλο αριθμό slices και άρα δεν επηρεάζουν ιδιαίτερα τον πάροχο υπηρεσίας οικονομικά. Αυτό που πραγματικά μετρά στη μείωση του revenue είναι η σπατάλη του BW.

➔Εάν ένα εικονικό δίκτυο δεν μπορεί να απεικονιστεί θέτουμε R/C=0.
Γενικά, αυτός ο λόγος μπορεί να πάρει τιμές από 0 έως 1, όπου 1➔βέλτιστη απεικόνιση (ελάχιστη δυνατή κατανάλωση πόρων) και 0➔ανέφικτη απεικόνιση.

V.1.2 *Time-driven λογική προσομοίωσης σε παρελθούσα έρευνα και το βασικό πρόβλημά της*

Μέχρι τώρα σε αντίστοιχες έρευνες έχουν προταθεί λογικές προσομοίωσης της παρακάτω μορφής:



Σχήμα 15: Περιβάλλον time-driven προσομοίωσης για δυναμικά αφικνούμενες VNRs (από [1])

Δηλ. μέχρι τώρα έχουν εφαρμοστεί time-driven simulations θεωρώντας δυναμικές Poisson αφίξεις. Ο χρόνος χωρίζεται σε διαστήματα ίσου μεγέθους Δt. Σε κάθε χρονική στιγμή t ο αλγόριθμος, όπως προσομοιώνεται, προσπαθεί να χαρτογραφήσει όλες τις αιτήσεις εικονικών δικτύων r_i με χρόνους άφιξης $a_i<(t+\Delta t)$. Όλες οι αιτήσεις με $a_i+l_i>t$ διαγράφονται και οι αντίστοιχοι πόροι του υποστρώματος απελευθερώνονται.

Το βασικό πρόβλημα της παραπάνω λογικής προσομοίωσης είναι ότι είναι σύγχρονη και οδηγούμενη από το ρολόι (clock-driven). Δηλ. η τιμή του ρολογιού αυξάνεται κατά σταθερά βήματα, ελέγχονται τα γεγονότα που συνέβησαν σε κάθε βήμα και γίνονται οι αντίστοιχες ενέργειες. Η τεχνική αυτή είναι κατάλληλη για συστήματα διακριτού χρόνου αλλά κρίνεται μάλλον ανεπαρκής για συστήματα συνεχούς χρόνου. Εδώ μάλιστα όπου ο χρόνος που απαιτεί η διενέργεια του αλγορίθμου απεικόνισης είναι άγνωστη και δεν ακολουθεί καμία γνωστή κατανομή, η υιοθέτηση ενός διακριτού μοντέλου προσομοίωσης δεν ενδείκνυται καθώς δεν παρέχει την απαιτούμενη ακρίβεια (όχι δυναμική αλλά bulk προσομοίωση).

V.1.3 *Event-driven λογική προσομοίωσης που χρησιμοποιείται στην εργασία*

Για να αντιμετωπιστεί το παραπάνω πρόβλημα και να προσομοιωθεί ένα πραγματικά δυναμικό σύστημα συνεχούς χρόνου, στην εργασία υιοθετείται η λογική της ασύγχρονης προσομοίωσης, δηλ. της προσομοίωσης οδηγούμενης από τα γεγονότα (event-driven). Εδώ η τιμή του ρολογιού αυξάνεται από την μία στιγμή γεγονότος στην επόμενη, άρα κατά βήματα μεταβλητού μήκους, και σε κάθε στιγμή εκδήλωσης γεγονότος γίνονται οι απαραίτητες ενέργειες. Χρησιμοποιείται πάλι ένα βασικό χρονικό παράθυρο παρατήρησης Δt για τμηματοποίηση της προσομοίωσης από την πλευρά του παρατηρητή, αλλά στην πραγματικότητα η προσομοίωση είναι συνεχούς και όχι διακριτού χρόνου. Το πολύ σημαντικό πλεονέκτημα αυτής της λογικής είναι ότι γεγονότα μπορούν με τις κατάλληλες ενέργειες να δρομολογούν άλλα μελλοντικά γεγονότα που σχετίζονται με αυτά, οδηγώντας τελικά σε μία αυτο-εξελισσόμενη δυναμική προσομοίωση. Το μειονέκτημά της είναι ότι λόγω του ασύγχρονου και δυναμικού χαρακτήρα της δεν είναι εκ των προτέρων γνωστός ο συνολικός απαιτούμενος χρόνος διεξαγωγής της γι' αυτό συνήθως τίθεται ένα άνω όριο επιτρεπτό όριο χρονικής παρατήρησης. Η τεχνική αυτή χρησιμοποιείται σε όλες τις γλώσσες προσομοίωσης (π.χ. Simula) για συστήματα διακριτών γεγονότων.

V.1.4 *Χρονοδρομολόγηση γεγονότων: λεπτομερής περιγραφή*

Σύμφωνα με τη μέθοδο αυτή υπάρχει ένα υποπρόγραμμα (υπορουτίνα) συνδεδεμένο με κάθε τύπο γεγονότος, το οποίο καλείται κάθε φορά που λαμβάνει χώρα το αντίστοιχο γεγονός. Το πρόγραμμα τηρεί ένα είδος «ημερολογίου», το οποίο ονομάζεται λίστα γεγονότων και περιλαμβάνει κάθε στιγμή τα γεγονότα που είναι προγραμματισμένα να συμβούν διατεταγμένα σύμφωνα με τις αντίστοιχες χρονικές στιγμές. Κάθε φορά το ρολόι παίρνει ως τιμή τη χρονική στιγμή του πλησιέστερου γεγονότος (αρχή της λίστας) και καλεί το υποπρόγραμμα του αντίστοιχου τύπου. Το τελευταίο ενημερώνει την κατάσταση του συστήματος , συλλέγει στατιστικά στοιχεία και ενδεχομένως δρομολογεί νέα γεγονότα ενημερώνοντας αντίστοιχα τη λίστα γεγονότων. Οι πληροφορίες για τη λογική αυτή πάρθηκαν από το [26].

V.1.5 *Διαχείριση ουρών*

Στην προσομοίωση χρησιμοποιούνται 3 διπλά συνδεδεμένες ουρές στοιχείων για την διαχείριση των γεγονότων της προσομοίωσης και των αιτήσεων που τα προκαλούν:

- **Ουρά-λίστα γεγονότων** ➔ πάντα ταξινομημένη σύμφωνα με το χρόνο εκδήλωσης των γεγονότων. Το πλησιέστερο στο τρέχον ρολόι γεγονός τοποθετείται στην κορυφή της λίστας και ακολουθούν μεταγενέστερα γεγονότα. Στην κυρίως προσομοίωση μελετάται πάντα το κορυφαίο γεγονός, ενώ ενδιάμεσα στο πρόγραμμα γίνεται εισαγωγή και διαγραφή γεγονότων χωρίς όμως να παραβιάζεται ποτέ η ταξινομημένη διάταξη της λίστας. Κατά την εξέταση των γεγονότων ο καθολικός χρονομετρητής (ρολόι) του προγράμματος παίρνει σε κάθε περίπτωση ως τιμή την χρονική στιγμή εκδήλωσης του κορυφαίου γεγονότος.

- **Ουρά υποδοχής αιτήσεων** ➔ η ουρά αυτή είναι στη βάση της FIFO με συγκεκριμένες όμως παραδοχές. Εδώ τοποθετείται αρχικά κάθε αίτηση που εισέρχεται στο σύστημα και απαιτεί εξυπηρέτηση. Αφού οι αιτήσεις καταφτάνουν δυναμικά αλλά σειριακά στο χρόνο η ουρά αυτή είναι πάντοτε ταξινομημένη ως προς τον χρόνο άφιξης των αιτήσεων. Αιτήσεις που έχουν φτάσει νωρίτερα είναι τοποθετημένες κοντά στην κορυφή της ουράς ενώ μεταγενέστερες στο τέλος της. Προτεραιότητα χαρτογράφησης-εξυπηρέτησης έχουν οι κορυφαίες αιτήσεις. Προσοχή όμως: για να αποφευχθεί το φαινόμενο HOL (Head Of Line) μπλοκαρίσματος, το οποίο παρατηρείται όταν «ελαφριές» αιτήσεις ενσωμάτωσης βρίσκονται πίσω από μία κορυφαία «βαριά» αίτηση που ίσως είναι αδύνατον να ταξινομηθεί, έχει υλοποιηθεί η ακόλουθη διαδικασία:

  Αρχικά ελέγχεται η κορυφαία αίτηση. Αν χαρτογραφηθεί επιτυχώς γίνονται οι κατάλληλες δεσμεύσεις πόρων και εξετάζεται κατόπιν η επόμενη στην ουρά αίτηση. Αν όμως η χαρτογράφηση είναι ανεπιτυχής, τότε δρομολογείται η εξέταση της αμέσως επόμενης στην ουρά κ.ο.κ. Σε περίπτωση που φτάσουμε στο τέλος της ουράς η διαδικασία επαναλαμβάνεται κυκλικά από την κορυφή της ουράς (round-robin). Έτσι μία σε πρώτη εξέταση αποτυχημένη αίτηση έχει την ευκαιρία να εξεταστεί ξανά όταν έρθει κυκλικά η σειρά της. Το σύστημα χαρτογράφησης δεν «κολλάει» σε αυτήν, άρα αποφεύγεται το φαινόμενο της «λιμοκτονίας» των επόμενων στην ουρά αιτήσεων. Κάθε αίτηση διαγράφεται από την ουρά αυτή είτε όταν

χαρτογραφηθεί επιτυχώς και άρα είναι έτοιμη να εξυπηρετηθεί, είτε όταν παρέλθει ο μέγιστος επιτρεπτός χρόνος παραμονής στο σύστημα (μη ομαλή λήξη αίτησης). Η ταξινομημένη διάταξη της ουράς δεν αλλοιώνεται σε καμία περίπτωση. Ο χρόνος αναμονής κάθε αίτησης στο σύστημα λήφθηκε για απλοποίηση της προσομοίωσης ίσος με τη διάρκειά της. Άρα αν μία αίτηση φτάσει τη χρονική στιγμή t_{arr} και έχει διάρκεια Dt_{dur}, θα περιμένει για Dt_{dur} μέχρι να διαγραφεί από το σύστημα (μη ομαλή εκπνοή). Αν χαρτογραφηθεί επιτυχώς πριν από τη στιγμή t_{arr+} Dt_{dur}, π.χ. τη στιγμή t_{map} τότε θα περάσει την επόμενη ουρά και θα εκπνεύσει ομαλά τη στιγμή t_{map}+ Dt_{dur}.

- **Ουρά εξυπηρέτησης αιτήσεων ➔** η ουρά αυτή φιλοξενεί τις αιτήσεις που έχουν χαρτογραφηθεί επιτυχώς και των οποίων τα εικονικά δίκτυα έχουν υλοποιηθεί πάνω στο υπόστρωμα. Είναι ταξινομημένη σύμφωνα με την πλησιέστερη χρονικά ομαλή εκπνοή αίτησης. Κορυφαία αίτηση είναι εκείνη που θα λήξει νωρίτερα. Κάθε αίτηση που λήγει ομαλά, όταν παρέλθει ο μέγιστος επιτρεπτός χρόνος λειτουργίας του αντίστοιχου εικονικού δικτύου, διαγράφεται από την ουρά.

V.1.6 *Βασικά γεγονότα προσομοίωσης*

Τα βασικά γεγονότα που συμβαίνουν κατά τη διάρκεια της προσομοίωσης είναι τα ακόλουθα:

- **Άφιξη αίτησης➔**Δημιουργείται μία καινούρια αίτηση ενσωμάτωσης εικονικού δικτύου και μεταφέρεται στην αρχική ουρά αναμονής για μελλοντική εξυπηρέτηση. Δρομολογείται το γεγονός της μη ομαλής λήξης της (χωρίς δηλ. να χαρτογραφηθεί ποτέ). Επίσης δρομολογείται η επόμενη Poisson άφιξη (διαστήματα ανάμεσα σε διαδοχικές αφίξεις ακολουθούν την εκθετική κατανομή με παράμετρο λ = ρυθμός αιτήσεων/sec). Επιπλέον, αν δεν έχει ήδη δρομολογηθεί το γεγονός εξέτασης της αρχικής ουράς για χαρτογράφηση, δρομολογεί την εξέταση του εαυτού της.

- **Έλεγχος διαθέσιμης απεικόνισης➔**Εξετάζεται αν η παρούσα αίτηση προς εξέταση μπορεί να χαρτογραφηθεί. Αν ναι δρομολογείται το γεγονός έναρξης εξυπηρέτησης της αίτησης. Σε κάθε περίπτωση δρομολογείται το γεγονός εξέτασης της επόμενης στην ουρά υποδοχής αίτησης (ή κυκλικά της επικεφαλής). Αν η ουρά υποδοχής είναι κενή από αιτήσεις η δρομολόγηση του γεγονότος εξέτασής της επαφίεται στην επόμενη χρονικά άφιξη, όπως αναφέρθηκε παραπάνω.

- **Έναρξη εξυπηρέτησης αίτησης➔**Μετά την επιτυχή χαρτογράφηση έχουν δεσμευτεί οι αναγκαίοι πόροι του υποστρώματος και η αίτηση μεταφέρεται από την αρχική ουρά υποδοχής στην ουρά εξυπηρετούμενων αιτήσεων. Δρομολογείται το γεγονός της ομαλής λήξης της.

- **Μη ομαλή λήξη αίτησης χωρίς χαρτογράφηση➔**Η αίτηση λήγει μη ομαλά ενώ βρίσκεται ακόμα στην ουρά υποδοχής. Κάθε γεγονός που σχετίζεται με αυτήν και είχε ήδη δρομολογηθεί διαγράφεται από την λίστα και η αίτηση διαγράφεται από την ουρά υποδοχής και καταργείται από το σύστημα. Προσοχή όμως: εάν αυτή η αίτηση ήταν η επόμενη προς εξέταση από το

σύστημα χαρτογράφησης τότε λαμβάνουν χώρα οι απαραίτητες ενέργειες για την παράδοση της σκυτάλης εξέτασης στην επόμενη μη λήξασα αίτηση.

- **Ομαλή λήξη αίτησης μετά από χαρτογράφηση**➔Η αίτηση λήγει ομαλά μετά την εξυπηρέτησή της, απελευθερώνει τους πόρους του υποστρώματος που είχε δεσμεύσει, διαγράφεται από την ουρά εξυπηρέτησης και καταργείται από το σύστημα.

V.1.7 *Τρόπος παραγωγής αποτελεσμάτων προσομοίωσης*

Τα βασικά αποτελέσματα που παράγει η προσομοίωση είναι τα εξής:

- **Συνολικός θεωρητικός χρόνος προσομοίωσης**➔υπολογίζεται ως ο χρόνος εκδήλωσης του τελευταίου στη λίστα γεγονότος. Ο χρόνος αυτός αναφέρεται για λόγους πληρότητας, στα αποτελέσματα λαμβάνεται υπόψη μόνο ο πραγματικός χρόνος του προγράμματος αφού αυτός ενδιαφέρει κυρίως.

- **Συνολικός πραγματικός χρόνος προσομοίωσης**➔υπολογίζεται εντός του κώδικα με μέτρηση της συνολικής πραγματικής διάρκειας εκτέλεσης του προγράμματος.

- **Ποσοστό αποδεκτότητας (Acceptance Ratio-AR)** ➔ (αριθμός αιτήσεων που απεικονίζονται επιτυχώς στο υπόστρωμα) / (συνολικός αριθμός αφικνούμενων αιτήσεων). Το αποτέλεσμα αυτό είναι συσσωρευτικό και εξάγεται στο τέλος από την προσομοίωση όπου έχουν υπολογιστεί τα πλήθη των συνολικών αιτήσεων και εκείνων που χαρτογραφήθηκαν και έληξαν κανονικά. Δείχνει απλά την απόδοση του αλγορίθμου όσον αφορά το ποσοστό των επιτυχών χαρτογραφήσεων που παράγει στο σύνολο των αφίξεων που λαμβάνει καθ' όλη τη διάρκεια της προσομοίωσης.

- **Μέσο R/C** ➔ (κέρδος παρόχου υπηρεσίας)/(κόστος χαρτογράφησης). Το αποτέλεσμα αυτό είναι η μέση τιμή των R/C όλων των αιτήσεων που έχουν καταφθάσει στο σύστημα στο διάστημα διεξαγωγής της προσομοίωσης. Υπενθύμιση: $R/C \in (0,1)$ με 0➔ανέφικτη χαρτογράφηση, 1➔βέλτιστη χαρτογράφηση. Άρα επιθυμητές τιμές είναι όσες βρίσκονται κοντά στο 1.

V.1.8 *Βασικές παράμετροι προσομοίωσης*

V.1.8.1 *Χρονική διάρκεια και ρυθμός αφίξεων*

Βασική μονάδα χρόνου παρατήρησης = Δt = 20sec
Μέσος ρυθμός αφίξεων Poisson = λ = 5 αιτ./Δt = 0,2 αιτ./sec
Διάρκεια αιτήσεων = ομοιόμορφα κατανεμημένη στο (1,9)*Δt.
Μέγιστος αριθμός αφικνούμενων αιτήσεων = 20 αιτ. για το μέρος Α, 100 για το Β
Μέγιστος θεωρητικός χρόνος προσομοίωσης = 50*Δt = 1000 sec
➔Οι παραπάνω παράμετροι διατηρούνται σταθερές σε κάθε εκτέλεση της προσομοίωσης ενώ μεταβάλλεται ο αριθμός των κόμβων και οι απαιτήσεις των εικονικών δικτύων. Η διαδικασία κατασκευής του υποστρώματος και των εικονικών δικτύων, καθώς και ο πειραματικός προσδιορισμός των παραμέτρων τους περιγράφεται παρακάτω.

V.1.8.2 *Κατασκευή υποστρώματος και εικονικών δικτύων*

Για την κατασκευή των απαιτούμενων τοπολογιών-γράφων των δικτύων χρησιμοποιήθηκε C κώδικας που παρατίθεται στο παράρτημα στο τέλος της εργασίας.

Σε όλα τα δίκτυα (φυσικά και εικονικά) ακολουθείται η εξής τοπολογία: αστέρια τερματικών γύρω από routers και αυθαίρετη-τυχαία σύνδεση των routers μεταξύ τους. Για κάθε τερματικό επιλέγεται τυχαία ένας router προς σύνδεση (το id του router ακολουθεί την ομοιόμορφη στο διάστημα (min_router_id, max_router_id)). Επιπλέον, κάθε ζευγάρι routers είναι τυχαία συνδεδεμένο με πιθανότητα 0.5. Συνεπώς σε κάθε κατασκευασθέν δίκτυο υφίσταται κατά μέσο όρο ο ακόλουθος αριθμός links: terminal_num+router_num*(router_num-1)/4. Προσοχή πρέπει να δοθεί στο γεγονός ότι για την παραγωγή όλων των συνεχών κατανομών πιθανότητας υιοθετήθηκε το μοντέλο της αντιστροφής συνεχών κατανομών ([25]), με βάση την ομοιόμορφη κατανομή στο (0,1).

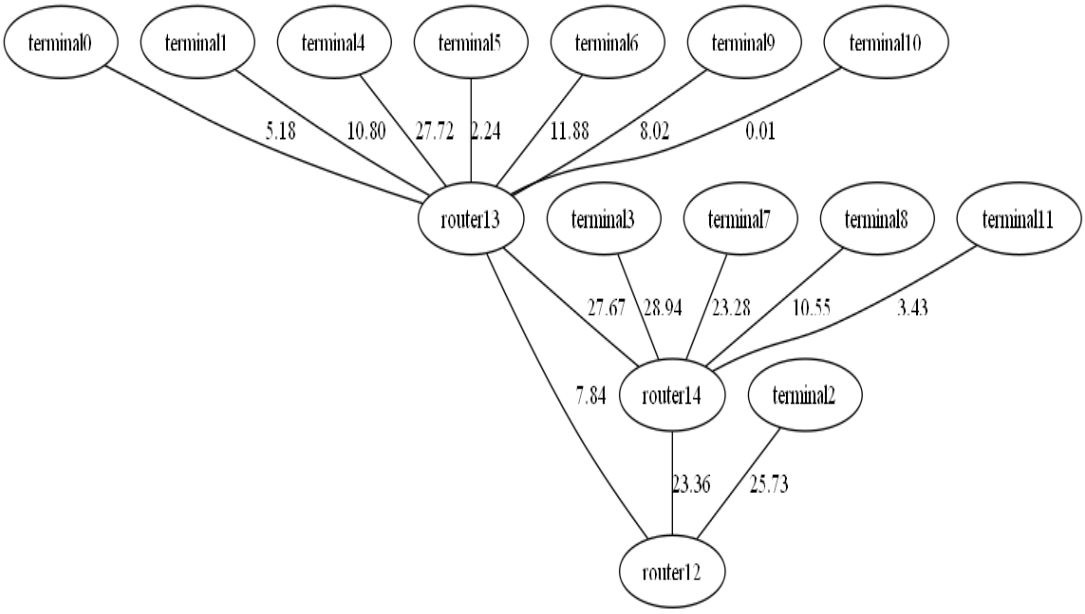
Το φυσικό δίκτυο-υπόστρωμα ρυθμίστηκε να έχει περίπου 150 κόμβους με αναλογία routers/terminals=1/4 (30 routers – 120 terminals) και περίπου 350 links. Αυτή η κλίμακα αντιστοιχεί σε ένα AS (Autonomous System) μεσαίου μεγέθους. Η CPU των φυσικών τερματικών ακολουθεί την ομοιόμορφη κατανομή στο (0,100) όπως και το BW των φυσικών links. Ο αριθμός των slices των φυσικών τερματικών ακολουθεί την ομοιόμορφη στο (0,10) ενώ των routers στο (0,50). Τα εικονικά δίκτυα έχουν μεταβαλλόμενο αριθμό κόμβων από 10 έως 40 με αναλογία routers/terminals=1/4. Για τις ανάγκες της προσομοίωσης (απλοποίηση διαδικασίας) δεν ενεργοποιήθηκαν οι πρόσθετοι περιορισμοί που αφορούν την επιθυμητή θέση των φυσικών απεικονίσεων και της τοποθέτησης των GateWays. Αυτά τα constraints μπορούν να ενεργοποιηθούν για συγκεκριμένες πραγματικές τοπολογίες ρεαλιστικών απαιτήσεων σε μελλοντική εργασία σχετικά με το θέμα. Χρησιμοποιούνται μόνο το BW και τα max_hops ως απαιτήσεις των εικονικών ζεύξεων και η CPU ως απαίτηση των εικονικών τερματικών (και φυσικά για όλους τους κόμβους η απαίτηση ύπαρξης slice για απεικόνιση). Ορίζεται max_bw=max_cpu=β.

➔Στο μέρος Α της προσομοίωσης παρουσιάζονται διαγράμματα για τις ακόλουθες τιμές μεγεθών των εικονικών δικτύων:
10 nodes: 2 routers, 8 terminals
20 nodes: 4 routers, 16 terminals
30 nodes: 6 routers, 24 terminals
40 nodes: 8 routers, 32 terminals
Η σχεδίαση γίνεται για τιμές β=10,30,50,70,90 και max_hops=2,3,4. Για κάθε τριπλέτα (nodes, β, max_hops) γίνεται επεξεργασία 20 αιτήσεων, άρα συνολικά στο μέρος Α γίνεται επεξεργασία για 4 (τιμές για μέγεθος) * 5 (τιμές για BW) * 3 (τιμές για max_hops) * 20 (αιτήσεις κάθε φορά) = 1200 αιτήσεις. Ανά 20 αιτήσεις παρατηρείται **ομογενές** μείγμα, δηλ. κάθε εικοσάδα έχει στο σύνολό της ίδιες τιμές της τριπλέτας. Άρα μελετώνται 60 ομογενή δείγματα των 20 αιτήσεων το καθένα.

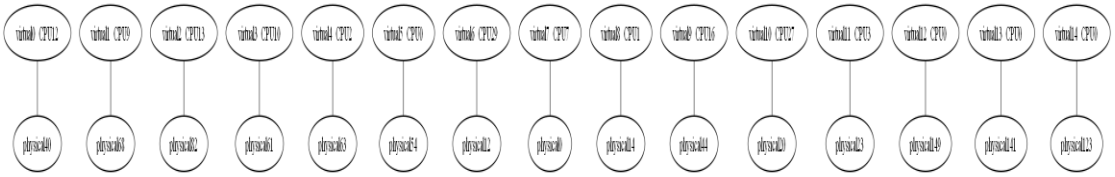
➔Στο μέρος Β της προσομοίωσης εξάγονται οι τιμές του πραγματικού χρόνου εκτέλεσης, του AR και του λόγου R/C για ένα **ετερογενές** μείγμα αιτήσεων διαφορετικών απαιτήσεων. Συνολικά γίνεται επεξεργασία 100 αιτήσεων με μεταβαλλόμενο αριθμό routers από 2 έως 8 και τερματικά τετραπλάσια κάθε φορά του αριθμού των routers. Το BW και η CPU είναι ομοιόμορφα κατανεμημένα στο (0,β), όπου β ομοιόμορφα κατανεμημένο στο [10,90] ενώ τα max_hops παίρνουν τις τιμές 2,3,4 με χρήση της ομοιόμορφης στο [2,4].

V.1.9 *Γραφικό παράδειγμα χαρτογράφησης ενός μονού VN (single-VN mapping)*

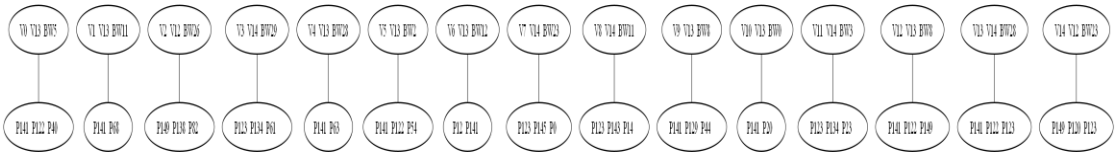
Εικονικό δίκτυο (12 τερματικά, 3 routers, β =30, max_hops=2)



Απεικόνιση κόμβων (φαίνονται οι απαιτήσεις CPU των virtual terminals)



Απεικόνιση ζεύξεων (φαίνονται οι απαιτήσεις BW των virtual links ενώ max_hops=2)



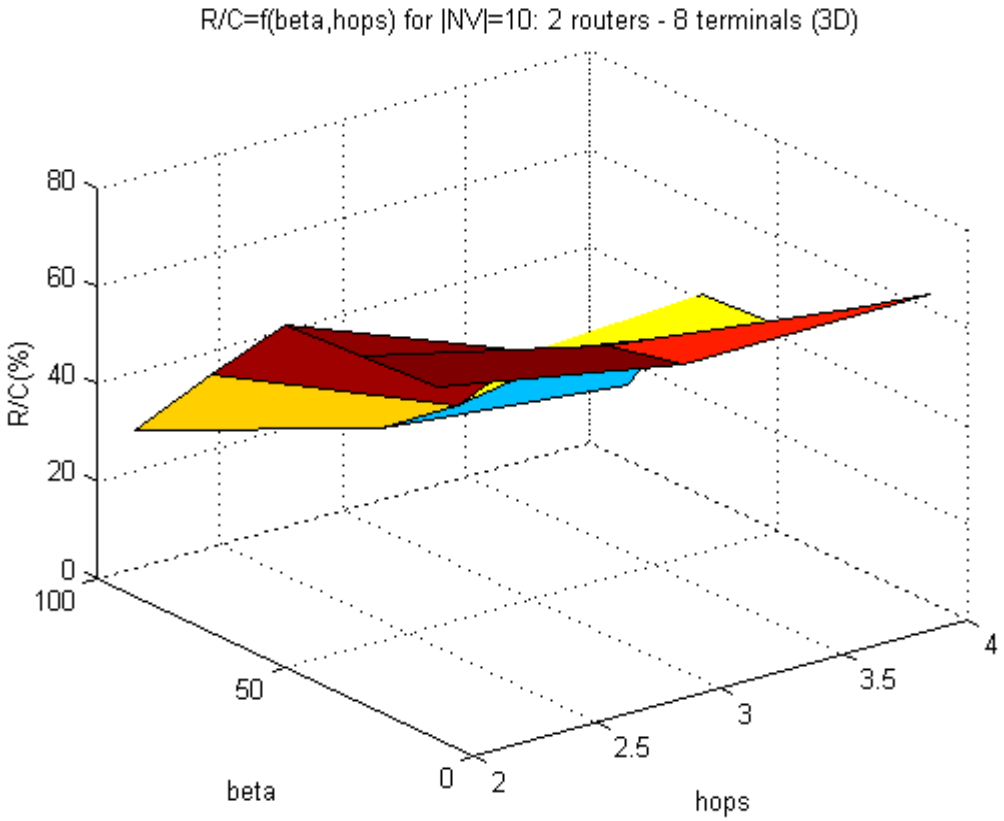
Το υπόστρωμα δεν απεικονίζεται εδώ λόγω μεγέθους, αλλά τα βασικά του στοιχεία είναι ότι διαθέτει 30 routers (slices_num ομοιόμορφος στο [0,50]) και 120 terminals (CPU ομοιόμορφη στο [0,100], slices_num ομοιόμορφο στο [0,10]) και 350 ζεύξεις με BW ομοιόμορφο στο [0,100]. Αυτή η περίπτωση είναι ένα απλό παράδειγμα για γραφική εξήγηση του τι σημαίνει χαρτογράφηση εικονικού δικτύου.

➔Στο γενικευμένο περιβάλλον προσομοίωσης που δημιουργήθηκε, χαρτογραφήσεις όπως οι παραπάνω λαμβάνουν χώρα ακολουθιακά και με πολύ μεγάλη ταχύτητα.

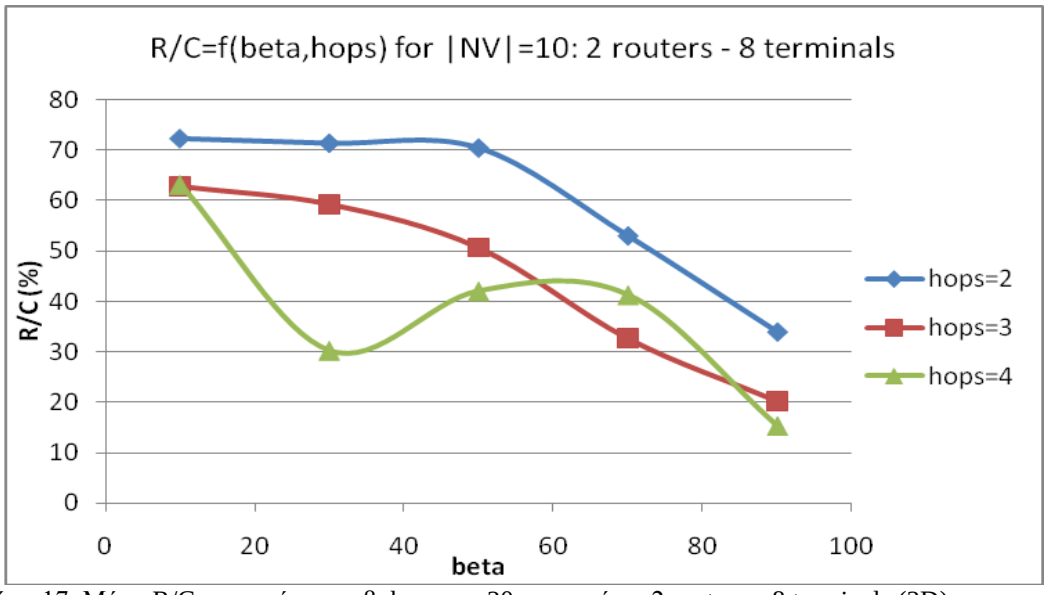
V.2 *Αποτελέσματα simulation για διαφορετικές παραμέτρους των εικονικών δικτύων*

V.2.1 *Μέρος Α: Διαγράμματα (3D - 2D) R/C, AR, Time για διαφορετικές τριπλέτες τιμών (μέγεθος, β, max_hops) για ομογενή δείγματα 20 αιτήσεων/δείγμα*

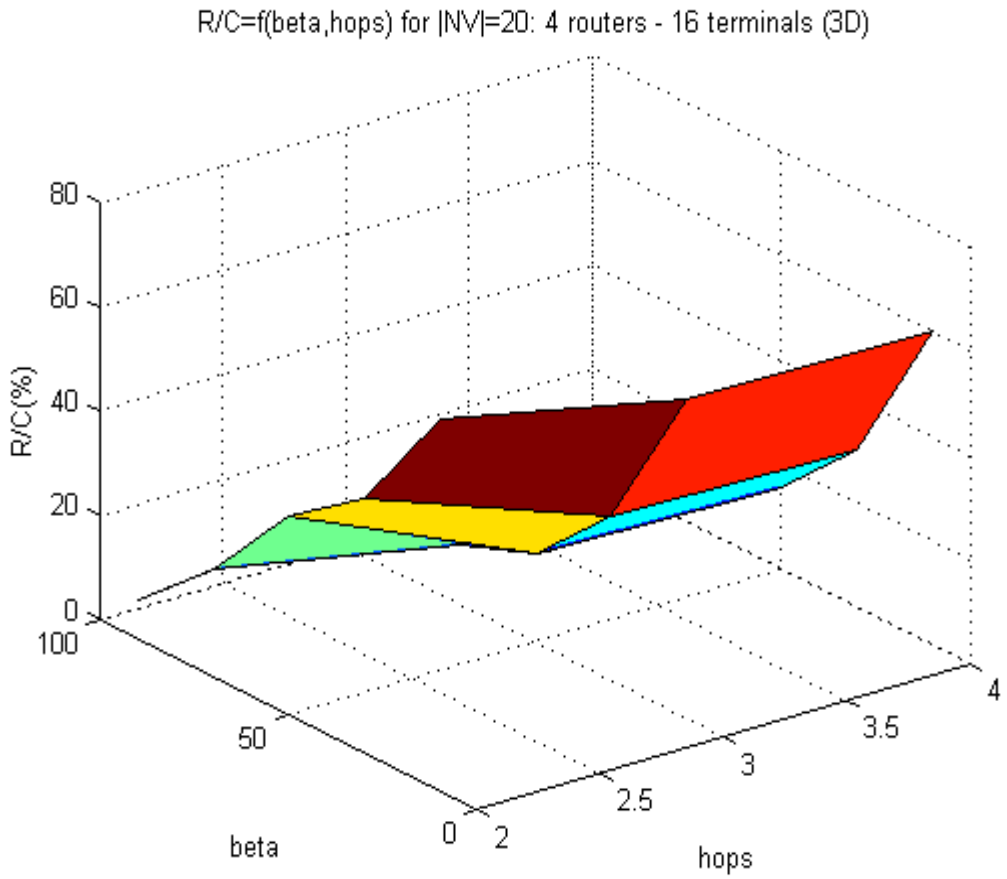
V.2.1.1 *Revenue-to-Cost Ratio R/C (%) για διάφορες τιμές μεγεθών εικονικών δικτύων,β, hops*



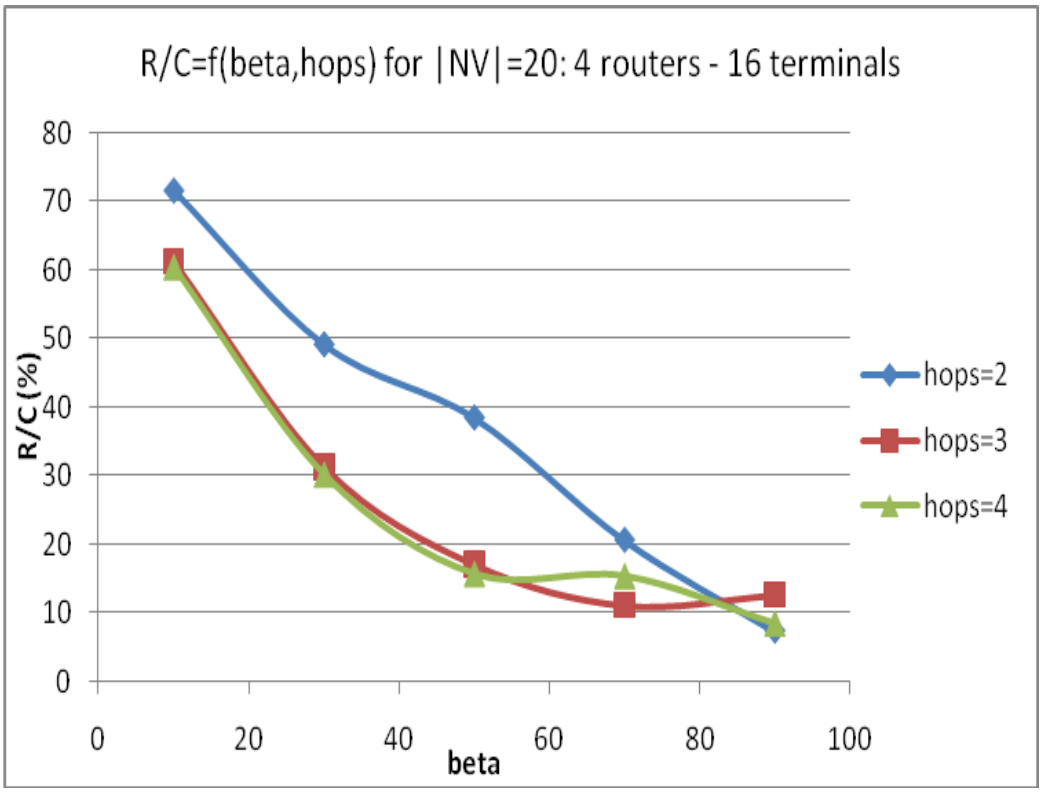
Σχήμα 16: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (3D)



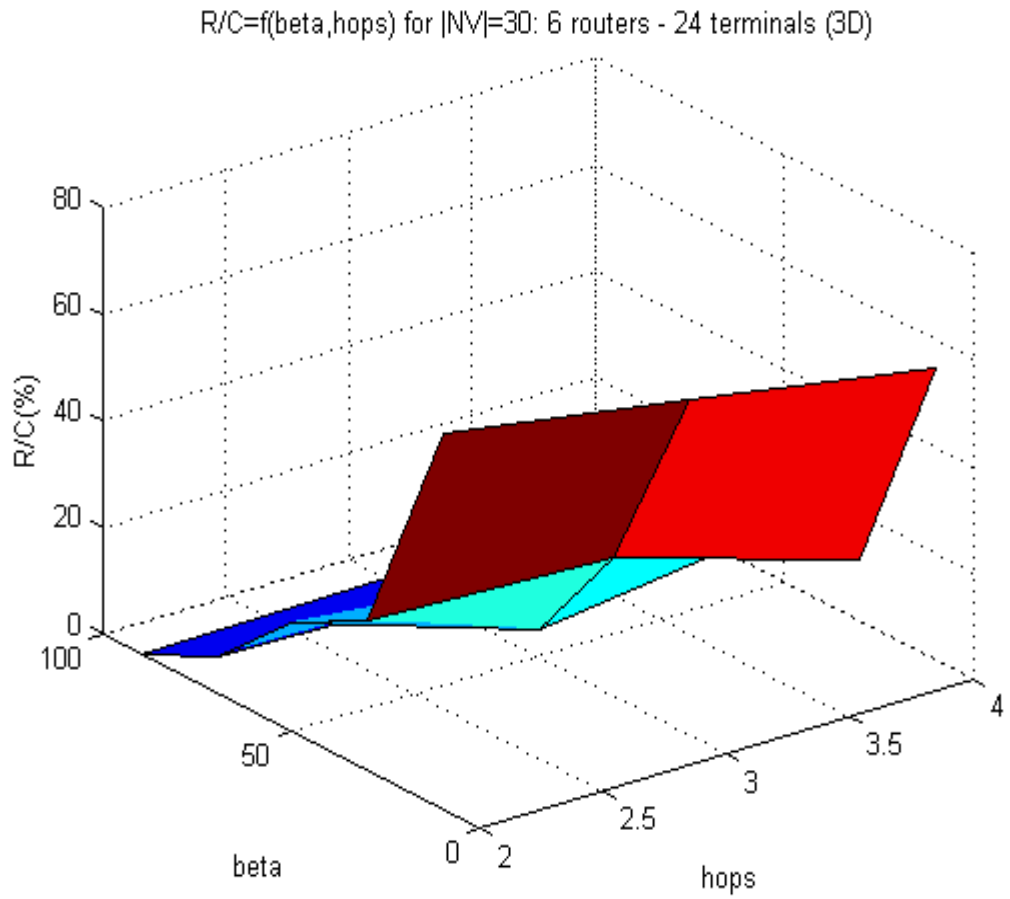
Σχήμα 17: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (2D)



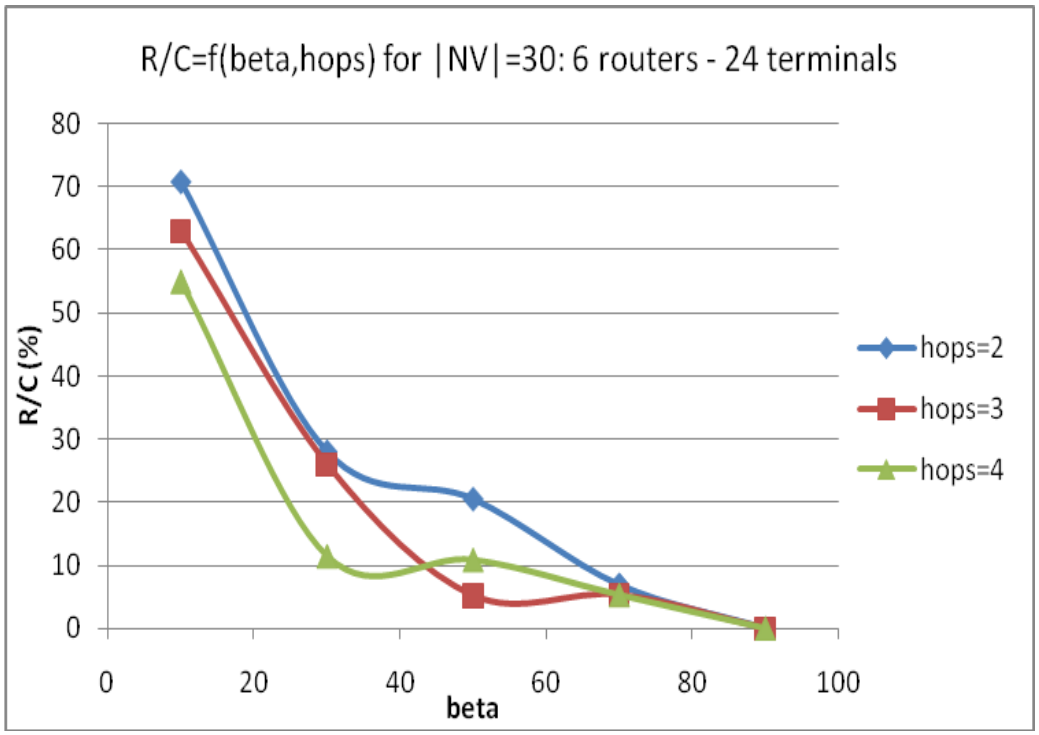
Σχήμα 18: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (3D)



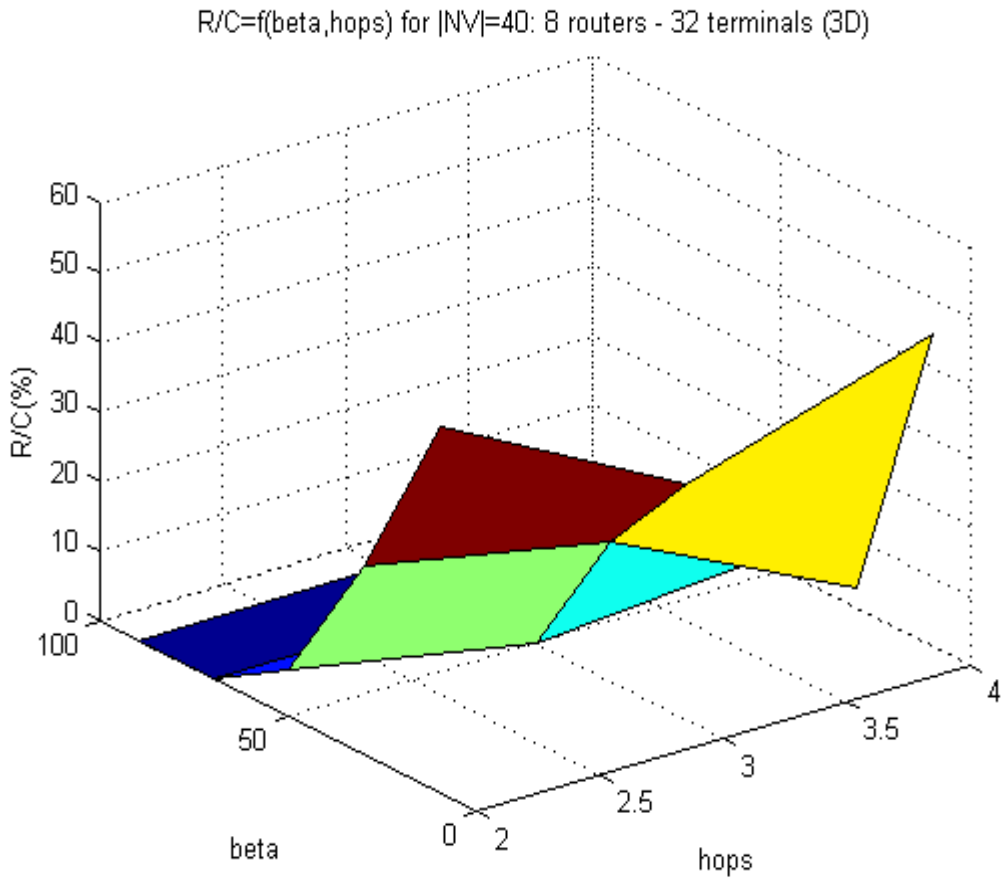
Σχήμα 19: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (2D)



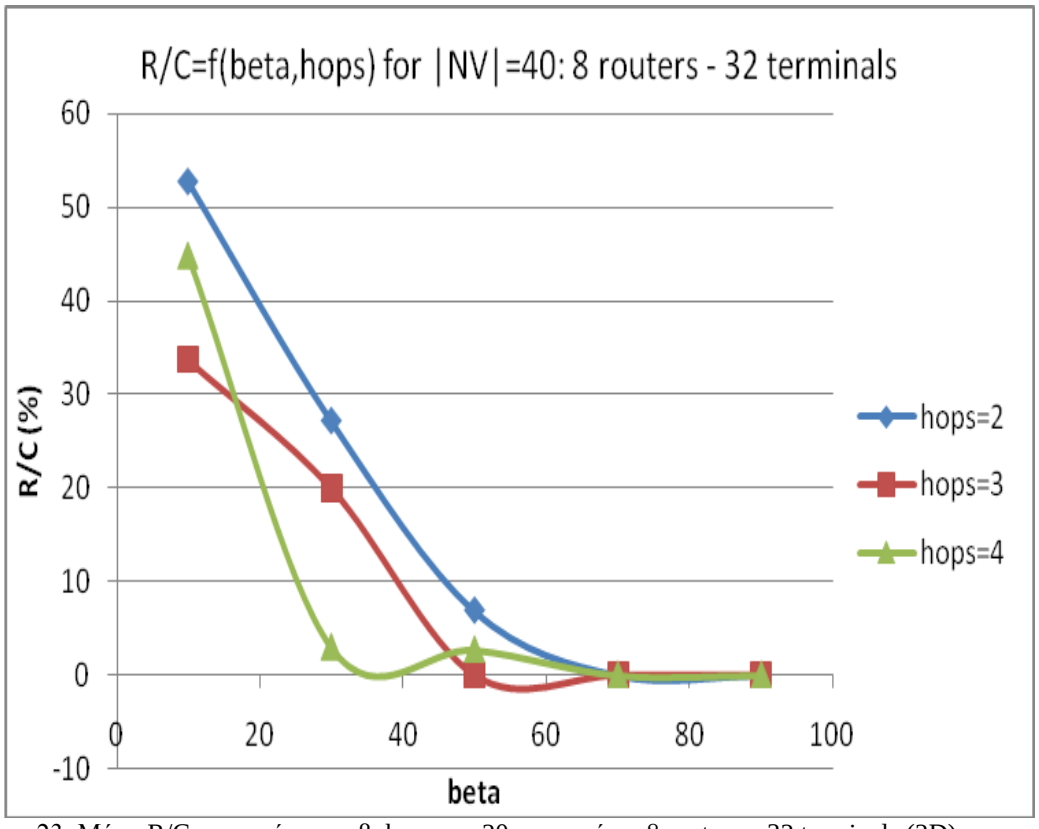
Σχήμα 20: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (3D)



Σχήμα 21: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (2D)

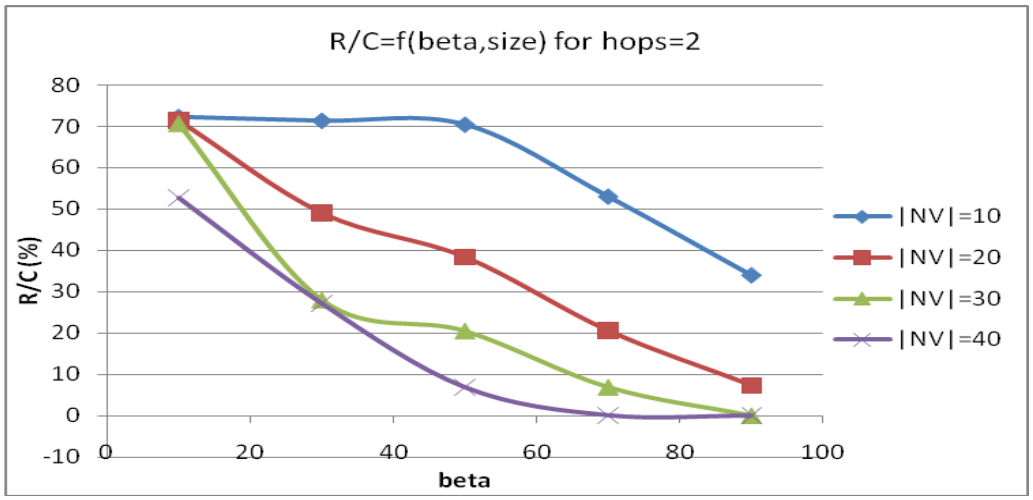


Σχήμα 22: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (3D)

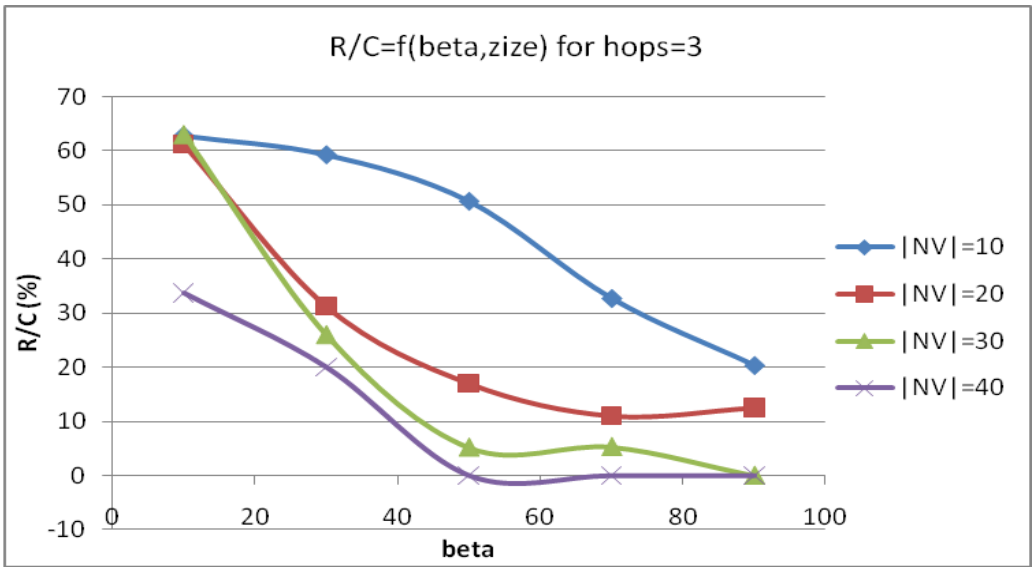


Σχήμα 23: Μέσο R/C ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (2D)

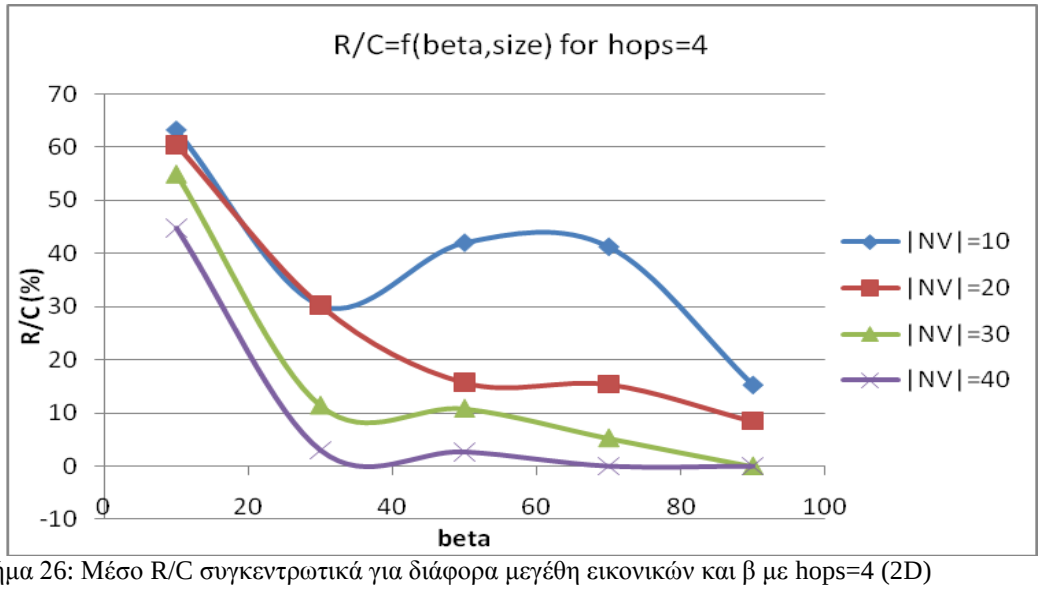
Συγκεντρωτικά R/C για διάφορες τιμές hops



Σχήμα 24: Μέσο R/C συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=2 (2D)

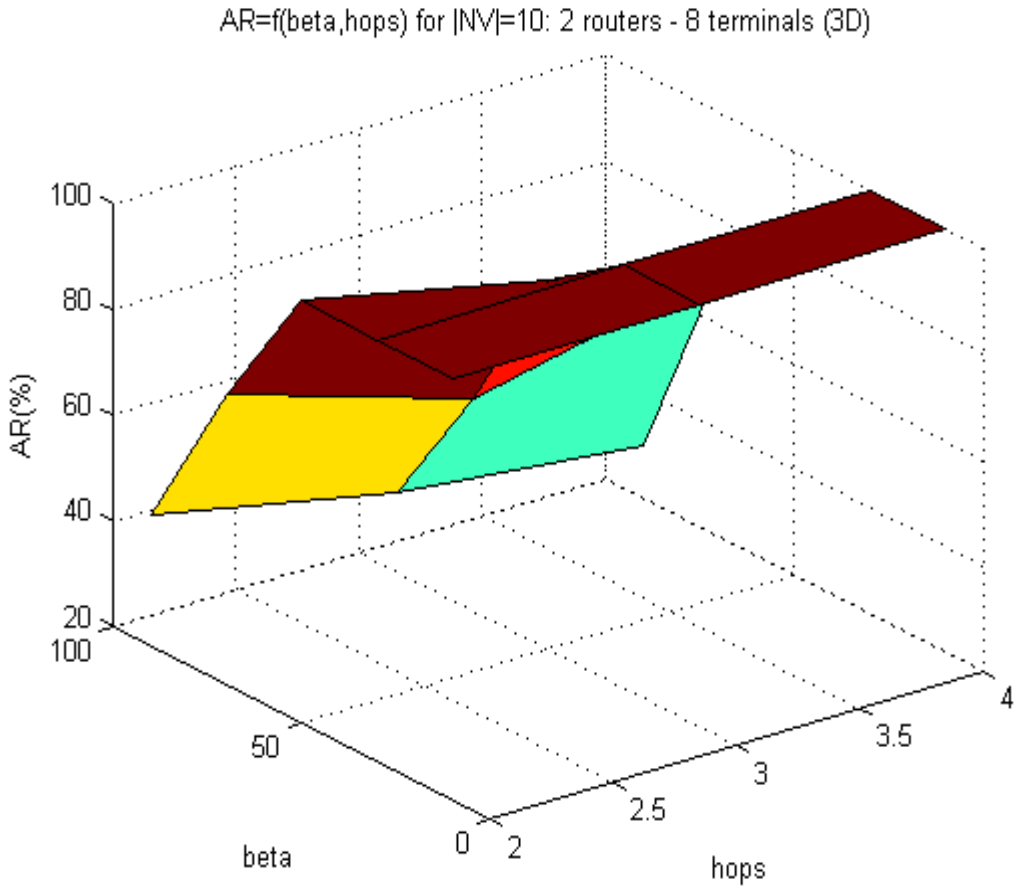


Σχήμα 25: Μέσο R/C συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=3 (2D)

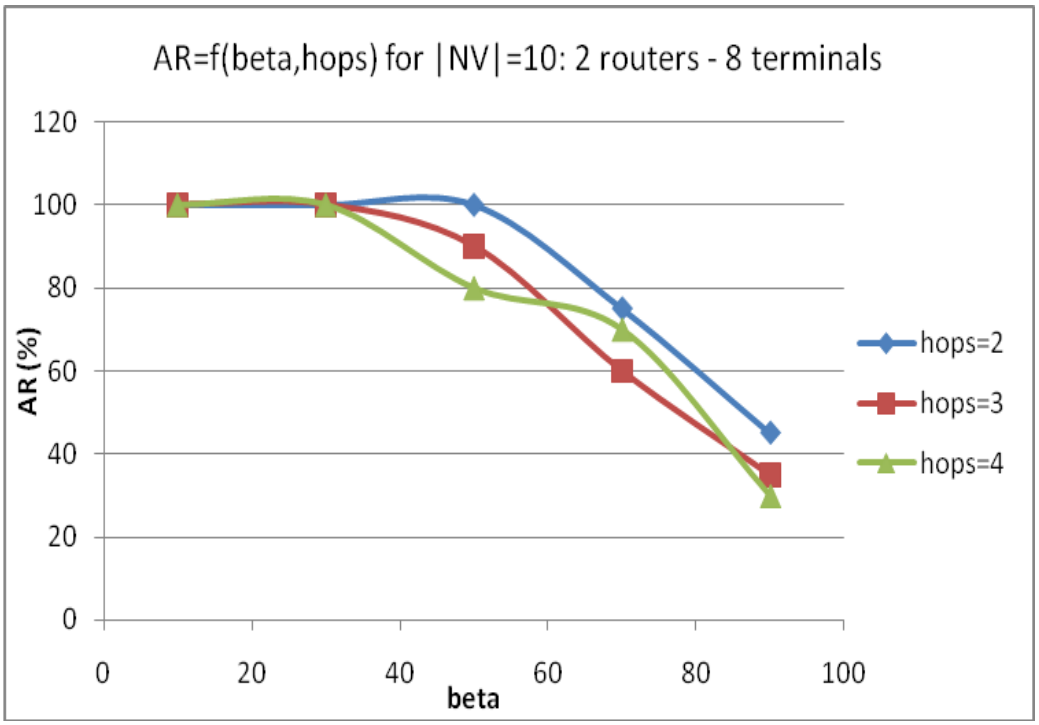


Σχήμα 26: Μέσο R/C συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=4 (2D)

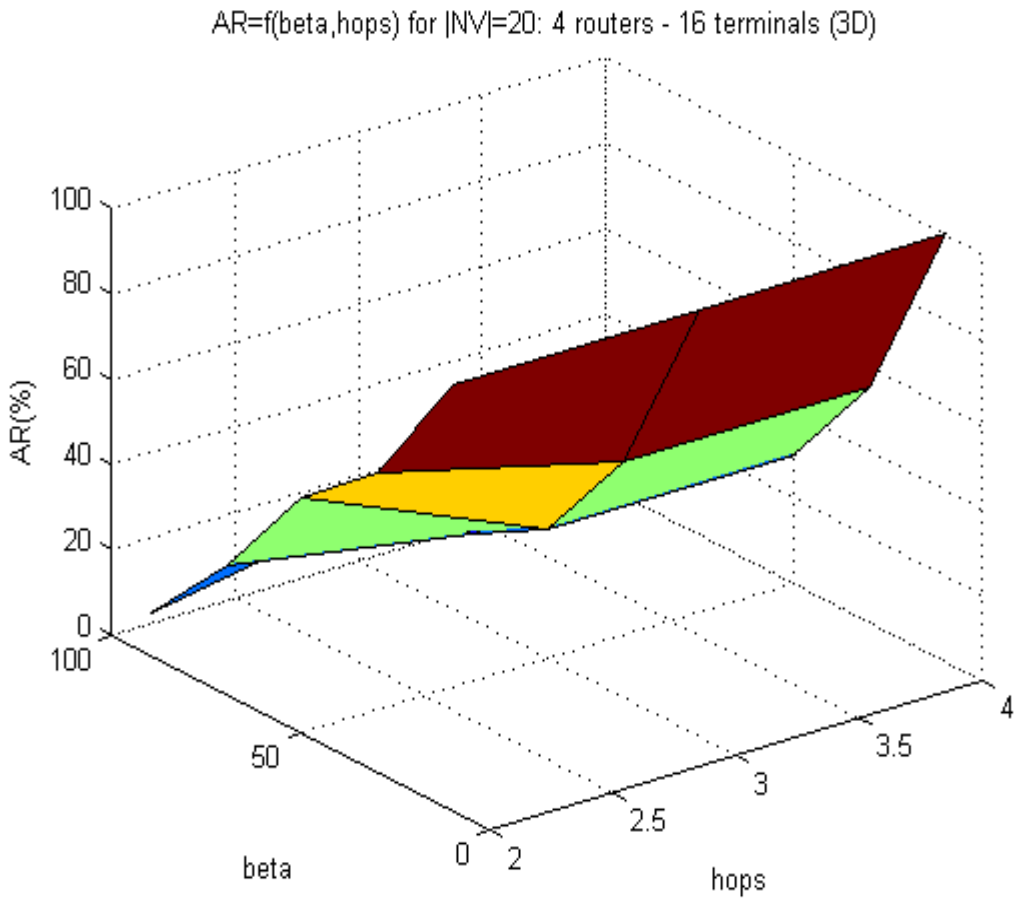
V.2.1.2 *Acceptance-Ratio AR (%) για διάφορες τιμές μεγεθών εικονικών δικτύων,β,hops*



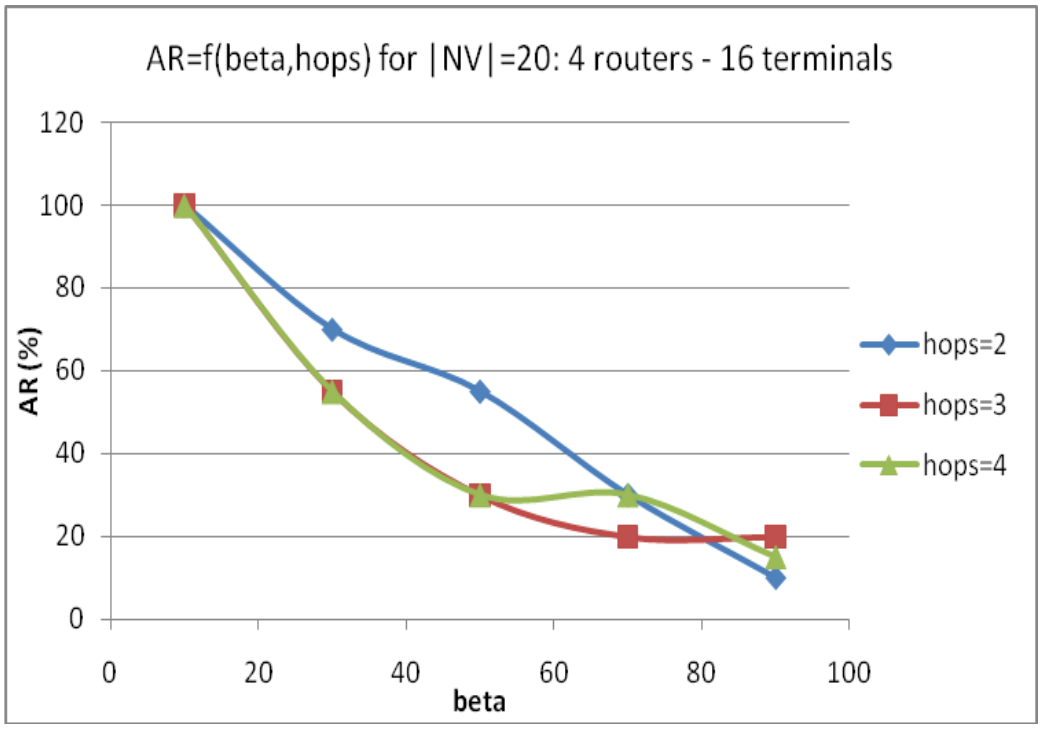
Σχήμα 27: AR ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (3D)



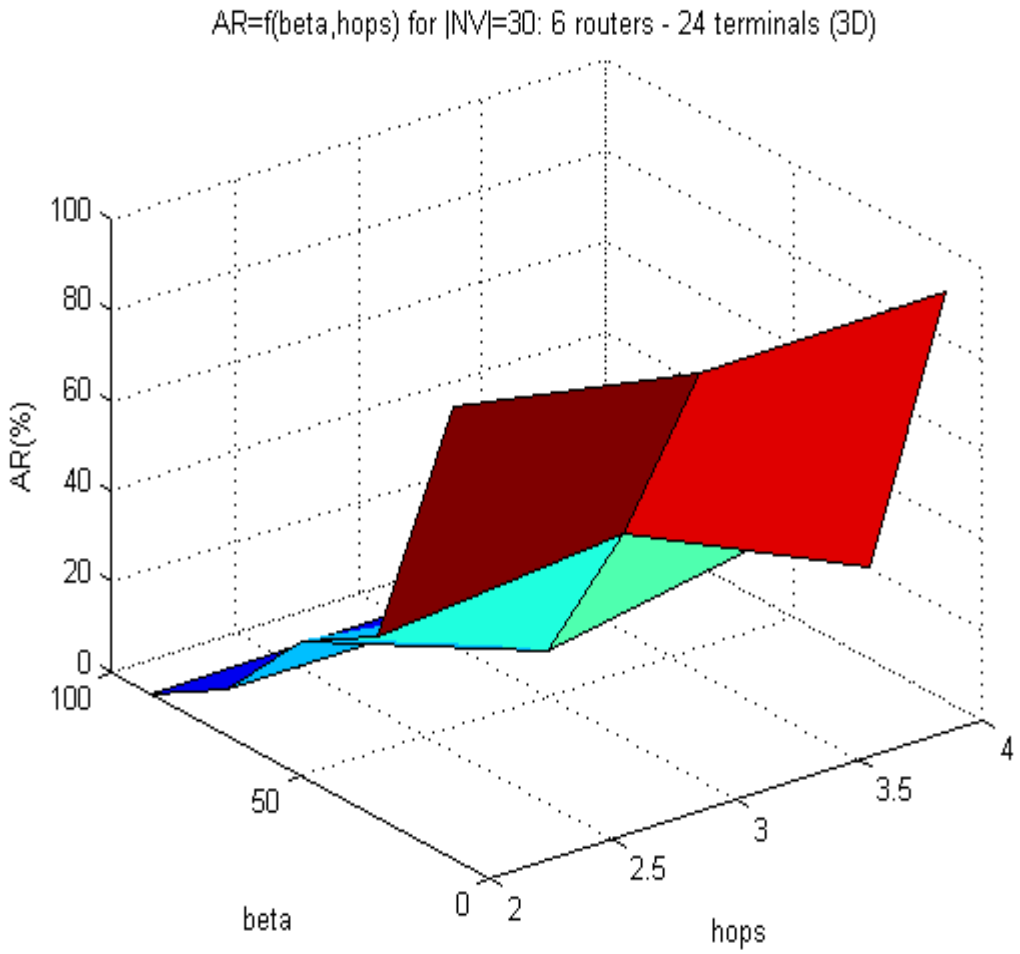
Σχήμα 28: AR ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (2D)



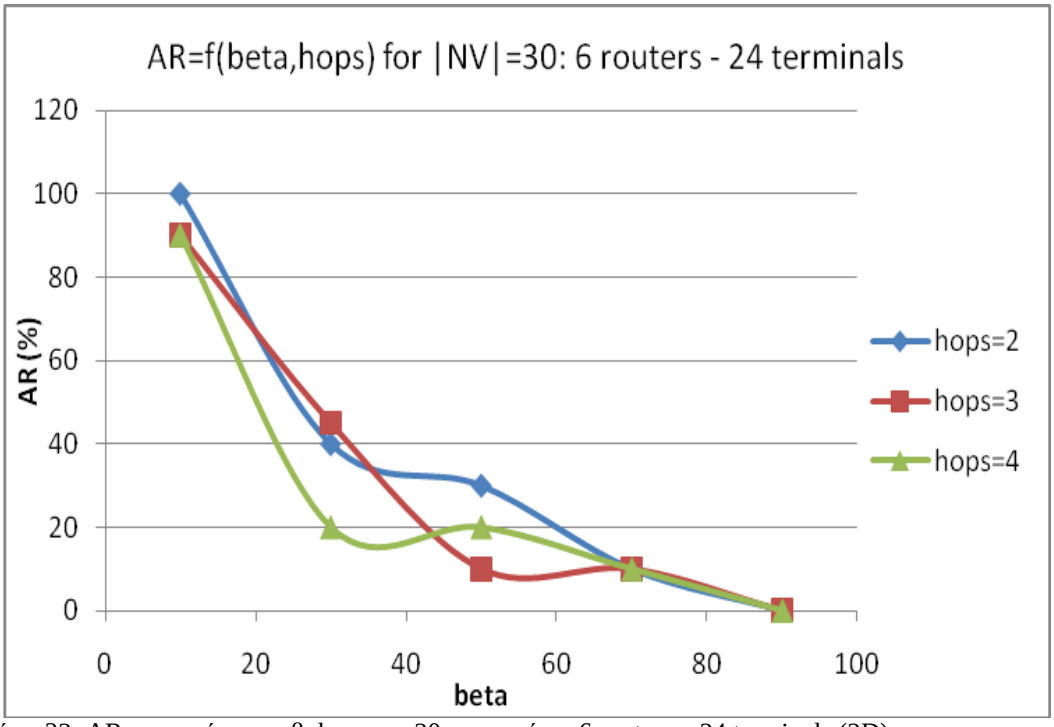
Σχήμα 29: AR ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (3D)



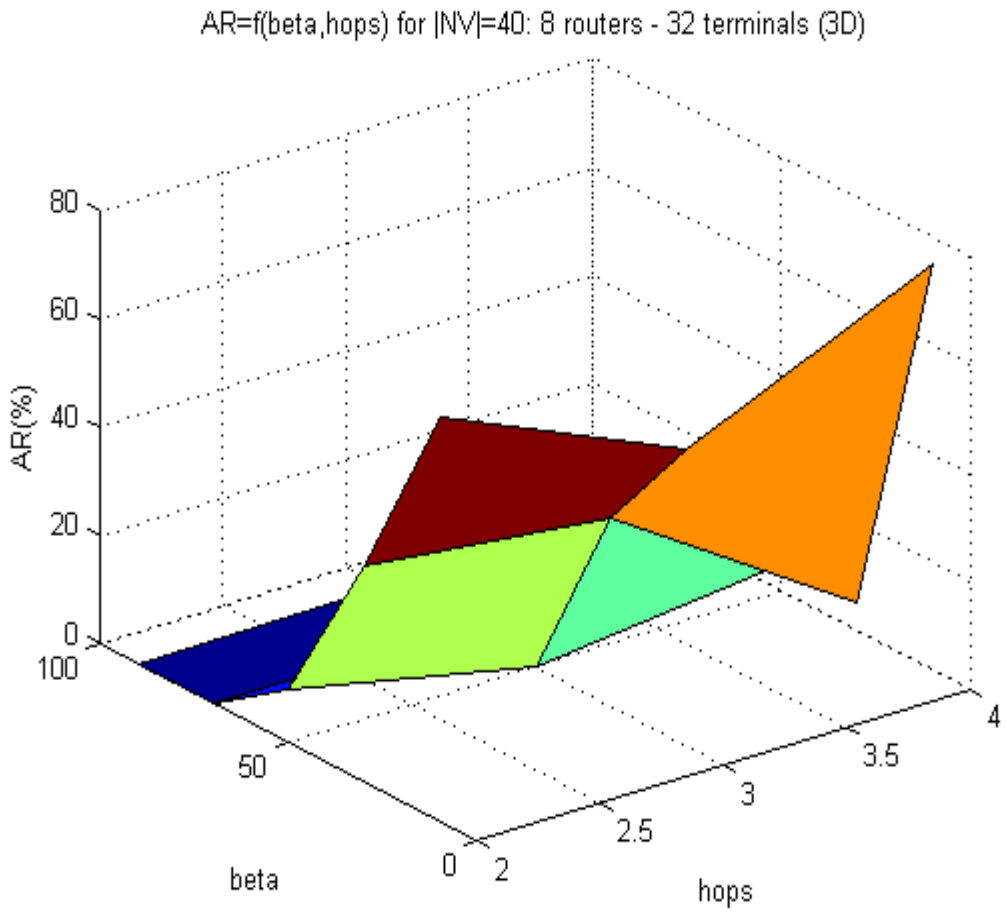
Σχήμα 30: AR ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (2D)



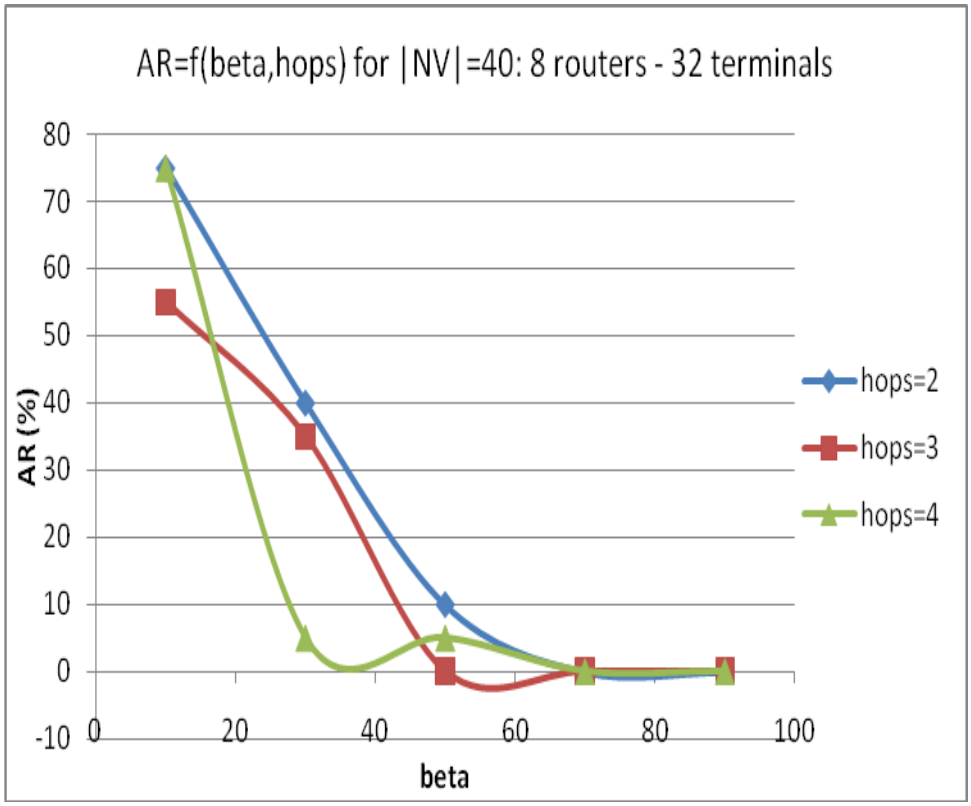
Σχήμα 31: AR ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (3D)



Σχήμα 32: AR ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (2D)

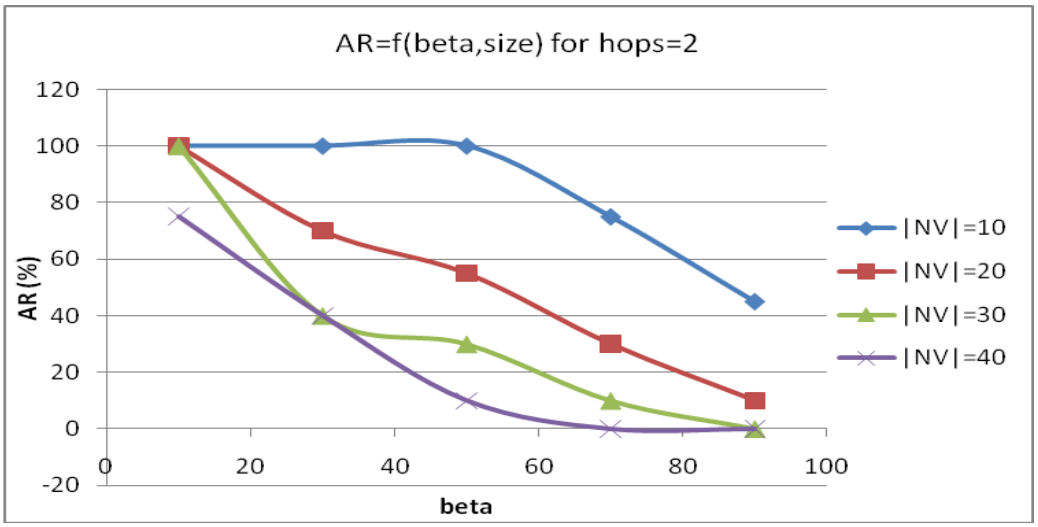


Σχήμα 33: AR ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (3D)

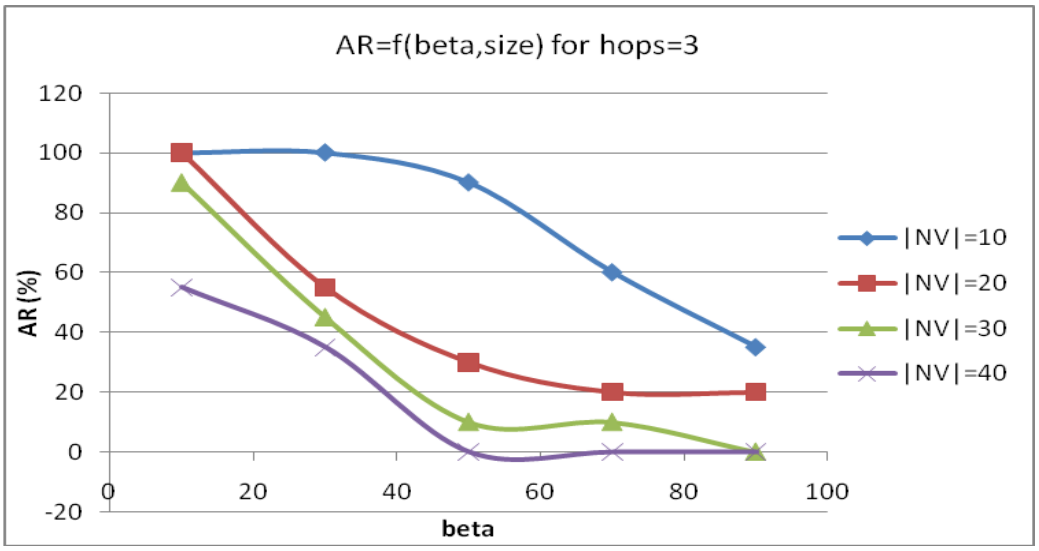


Σχήμα 34: AR ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (2D)

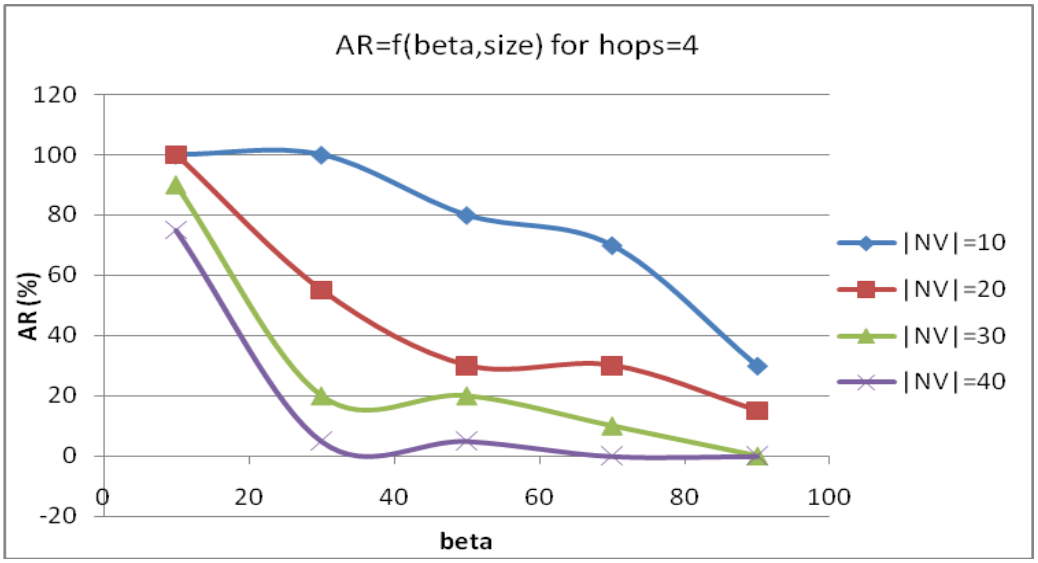
Συγκεντρωτικά AR για διάφορες τιμές hops



Σχήμα 35: AR συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=2 (2D)

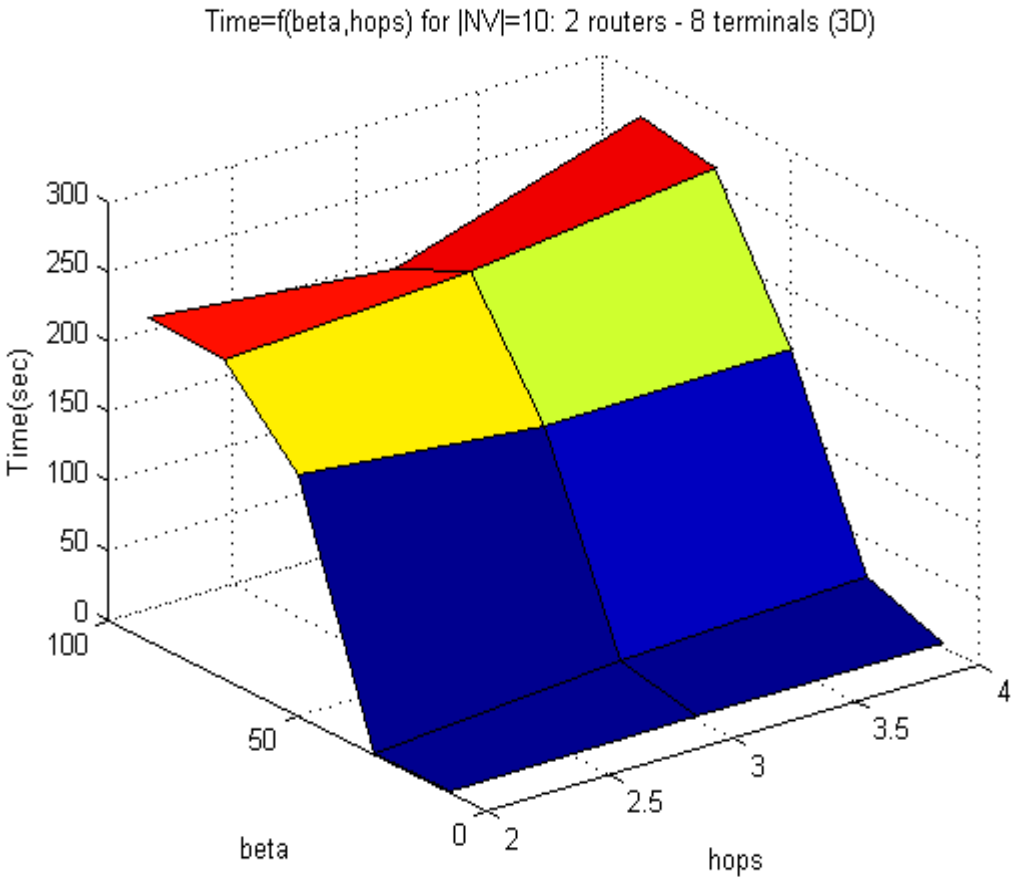


Σχήμα 36: AR συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=3 (2D)

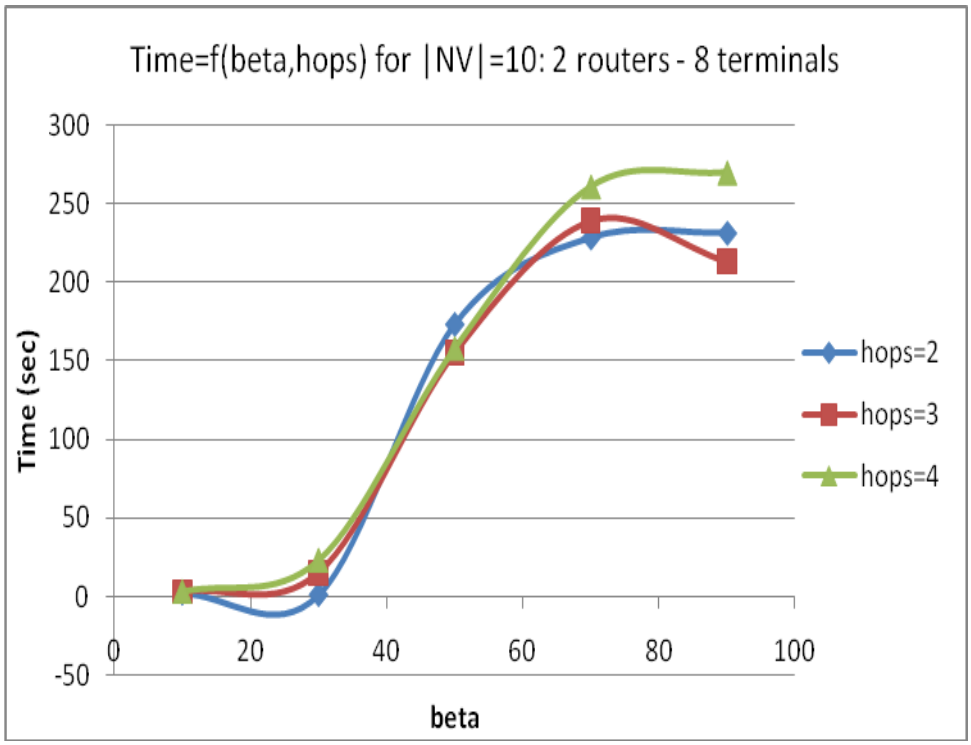


Σχήμα 37: AR συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=4 (2D)

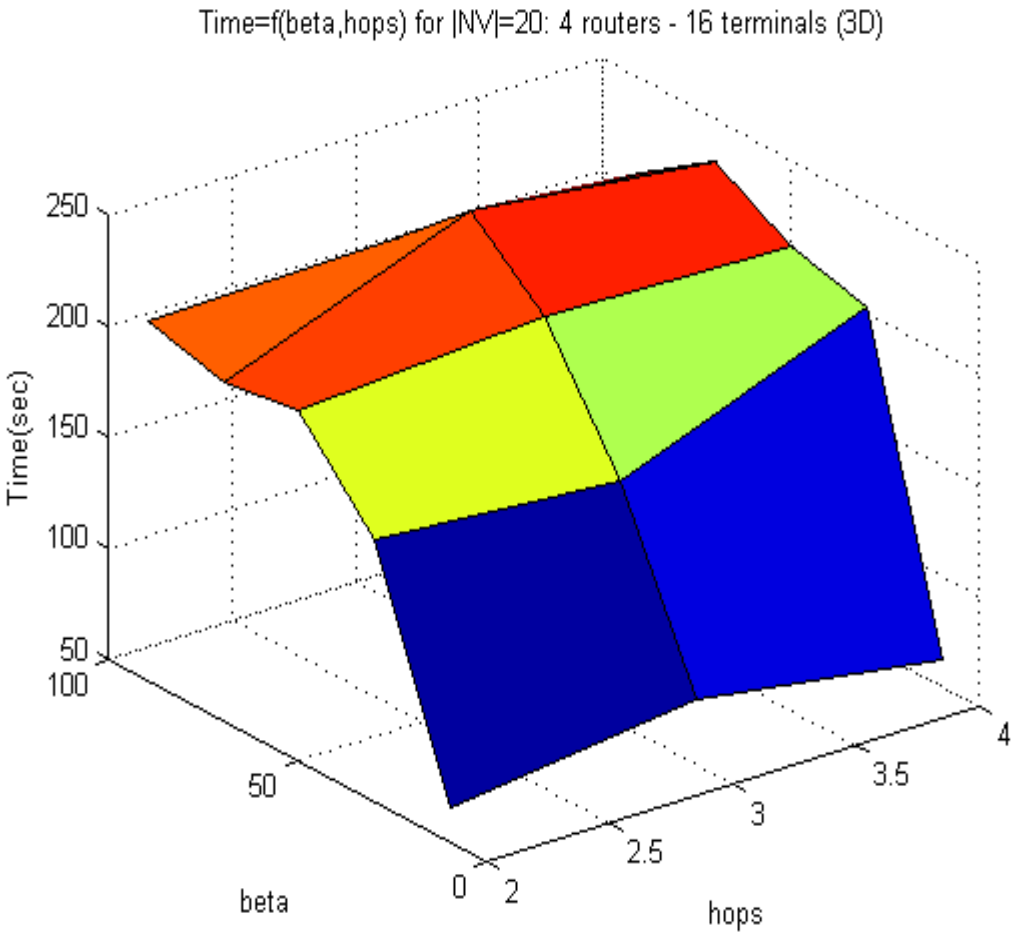
V.2.1.3 *Time (sec) για διάφορες τιμές μεγεθών εικονικών δικτύων,β,hops*



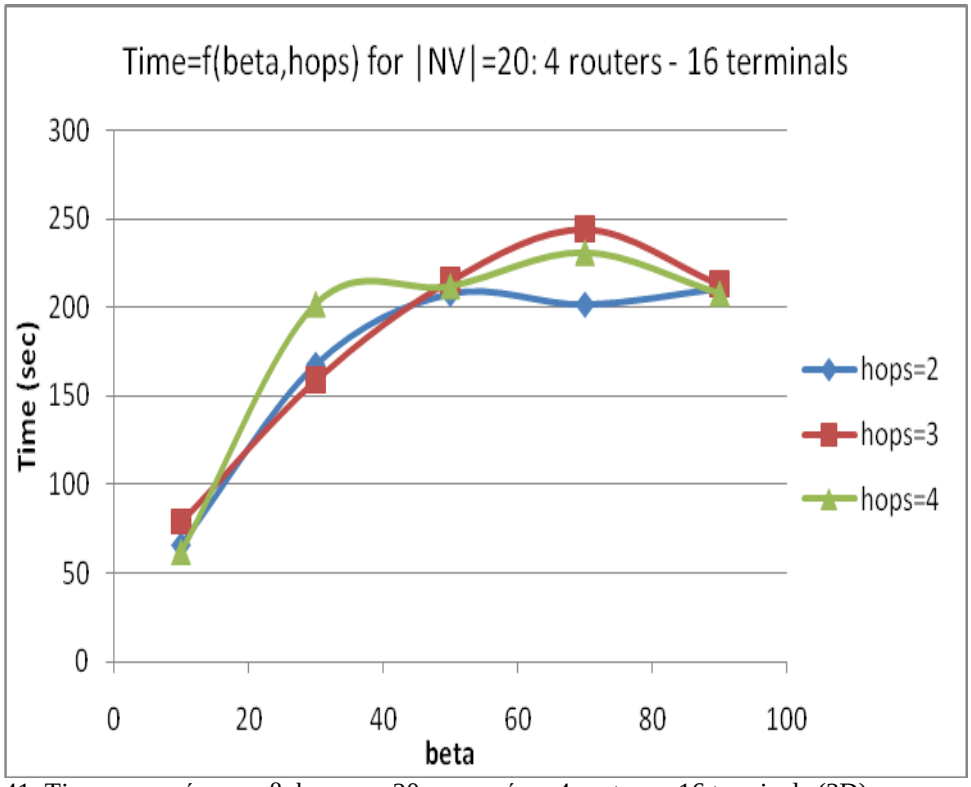
Σχήμα 38: Time ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (3D)



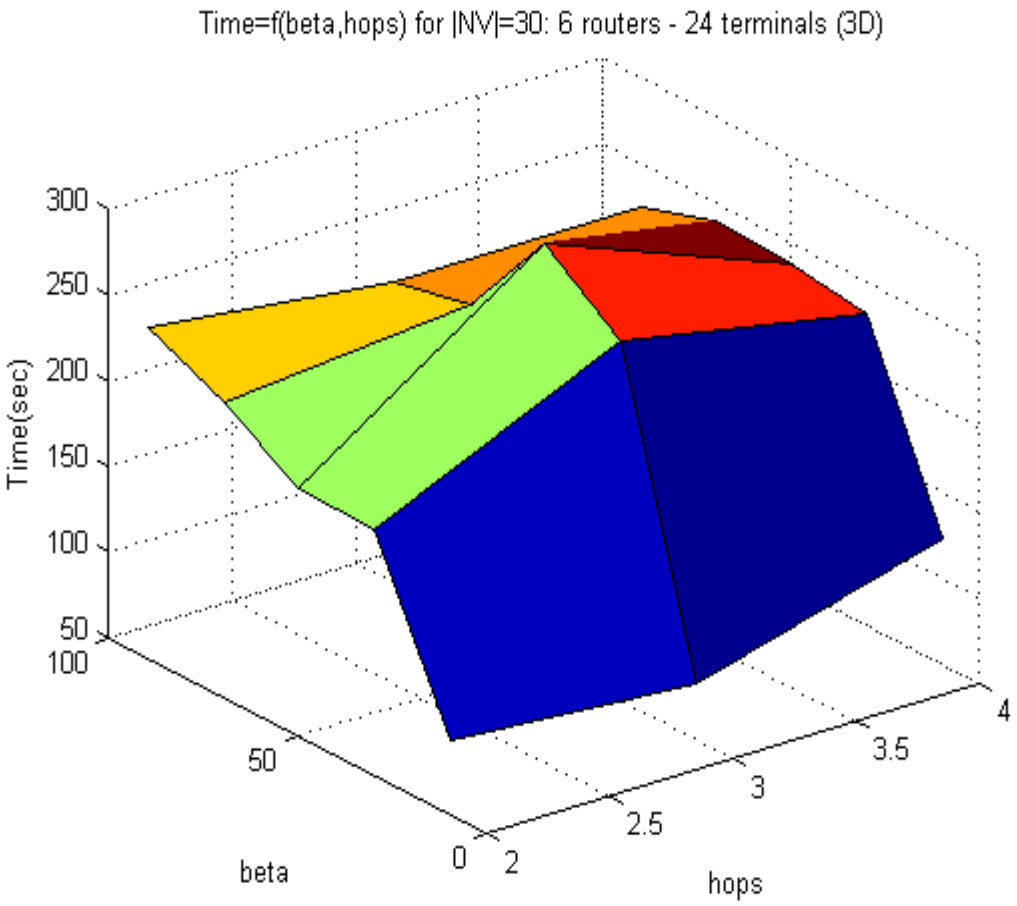
Σχήμα 39: Time ως συνάρτηση β, hops για 20 εικονικά με 2 routers – 8 terminals (2D)



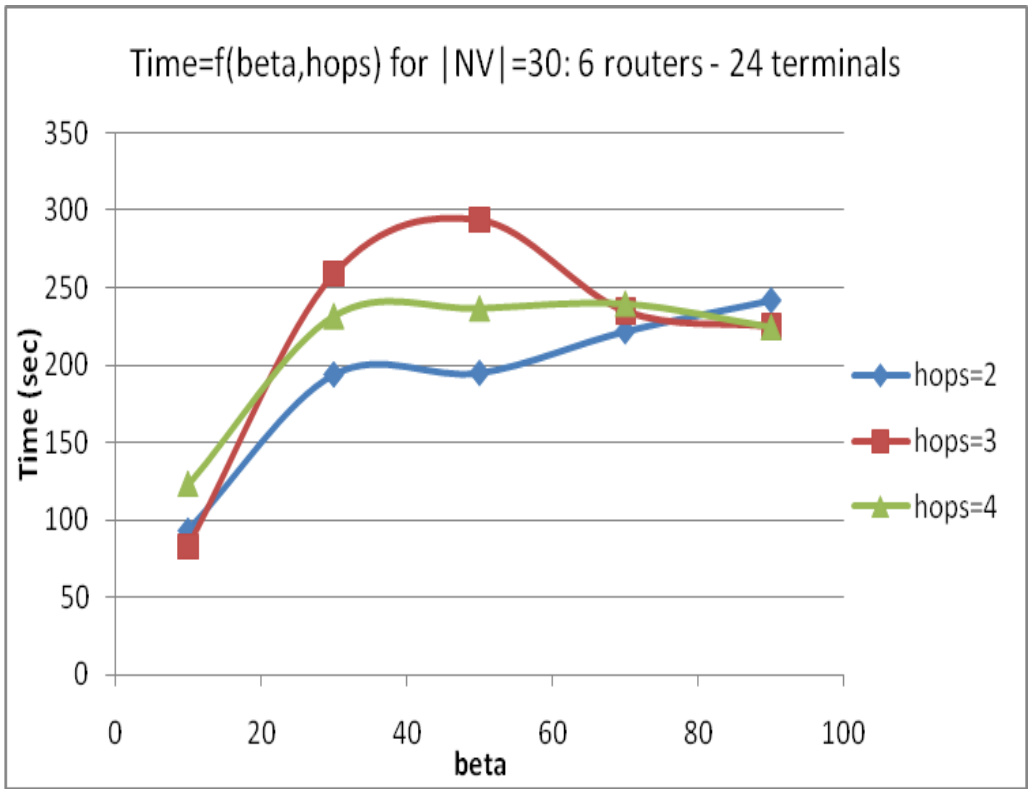
Σχήμα 40: Time ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (3D)



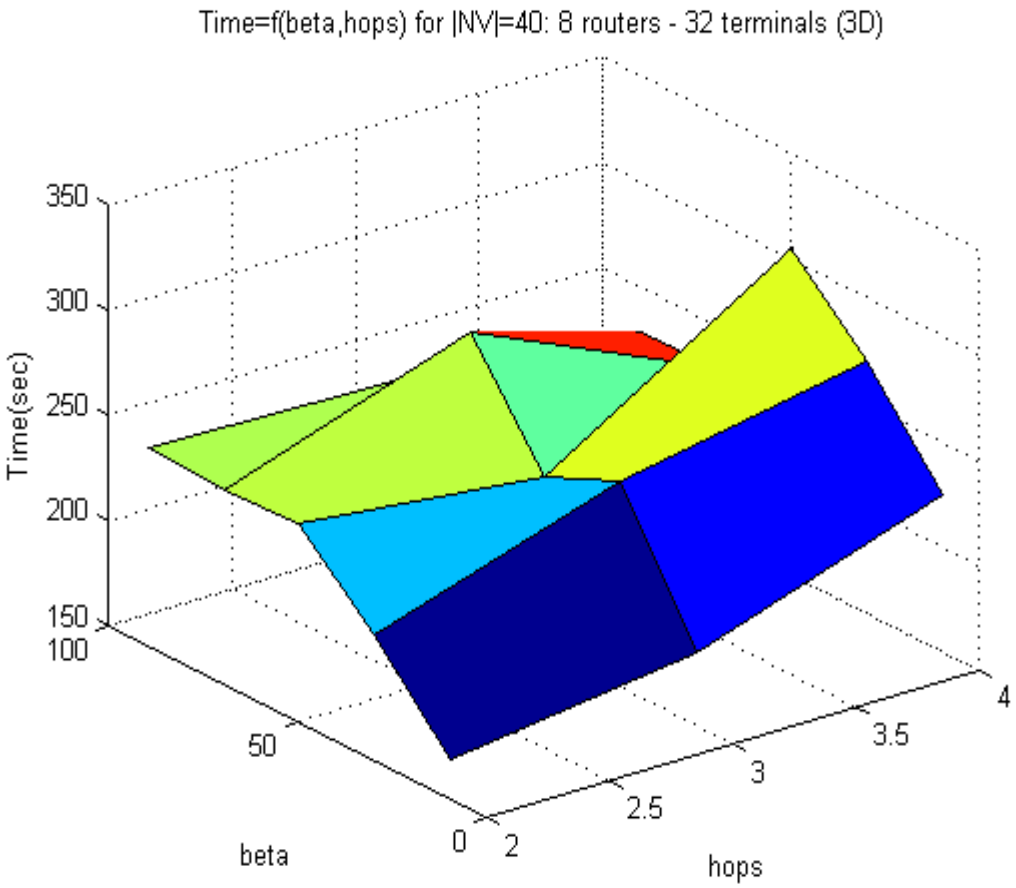
Σχήμα 41: Time ως συνάρτηση β, hops για 20 εικονικά με 4 routers – 16 terminals (2D)



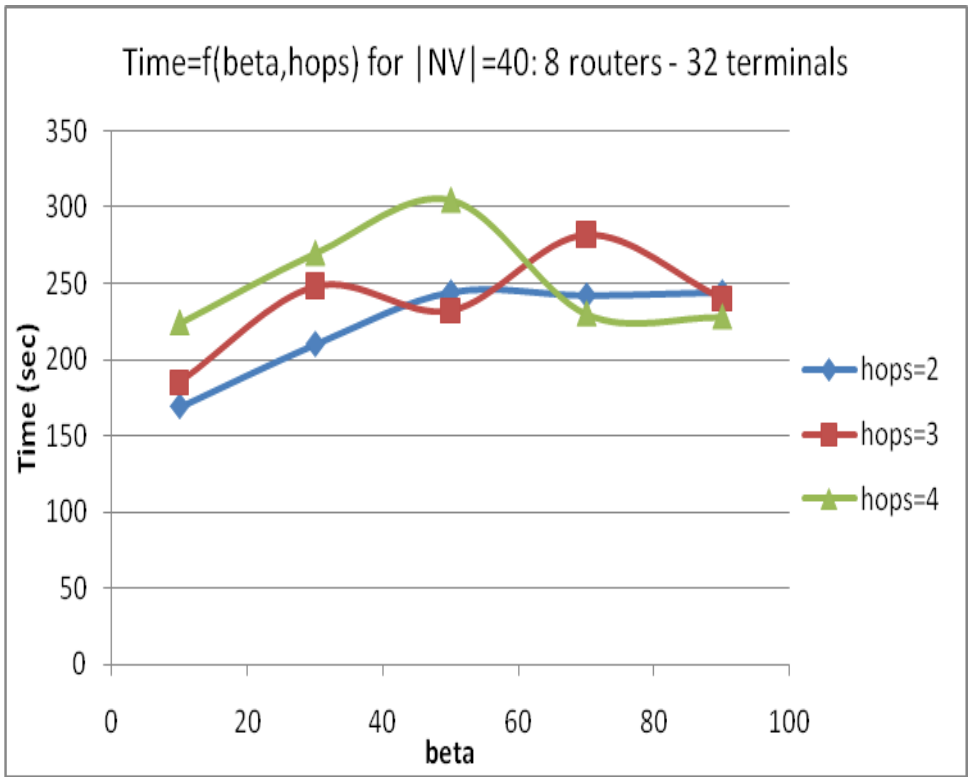
Σχήμα 42: Time ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (3D)



Σχήμα 43: Time ως συνάρτηση β, hops για 20 εικονικά με 6 routers – 24 terminals (2D)

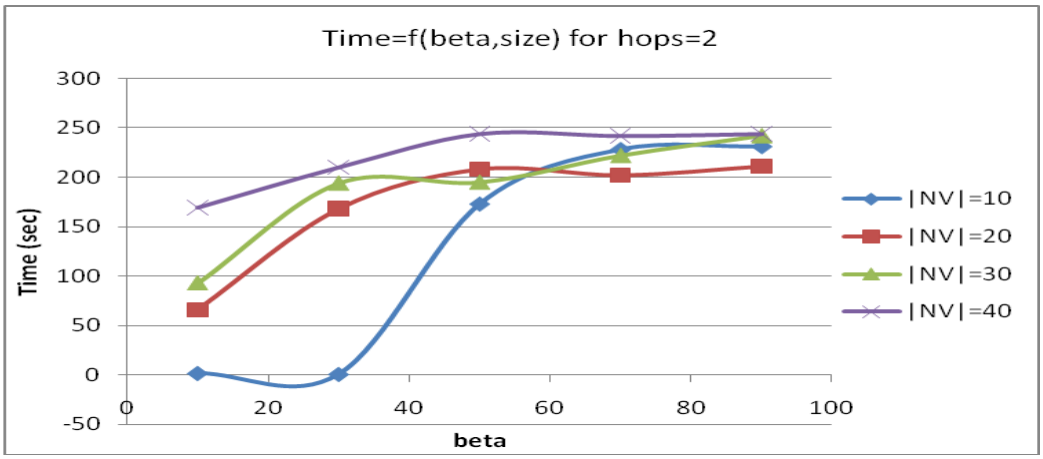


Σχήμα 44: Time ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (3D)

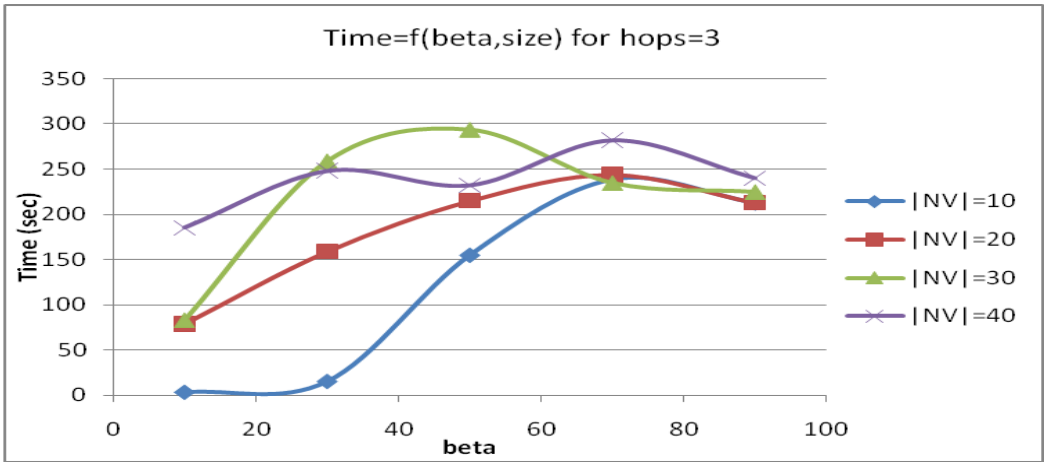


Σχήμα 45: Time ως συνάρτηση β, hops για 20 εικονικά με 8 routers – 32 terminals (2D)

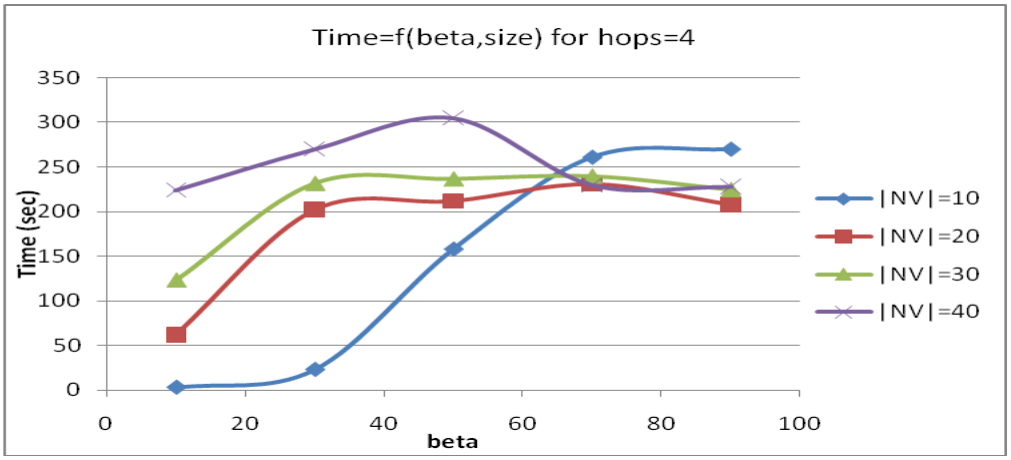
Συγκεντρωτικά Time για διάφορες τιμές hops



Σχήμα 46: Time συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=2 (2D)



Σχήμα 47: Time συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=3 (2D)



Σχήμα 48: Time συγκεντρωτικά για διάφορα μεγέθη εικονικών και β με hops=4 (2D)

V.2.2 *Μέρος Β: αποτελέσματα R/C (%), AR (%), Time (sec) για ετερογενές μείγμα 100 αιτήσεων (διαφορετικά μεγέθη, β, hops)*

R/C = 13.5 %
AR = 21 %
Time = 631 sec

V.3 *Τελικά συμπεράσματα*

1) Ο συγχρονισμός των φάσεων ανάθεσης κόμβων και ζεύξεων οδηγεί σε υψηλή ταχύτητα λειτουργίας, μεγάλο ποσοστό αποδεκτότητας και χαμηλά κόστη για τον πάροχο υπηρεσίας για ποικίλες τοπολογίες και απαιτήσεις εικονικών δικτύων.

2) Ο αλγόριθμος επωφελείται από την ύπαρξη μεγάλων σε μέγεθος υποστρωμάτων εκμεταλλευόμενος ικανοποιητικά τους πόρους τους και παράγοντας γρήγορα αποδοτικές χαρτογραφήσεις ακόμα και για μεγάλα εικονικά δίκτυα υψηλών απαιτήσεων (μέχρι κάποιο όριο φυσικά).

3) Η χρήση αλγορίθμων συντομότερου μονοπατιού (Dijkstra) αυξάνει το bandwidth που συνολικά απομένει στο υπόστρωμα προς χρήση και μειώνει το συνολικό συσσωρευτικό κόστος που αφορά το bandwidth των φυσικών μονοπατιών, καθώς αυτά έχουν το μικρότερο δυνατό μήκος σε hops.

4) Παρόλο που δεν παρέχεται δυνατότητα διαχωρισμού ή μετανάστευσης μονοπατιού στον βασικό αλγόριθμο (χρήση single-path ⇔ unsplittable flow λογικής), εντούτοις αυτός επωφελείται από τα χαμηλότερα κόστη που επιτυγχάνει λόγω της ιδιότητας (3) και αποφέρει υψηλό ποσοστό αποδεκτότητας. Επιπλέον μπορεί μελλοντικά να χρησιμοποιηθεί ο επιπλέον προτεινόμενος αλγόριθμος IV.5.1 για ακόμα μεγαλύτερο ποσοστό αποδεκτότητας.

5) Η αποκοπή ακατάλληλων ζευγών αντιστοιχίσεων κόμβων με χρήση έξυπνων heuristics εξαρχής στον αλγόριθμο βελτιώνει σημαντικά το χρόνο εκτέλεσης του αλγορίθμου μειώνοντας δραστικά την περιοχή αναζήτησης του προβλήματος.

6) Οι βελτιστοποιήσεις που έχουν γίνει στον αλγόριθμο μειώνουν δραστικά το χώρο αναζήτησής του και οδηγούν σε γρηγορότερους χρόνους λειτουργίας και σε χαμηλές απαιτήσεις μνήμης κατά την εκτέλεση.

7) Η ταξινόμηση των κατάλληλων ζευγών (που περνάνε από το βήμα βελτιστοποίησης) σύμφωνα με την μεγαλύτερη σταθμισμένη εναπομείνασα χωρητικότητα πόρων των φυσικών κόμβων καθώς και τις απαιτήσεις των εικονικών κόμβων που περιέχονται σε αυτά, οδηγεί σε αποδοτικότερες χαρτογραφήσεις και μεγιστοποιεί την συνολική εναπομείνασα χωρητικότητα του υποστρώματος. Η λογική ταξινόμησης είναι άπληστη (greedy).

8) Όπως ήταν αναμενόμενο, παρατηρήθηκε ότι εικονικά δίκτυα με μεγάλο αριθμό κόμβων και υψηλή τιμή του β είναι δυνητικά δυσκολότερο να απεικονιστούν από αντίστοιχα μικρότερα σε μέγεθος και λιγότερο απαιτητικά σε πόρους δίκτυα (τα οποία χαρτογραφούνται και γρηγορότερα). Δηλ. κατά μέσο όρο καθώς περνάμε από «ελαφρά», μικρά εικονικά σε μεγαλύτερα και «βαρύτερα» μειώνεται το R/C, μειώνεται το AR και αυξάνεται ο χρόνος εκτέλεσης.

9) Προσοχή πρέπει να δοθεί στο φαινόμενο που παρατηρείται όσον αφορά την απαίτηση max_hops: εικονικά δίκτυα που είναι πιο απαιτητικά ως προς την καθυστέρηση των φυσικών μονοπατιών απεικόνισης σε max_hops οδηγούν σε αποδοτικότερες χαρτογραφήσεις (μεγαλύτερο R/C), υψηλότερο ποσοστό αποδεκτότητας και γρηγορότερο χρόνο εκτέλεσης από ό,τι χαλαρότερα σε απαιτήσεις καθυστέρησης δίκτυα. Για κάθε συνδυασμό παραμέτρων μέγεθος-βήτα υπάρχει συγκεκριμένη τιμή max_hops που επιτυγχάνει υψηλό R/C, Acceptance ratio και σχετικά χαμηλό χρόνο εκτέλεσης. Συνήθως είναι η μικρή τιμή 2. Αυτό δεν είναι καθόλου παράλογο, και αιτιολογείται θεωρητικά ως εξής: αυστηρότερη απαίτηση max_hops → αναζήτηση και εύρεση μικρότερου μήκους φυσικών μονοπατιών απεικόνισης των εικονικών ζεύξεων → μικρότερη συνολική σπατάλη σε BW + μικρότερο search-space αλγορίθμου. Δηλ. ανάλογες αιτήσεις οδηγούν σε αύξηση της εναπομείνασας χωρητικότητας του υποστρώματος όσον αφορά το BW (αποδοτικότερες χαρτογραφήσεις αφού βελτιώνουν την αξιοποίηση των πόρων του υποστρώματος).
Αυτό φυσικά ισχύει με κάποιο όριο ανοχής: αν επιλεγούν πολύ περιοριστικές τιμές (π.χ. 1) μπορεί να αυξηθεί σημαντικά το ποσοστό των απορριφθέντων αιτήσεων εικονικών δικτύων λόγω του προβλήματος μη ύπαρξης ισομορφισμών που αναφέρθηκε εξαρχής. Από την άλλη πλευρά υψηλές τιμές max_hops μπορεί διαισθητικά να φαίνονται ότι οδηγούν σε υψηλότερο Acceptance Ratio αλλά στην πράξη όσο αυξάνονται, αυξάνεται και η σπατάλη BW των φυσικών ζεύξεων (προτίμηση μακρύτερων μονοπατιών) και το search-space του αλγορίθμου και άρα παρατηρείται χαμηλό R/C και AR και αύξηση του χρόνου εκτέλεσης. Το φαινόμενο της υπερίσχυσης των απαιτητικών σε hops αιτήσεων μειώνεται αισθητά για αυξημένα σε απαιτήσεις πόρων εικονικά (μεγάλο μέγεθος και β) καθώς στις περιπτώσεις αυτές η εύρεση μικρών σε μήκος φυσικών μονοπατιών με υψηλό bottleneck BW καθίσταται πολύ δύσκολη. Τότε πρέπει να προτιμούνται πιο χαλαρές σε απαιτήσεις καθυστέρησης αιτήσεις.

10) Οι υψηλές απαιτήσεις μπορούν να δράσουν θετικά σε ορισμένες (ακραίες) περιπτώσεις μειώνοντας το search-space του αλγορίθμου και οδηγώντας σε χαμηλότερους χρόνους εκτέλεσης.

11) Οι χρόνοι εκτέλεσης ανά αίτηση (μπορούν να βρεθούν ως μέσοι όροι διαιρώντας με το εκάστοτε πλήθος αιτήσεων) παρουσιάζονται σε πρώτη φάση να είναι αυξημένοι σε σχέση με το πρωτότυπο [1] των Γερμανών επιστημόνων. Αυτό οφείλεται στο πολύ σημαντικό constraint που έχει εισαχθεί σε αυτήν την εργασία σχετικά με την συμφωνία στον τύπο φυσικού-εικονικού κόμβου (τερματικό→τερματικό, router→router). Ο συγκεκριμένος περιορισμός δυσκολεύει αρκετά τον αλγόριθμο καθώς πρέπει να δομηθούν εικονικά δίκτυα που παρουσιάζουν διαφοροποίηση στον τύπο των κόμβων τους, χαρακτηριστικό που καθορίζει την μορφή της τοπολογίας τους και την καθιστά περισσότερο απαιτητική στην απεικόνιση.

12) Επιπλέον, ο μέσος χρόνος εκτέλεσης των προσομοιώσεων του μέρους Α (20 αιτήσεις τη φορά) κυμάνθηκε από 2 sec (για πολύ μικρά και χαλαρά σε απαιτήσεις εικονικά δίκτυα) μέχρι και 300 sec (για μεγάλα και «βαριά» εικονικά). Ως μεγέθη είναι σχετικά λογικά για προσομοιώσεις τέτοιας κλίμακας και απαιτήσεων καθώς ο αλγόριθμος που χρησιμοποιήθηκε είναι ευριστικός. Δηλ. βασικός περιορισμός είναι η καθυστέρηση απόκρισης μη ύπαρξης λύσης σε κάποια προβλήματα χαρτογράφησης που παρουσιάστηκαν κατά την προσομοίωση, καθώς ο αλγόριθμος είναι ευριστικός-αναδρομικός και το πρόβλημα NP-hard. Άρα η απόφαση μη αποκρισιμότητας μπορεί θεωρητικά να χρειαστεί εκθετικό χρόνο (σε σχέση με το μέγεθος και τις απαιτήσεις της εισόδου που είναι τα εικονικά δίκτυα προς χαρτογράφηση) σε ένα υπολογιστικό σύστημα. Για το λόγο αυτό τέθηκε ένα άνω όριο στον αριθμό των κλήσεων της βασικής αναδρομικής συνάρτησης του αλγορίθμου (όπως και στο []), του οποίου η υπέρβαση οδηγεί στο χαρακτηρισμό της χαρτογράφησης ως αποτυχημένης. Το όριο αυτό αναφέρεται ακριβώς ως συνάρτηση της εισόδου (τετραγωνική εξάρτηση) στην ενότητα IV.4.1. Αυτή η δυσμενής περίπτωση επίτευξης του ορίου παρουσιάζεται συχνά όταν υπάρχει πολύ υψηλός φόρτος εικονικών αιτήσεων (τόσο σε αριθμό όσο και σε απαιτήσεις λειτουργίας) και το υπόστρωμα βρίσκεται πολύ κοντά στα όρια της χωρητικότητάς του. Αυτό βέβαια είναι λογικό λόγω της ευριστικής φύσης του αλγορίθμου, οπότε στις περιπτώσεις αυτές θα ήταν πράγματι καλύτερο να χρησιμοποιηθούν προηγμένοι αλγόριθμοι MCF για την όσο το δυνατόν αποδοτικότερη αξιοποίηση του υποστρώματος.

13) Τέλος, γίνεται αντιληπτός και ένας επιπλέον περιορισμός του ευριστικού αλγορίθμου που χρησιμοποιήθηκε: επιλέγει με άπληστο τρόπο τις χαρτογραφήσεις των κόμβων (λογικός ως τρόπος σκέψης) αλλά διενεργεί μία καθαρά best-effort επιλογή των χαρτογραφήσεων των εικονικών ζεύξεων σε φυσικά μονοπάτια (πρόβλημα). Δηλ. δεν επιλέγει τα πιο «πλατιά» (με μεγαλύτερο BW) φυσικά μονοπάτια, αλλά τα πιο σύντομα που ικανοποιούν και την απαίτηση max_hops και την απαίτηση BW. Αυτό είναι από μία άποψη ορθό διότι μικρά σε μήκος φυσικά μονοπάτια ισοδυναμούν με λιγότερη σπατάλη σε BW για το υπόστρωμα, αλλά δεν εξασφαλίζει ότι η επιλογή αυτή των μονοπατιών δεν θα οδηγήσει σε bottlenecks. Δηλ. υπάρχει περίπτωση να επιλεγούν «κακές» φυσικές γραμμές από πλευράς BW οι οποίες μελλοντικά δεν θα μπορούν να αξιοποιηθούν από καταλληλότερα εικονικά δίκτυα. Φυσικά το πρόβλημα maximum BW- minimum delay είναι δύσκολο καθώς μπορεί να μην έχει καν λύση, συνεπώς μόνο ευριστικές προσεγγίσεις μπορούν να εφαρμοστούν όπως συμβαίνει και σε αυτήν την περίπτωση.

14) Γενικά επιβεβαιώθηκαν πολλές από τις παρατηρήσεις του πρωτότυπου [1], αλλά το σημαντικότερο είναι ότι προέκυψαν πολλά επιπλέον χρήσιμα συμπεράσματα λόγω της εφαρμογής του αλγορίθμου σε τοπολογίες terminals-routers που βρίσκονται πολύ κοντά στη μορφή των πραγματικών IP δικτύων (routers-gateways, terminals, delay constraint). Ακριβώς εξαιτίας αυτού του ιδιαίτερου χαρακτηριστικού δεν υπήρχε η δυνατότητα να γίνει σύγκριση με κάποια παρελθούσα έρευνα, καθώς συνήθως χρησιμοποιούνται τοπολογίες όπου όλοι οι κόμβοι θεωρούνται ισοδύναμοι ως προς τον τύπο.

Το σύστημα που υλοποιήθηκε μπορεί να θεωρηθεί γρήγορο και αποδοτικό και αποδείχτηκε ότι μπορεί να χειρίζεται με σχετική ευκολία και σε πραγματικό χρόνο δυναμικά φορτία μεσαίου μεγέθους και απαιτήσεων. Μάλιστα, σε περίπτωση ύπαρξης εφικτής λύσης ενσωμάτωσης για κάθε εικονικό δίκτυο, αποκρίνεται πολύ γρήγορα βρίσκοντας μία αποδεκτή λύση ενώ έχει ως άνω όριο εκτέλεσης τετραγωνικό ως προς την είσοδο χρόνο για αντιμετώπιση της NP-hardness του προβλήματος.

VI. Ανασκόπηση – μελλοντική έρευνα

VI.1 *Ανασκόπηση – Συνεισφορά της εργασίας*

Στην εργασία αυτή χρησιμοποιήθηκε ως βάση και τροποποιήθηκε ένας υπάρχων αλγόριθμος ενσωμάτωσης εικονικών δικτύων ο οποίος προγραμματίστηκε εξαρχής σε γλώσσα C. Ο αλγόριθμος στηρίζεται στην ευριστική ανίχνευση του ισομορφισμού μεταξύ υπογράφων και χαρτογραφεί τους κόμβους και τις ζεύξεις στην ίδια φάση επιτυγχάνοντας υψηλή τάξη συγχρονισμού της απεικόνισης, αποδοτικές χαρτογραφήσεις και πολύ μεγάλη ταχύτητα λειτουργίας. Επίσης πέραν της υλοποίησης του βασικού αλγορίθμου, προγραμματίστηκε και ένας τυχαιοκρατικός τρόπος παραγωγής γραφημάτων φυσικών και εικονικών δικτύων, προσδίδοντας στα στοιχεία τους (κόμβοι, ζεύξεις) ποικίλες απαιτήσεις λειτουργίας και περιορισμούς χωρητικότητας. Τα γραφήματα αυτά τροφοδοτούν ως είσοδοι τον αλγόριθμο και η μορφολογία τους είναι μέσα στα πλαίσια των σύγχρονων IP δικτύων τερματικών-δρομολογητών. Το βασικότερο είναι ότι προσομοιώνεται ουσιαστικά σε C και με βάση τον αλγόριθμο αυτό ένα γρήγορο σύστημα διαχείρισης και εξυπηρέτησης πολλαπλών, δυναμικά αφικνούμενων αιτήσεων εικονικών δικτύων. Με την προσομοίωση εξάγονται χρήσιμα συμπεράσματα για την αποδεκτότητα και αποδοτικότητα των χαρτογραφήσεων που παράγει το σύστημα σε πραγματικό χρόνο υπό συνθήκες ποικίλου φόρτου και για ποικίλης κλίμακας προβλήματα ενσωμάτωσης. Η προσομοίωση στηρίζεται στην χρονοδρομολόγηση γεγονότων και είναι μία καινοτόμος λογική προσομοίωσης για ανάλογα προβλήματα δυναμικών αφίξεων ενσωμάτωσης.

Ακολουθεί μία συνοπτική ανασκόπηση της δόμησης της εργασίας:

Αρχικά στο μέρος Ι έγινε μία εισαγωγή στην τεχνική της εικονικοποίησης δικτύων, στα πλεονεκτήματά της, στα προβλήματα που αντιμετωπίζει, στην υλοποίησή της και στο βασικό θέμα της εργασίας. Ακολούθως στο μέρος ΙΙ, πραγματοποιήθηκε μία αναφορά σε ανάλογες εργασίες και δημοσιεύσεις που είναι σχετικές με το θέμα και στις μεθόδους και λύσεις οι οποίες προτάθηκαν σε αυτές. Κατόπιν στο μέρος ΙΙΙ δόθηκε μία μαθηματική περιγραφή του μοντέλου των δικτύων και του προβλήματος ενσωμάτωσης. Στο μέρος IV παρουσιάστηκε ο πλήρης αλγόριθμος που χρησιμοποιήθηκε για την επίλυση του προβλήματος χρησιμοποιώντας ευριστικές μεθόδους. Επίσης προτάθηκαν για εξέταση ένας θεωρητικός αλγόριθμος εύρεσης μονοπατιού για πολλαπλές ροές και δύο αλγόριθμοι για την θεωρητική ανάκαμψη από φυσικές αστοχίες κόμβων και ζεύξεων. Το μέρος V περιέχει τα στοιχεία που αφορούν το περιβάλλον προσομοίωσης του βασικού αλγορίθμου και συγκεντρωτικά τα αποτελέσματα της προσομοίωσης, ακολουθούμενα από σχόλια και συμπεράσματα σχετικά με αυτά. Τέλος στο μέρος VI (τρέχον) γίνεται μία συνολική ανασκόπηση της εργασίας και της συνεισφοράς της στην έρευνα γύρω από το θέμα της ενσωμάτωσης εικονικών δικτύων και διατυπώνονται κάποιες προτάσεις για μελλοντική έρευνα στον γενικότερο τομέα της εικονικοποίησης δικτύων.

Η εργασία αυτή δημιουργήθηκε με στόχο να αποτελέσει έναν ακόμη κρίκο στην ερευνητική αλυσίδα με σκοπό την επίτευξη μίας αποδοτικής και οικονομικής λύσης του γενικού προβλήματος ενσωμάτωσης εικονικών δικτύων. Η συνεισφορά της είναι η υλοποίηση σε κώδικα και διεξοδική μελέτη ενός πολλά υποσχόμενου ευριστικού αλγορίθμου εφαρμοζόμενου σε σύγχρονης μορφής και τοπολογίας δίκτυα IP. Κείται στα πλαίσια του γενικότερου τομέα της εικονικοποίησης δικτύων που προβλέπεται ότι θα αλλάξει ολοκληρωτικά το τοπίο του Internet στο μέλλον.

VI.2 *Προτάσεις για μελλοντική έρευνα*

Με βάση τα συμπεράσματα που εξήχθησαν από αυτή την εργασία μπορούν να διατυπωθούν οι ακόλουθες προτάσεις για μελλοντική έρευνα:

VI.2.1 *Προσομοίωση πρακτικά*

Παραμετροποίηση του προγράμματος προσομοίωσης:

- Εισαγωγή διαφορετικών τύπων κόμβων (π.χ. Vswitches, Vservers)
- Εισαγωγή διαφορετικών τύπων πόρων (π.χ. hard_disk memory, Vprocesses RAM, κτλ.)
- Αποδοτικότερη αναπαράσταση γραφημάτων στη μνήμη (όχι απλά ως bulk σύνολα κόμβων και ακμών, αλλά με τη χρήση λιστών γειτνίασης ή πινάκων πρόσπτωσης)
- Χρήση διαφορετικών μετρικών κόστους όσον αφορά την οικονομική αντιμετώπιση του προβλήματος
- Υλοποίηση διαφορετικών πολιτικών διαχείρισης των εικονικών αιτήσεων και των ουρών όπου αυτές καταχωρούνται (π.χ. χρήση προτεραιοτήτων, γήρανση, αποδοτική και γρήγορη διαχείριση ουρών και διευθέτηση αιτήσεων)
- Χρήση πολυνηματικών λογικών προσομοίωσης ώστε το σύστημα υποδοχής αιτήσεων και το σύστημα εξυπηρέτησής τους να λειτουργούν πραγματικά παράλληλα (π.χ. ανάθεση ενός νήματος ανά ουρά). Στην περίπτωση αυτή πρέπει να προσεχθούν ιδιαίτερα περιπτώσεις σύγκρουσης λόγω ανταγωνιστικής πρόσβασης στους κοινούς πόρους του υποστρώματος.
- Αλλαγή ή βελτίωση των μηχανισμών παραγωγής τοπολογιών τόσο για το υπόστρωμα όσο και για τα εικονικά δίκτυα, ώστε να καλύπτονται ποικίλες τοπολογίες διαφορετικών απαιτήσεων που συναντώνται στο Internet.
- Μεταβολή οποιασδήποτε παραμέτρου είναι απαραίτητο για την προσαρμογή στις ανάγκες κάθε προσομοίωσης (ρυθμός άφιξης αιτήσεων κτλ.).

VI.2.2 *vnmFlib*

Βελτίωση ορισμένων χαρακτηριστικών του αλγορίθμου vnmFlib:

- Βελτίωση φάσεων βελτιστοποίησης και ταξινόμησης ζευγαριών αντιστοιχίσεων εικονικός κόμβος – φυσικός κόμβος, π.χ. με χρήση γρηγορότερων και αποδοτικότερων αλγορίθμων ταξινόμησης.
- Μηχανισμοί παροχής παραπάνω μνήμης στη φάση υπαναχώρησης (backtracking) και διερεύνηση μηχανισμών ανίχνευσης της μελλοντικής πορείας και εξέλιξης του αλγορίθμου με βάση την παρελθούσα συμπεριφορά του, ώστε το backtracking να είναι πιο γρήγορο και αξιόπιστο=> απαιτούνται εξυπνότερα heuristics.

- Βελτίωση συναρτήσεων ελέγχου ισομορφισμού μεταξύ των εικονικών υπογράφων, ώστε ο έλεγχος να είναι γρηγορότερος και αποδοτικότερος.

- Προσπάθειας σχεδίασης και εφαρμογής πιο αποδοτικών αλγορίθμων κατά τη φάση εύρεσης και αντιστοίχισης φυσικών μονοπατιών σε εικονικές ζεύξεις. Συγκεκριμένα προτείνεται να εφαρμοστεί ο προτεινόμενος στο μέρος ΙΙΙ αλγόριθμος 1 (εύρεσης συνόλου μονοπατιών), για εξαγωγή συμπερασμάτων σχετικά με τη βελτίωση του ποσοστού αποδεκτότητας των αιτήσεων και την αποδοτικότερη αξιοποίηση των πόρων του υποστρώματος που αυτή επιτυγχάνει χωρίς την απαίτηση εφαρμογής πολύπλοκων και χρονοβόρων MCF αλγορίθμων.

VI.2.3 *Πολυπλοκότητα αλγορίθμου*

Υπολογισμός θεωρητικών φραγμάτων πολυπλοκότητας αλγορίθμου vnmFlib και καθορισμός χειρότερης και πιο χρονοβόρας περίπτωσης.
Επίσης, προτείνεται η μελέτη και ο θεωρητικός υπολογισμός της απόκλισης των αποτελεσμάτων του αλγορίθμου από τη βέλτιστη λύση, σε περίπτωση που αυτή δύναται να καθοριστεί (μόνο θεωρητικά βέβαια λόγω NP-hardship).

VI.2.4 *Ανάκαμψη από αστοχίες*

Πειραματική εφαρμογή και ενσωμάτωση του προτεινόμενου αλγορίθμου 2 (ανάκαμψης από αστοχία φυσικού κόμβου ή ζεύξης) που προτείνεται στο τέλος του μέρους IV.

VI.2.5 *Προσομοίωση θεωρητικά*

Υπολογισμός διαστημάτων εμπιστοσύνης για τα αποτελέσματα της event-driven προσομοίωσης του συστήματος διαχείρισης αιτήσεων.
Προτείνεται επίσης χρήση τεχνικών περιορισμού του μεταβατικού φαινομένου της προσομοίωσης ([25]) για την επιτυχή μελέτη του συστήματος στην κατάσταση ισορροπίας (μόνιμη κατάσταση), όπου η μεταβλητότητα των παρατηρήσεων είναι μικρή. Αυτό μπορεί να επιτευχθεί με εφαρμογή των ακόλουθων μεθόδων αποκοπής:
- με βάση το εύρος τιμών
- με βάση τη συνολική μέση τιμή
- με χρήση μετατοπιζόμενης μέσης τιμής (moving average)
- με βάση τμηματικές μέσες τιμές (batch means)

Για την εξασφάλιση της ανεξαρτησίας των παρατηρήσεων μπορούν επίσης να χρησιμοποιηθούν οι ακόλουθες τεχνικές:
- ανεξάρτητες επαναλήψεις (independent replications)
- τμηματικές μέσες τιμές (batch means)
- η αναγεννητική μέθοδος (regenerative method)

VI.2.6 *Ασφάλεια-απομόνωση*

Διερεύνηση ζητημάτων ασφάλειας και απομόνωσης κατά τη λειτουργία των εικονικών δικτύων, ώστε να εξασφαλίζεται η ιδιωτικότητα των πόρων και η απουσία παρεμβολών της λειτουργίας ενός εικονικού δικτύου στα υπόλοιπα.

VI.2.7 *Τοπολογίες με constraints θέσης κόμβων*

Μελέτη εικονικών δικτύων με συγκεκριμένες απαιτήσεις-περιορισμούς όσον αφορά την επιθυμητή θέση των φυσικών κόμβων όπου θα απεικονιστούν οι εικονικοί. Ο περιορισμός μπορεί να αφορά μία μέγιστη αποδεκτή απόσταση από κάποιον φυσικό κόμβο (τερματικό ή router). Το αν ικανοποιείται ή όχι μπορεί εύκολα να υπολογιστεί με χρήση του αλγορίθμου εύρεσης φυσικών μονοπατιών που περιγράφθηκε στο IV.4.2.2.1, χρησιμοποιώντας ως πηγή τον υποψήφιο φυσικό κόμβο και ως προορισμό τον επιθυμητό, θέτοντας απαίτηση BW=0 (παράκαμψη περιορισμού BW) και κρατώντας μόνο το delay constraint.
Έτσι στη συνάρτηση *valid()*(IV.4.2.2) μπορεί να εφαρμοστεί ο παραπάνω έλεγχος για τον προσδιορισμό της καταλληλότητας κάποιας απεικόνισης εικονικός κόμβος-φυσικός κόμβος. Ως επιπλέον constraint μπορεί να δοκιμαστεί η απαίτηση ενός εικονικού τερματικού να ενσωματωθεί στην φυσική ομάδα τερματικών που σχετίζεται με ένα συγκεκριμένο φυσικό router-gateway. Μέριμνα για το τελευταίο constraint έχει ληφθεί στον C κώδικα της εργασίας.

VI.2.8 *Διαχείριση VNRs*

Εξέταση μεθόδων διαχείρισης των εικονικών αιτήσεων:

- CAC (Call Admission Control) => έλεγχος αποδεκτότητας εισερχόμενων κλήσεων-αιτήσεων και υπολογισμός πιθανότητας απόρριψής τους

- CDP (Call Drop Probability) => υπολογισμός απόρριψης ήδη εξυπηρετούμενης αίτησης λόγω φυσικού προβλήματος (αστοχία) ή πολιτικής προτεραιοτήτων

- Επόπτευση απόδοσης λειτουργίας (ικανοποιούνται διαρκώς όλες οι απαιτήσεις των αιτήσεων και αν ναι, πόσο αποδοτικά?)

- Επίβλεψη συμμόρφωσης με κανόνες (αλγόριθμοι αστυνόμευσης όπως ο αλγόριθμος διαρρέοντος δοχείου για την κίνηση των εικονικών πακέτων)

- Χρονοπρογραμματισμός και scheduling των φυσικών πόρων για αποδοτικό διαμοιρασμό τους στα ετερογενή εικονικά δίκτυα

- Ενημέρωση, διαχείριση και αντιμετώπιση εκτάκτων καταστάσεων που επηρεάζουν είτε το υπόστρωμα(π.χ. πτώση φυσικού κόμβου ή ζεύξης) είτε κάποιο/α εκ των εικονικών δικτύων
  (π.χ. εισβολή ιού ή εκδήλωση προγραμματιστικού bug), με το μικρότερο δυνατό κόστος για τα απεικονισθέντα εικονικά δίκτυα.

VI.2.9 *FCAPS*

Γενικά ζητήματα management με βάση την λογική FCAPS (Fault-Configuration-Accounting-Performance-Security), προσαρμοσμένη όμως στις απαιτήσεις της λειτουργίας των εικονικών δικτύων. Αυτό είναι ένα πολύ γενικό θέμα το οποίο χρήζει μεγάλων ερευνητικών επενδύσεων.

ΠΑΡΑΡΤΗΜΑ- ΚΩΔΙΚΑΣ ΕΡΓΑΣΙΑΣ

Λίγα λόγια για τον κώδικα

➔Ο κώδικας γράφτηκε σε γλώσσα ANSI C ([27]) σε λειτουργικό Windows Vista x86 με χρήση του IDE Dev-C++ της Bloodshed, version 4.9.9.2 (downloadable at http://www.bloodshed.net).

➔Με χρήση του κώδικα μπορούν να παραχθούν αρχεία .gv.txt τα οποία κατόπιν υπόκεινται σε επεξεργασία από τον GVedit Grpahviz Editor for Windows, version 0.99 beta (downloadable at http://www.graphviz.org/). Παράγονται με τον τρόπο αυτό γραφήματα του φυσικού υποστρώματος, των εικονικών γράφων, των απεικονίσεων κόμβων και ακμών κτλ. Τα γραφήματα αυτά χρησιμεύουν για την οπτική αναπαράσταση των αποτελεσμάτων του αλγορίθμου και των αντικειμένων που αυτός διαχειρίζεται.

➔Το βασικό αρχείο main_sim.c μπορεί να μεταγλωττιστεί και να τρέξει σε οποιοδήποτε λειτουργικό σύστημα ενώ δεν απαιτείται η ύπαρξη αρχείου Makefile λόγω των ορθά τοποθετημένων εντολών προεπεξεργαστή #include σε κάθε αρχείο. Έτσι μεταγλώττιση του main_sim.c προκαλεί αλυσιδωτή μεταγλώττιση όλων των υπολοίπων αρχείων συναρτήσεων και έχει ως συνέπεια την δημιουργία ενός μόνο εκτελέσιμου που τρέχει παντού. Προσοχή: στα linux το compilation απαιτεί τη σημαία –std=c99 και το linking τη σημαία –lm ώστε να λειτουργήσει το πρόγραμμα αποτελεσματικά.

➔Έχει δοθεί πολύ μεγάλη μέριμνα στην αποδοτική διαχείριση της μνήμης και των υπολογιστικών πόρων που απαιτεί η εκτέλεση του προγράμματος σε πολυάριθμες περιπτώσεις , δηλ. το πρόγραμμα-σύστημα που υλοποιήθηκε είναι ελαφρύ, γρήγορο και αποδοτικό.

➔Επίσης δόθηκε μεγάλη προσοχή στον προσεκτικό σχολιασμό του κώδικα και στην υιοθέτηση των χαρακτηριστικών της συντηρησιμότητας και της εύκολης ανάγνωσης αυτού. Επίσης έγινε προσπάθεια το πρόγραμμα να είναι φιλικό και οικείο στο χρήστη.

➔Ο κώδικας ικανοποιεί πλήρως όλες τις θεωρητικές προϋποθέσεις και περιγραφές που αναλύθηκαν στην εργασία.

➔Παρακάτω δίνονται συνοπτικά όλα τα στοιχεία που συναποτελούν τον κώδικα του συστήματος και της προσομοίωσης που υλοποιήθηκε:

- Η βιβλιοθήκη αντικειμένων που περιέχει όλα τα απαραίτητα αντικείμενα-δομές, αρχείο: objects_lib.h
- Οι συναρτήσεις παραγωγής γράφων και γραφικών απεικόνισης, αρχείο: graph_funs.c
- Οι βασικές συναρτήσεις του αλγορίθμου vnmFlib, αρχείο: basic_vnmFlib_funs.c
- Οι βοηθητικές συναρτήσεις του αλγορίθμου vnmFlib, αρχείο: aux_vnmFlib_funs.c
- Οι συναρτήσεις προσομοίωσης, αρχείο: simulation_funs.c
- Η κύρια συνάρτηση προγράμματος, αρχείο: main_sim.c

➔Τα παραπάνω βασικά στοιχεία του κώδικα παρουσιάζονται αναλυτικά στις ακόλουθες ενότητες. **Ο κώδικας περιλαμβάνεται και στο συνοδευτικό CD.**

Βιβλιοθήκη αντικειμένων που χρησιμοποιήθηκαν objects_lib.h

```
//-------------------------------------------------------------------------------
//Substrate Network

//Substrate Node
struct SubstrateNode_struct {
	int id;
	int type;	//0=>terminal, 1=>router
	float res_CPU;
	int res_slice_num;
	int group_GW_id;
	int assigned; //0=>no, 1=>yes
	int sliced;	//aux variable for slices doublecounting avoidance
};
typedef struct SubstrateNode_struct* SubstrateNode;

//Substrate Link
struct SubstrateLink_struct {
	int id;
	SubstrateNode end1;
	SubstrateNode end2;
	float res_BW;
};
typedef struct SubstrateLink_struct* SubstrateLink;

//Substrate Graph
struct SubstrateGraph_struct {
	int term_num, router_num, node_num, link_num, id;
	SubstrateNode* Substrate_N;  //set of nodes
	SubstrateLink* Substrate_L;  //set of links
};
typedef struct SubstrateGraph_struct* SubstrateGraph;

//Substrate Path-flow
struct SubstratePath_struct {
	int path_length;
	SubstrateNode* path_nodes;	//set of node
	SubstrateLink* path_links;	//set of links
};
typedef struct SubstratePath_struct* SubstratePath;

//-------------------------------------------------------------------------------
//Virtual Network

//Virtual Node
typedef struct VNNode_struct* VNNode;
struct VNNode_struct {
	int id;
	int type;		//0=>terminal, 1=>router
	float CPU_req;
	VNNode next_vrouter;	//if virtual terminal
	SubstrateNode SNode_req; //optional
	int req_group_GW_id;	//optional
};

//Virtual Link
struct VNLink_struct {
	int id;
	VNNode end1;
	VNNode end2;
	float BW_req;
	int max_hops;
};
typedef struct VNLink_struct* VNLink;

//Virtual Graph
struct VNGraph_struct {
	int term_num, router_num, node_num, link_num, id;
	VNNode* VN_N;  //set of nodes
	VNLink* VN_L;  //set of links
};
typedef struct VNGraph_struct* VNGraph;
```

```
//Virtual Neighbours
struct VNNeighbours_struct {
	int num;
	VNNode* VNNeighbours_array; //set of nodes
};
typedef struct VNNeighbours_struct* VNNeighbours;

//-------------------------------------------------------------------------------
//Virtual Network mapping
struct VNMapping_struct {
	int id;
	int node_assign_num;
	SubstrateNode* node_assign;
	int link_assign_num;
	SubstratePath* link_assign;
	int complete;
};
typedef struct VNMapping_struct* VNMapping;

//-------------------------------------------------------------------------------
//Simulation

typedef struct event_struct* eventptr;

//Virtual Network Request
typedef struct VNRequest_struct* VNRequest;
struct VNRequest_struct {
	int id;
	VNRequest previous;
	VNRequest next;
	VNGraph VNG;
	VNMapping VNM;
	float arrival_time;
	float deployment_time;
	float duration;
	eventptr event;
	int mapped;
	float R_dia_C;
};

//Event
struct event_struct {
	float time;
	VNRequest VNR;
	eventptr previous;
	eventptr next;
	int eventtype;  //0->arrival, 1->to_serve_queue examination, 2->switch queues,
			//3->expiration before mapping(abnormal), 4->expiration after mapping(normal)
};

//Virtual Network Request Queue
struct VNR_queue_struct {
	int length;
	VNRequest head;
	VNRequest tail;
};
typedef struct VNR_queue_struct* VNR_queueptr;

//Event Queue
struct event_queue_struct {
	int length;
	eventptr head;
	eventptr tail;
};
typedef struct event_queue_struct* event_queue_ptr;

//-------------------------------------------------------------------------------
//auxiliary objects

//F
struct F_type_struct {
	int length;
	VNNode* F_mat;
};
typedef struct F_type_struct* F_type;
```

```
//assignment pair nV-nP
struct pair_struct {
	VNNode nV;
	SubstrateNode nP;
};
typedef struct pair_struct* pair;

//set of assignment pairs
struct set_of_pairs_struct {
	int pair_num;
	pair* sop;
};
typedef struct set_of_pairs_struct* set_of_pairs;

//return object of valid function
struct valid_return_struct {
	VNNeighbours VNNeighb;		//Vneighbours of Vnode in GsubV
	VNLink* virtual_links;		//virtual links to be assigned
	SubstratePath* physical_path;  //substrate paths corresponding to vlinks
};
typedef struct valid_return_struct* valid_return;

//-------------------------------------------------------------------------------
```

Συναρτήσεις παραγωγής γράφων και γραφικών απεικόνισης graph_funs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "objects_lib.h"

//function prototypes

//randomization
float u_random(void);
float u_space_random(float a, float b);

//creation
SubstrateGraph new_SubstrateGraph(int id, int router_num, int max_link_num, int term_num, float max_bw, float term_max_cpu, int term_max_slice, int router_max_slice);
VNGraph new_VNGraph(int id, int router_num, int term_num, int max_link_num, float max_bw, float term_max_cpu, int max_hops);
VNGraph new_GsubV(VNGraph GV);
VNMapping new_VNM(VNGraph GV, int req_id);

//printing
void print_GP(SubstrateGraph GP, FILE* fp);
void print_GV(VNGraph GV, FILE* fp);
void print_VNM(VNMapping VNM, VNGraph GV, FILE* fpn, FILE* fpl);

//destruction
void destroy_GP(SubstrateGraph GP);
void destroy_GV(VNGraph GV);
void destroy_GsubV(VNGraph GsubV);
void destroy_VNM(VNMapping VNM, VNGraph GV);

//abnormal termination
void leave(void);


//create a new substrate graph (terminal stars around routers, arbitrary router-router connection)
SubstrateGraph new_SubstrateGraph(int id, int router_num, int term_num, int max_link_num, float max_bw, float term_max_cpu, int term_max_slice, int router_max_slice)
{
   SubstrateGraph GP=malloc(sizeof(struct SubstrateGraph_struct));
   if (GP==NULL)
      return NULL;
   GP->id=id;
   GP->term_num=term_num;
   GP->router_num=router_num;
   GP->node_num=router_num+term_num;
   GP->Substrate_N=calloc(GP->node_num,sizeof(SubstrateNode));
   if (GP->Substrate_N==NULL)
      return NULL;
   int node_id=0;
   int i=0;

   for (i=0;i<term_num;i++)
   {
      SubstrateNode snode=malloc(sizeof(struct SubstrateNode_struct));
      if (snode==NULL)
         return NULL;
      snode->id=node_id;
      snode->type=0;
      snode->res_CPU=u_space_random(0,term_max_cpu);
      snode->res_slice_num=(int)roundf(u_space_random(0,(float)term_max_slice));
      snode->assigned=0;
      snode->sliced=0;
      GP->Substrate_N[node_id]=snode;
      node_id++;
   }

   for (i=0;i<router_num;i++)
   {
      SubstrateNode snode=malloc(sizeof(struct SubstrateNode_struct));
      if (snode==NULL)
         return NULL;
      snode->id=node_id;
```

```
    snode->type=1;
    snode->res_CPU=0;
    snode->res_slice_num=(int)roundf(u_space_random(0,(float)router_max_slice));
    snode->assigned=0;
    snode->sliced=0;
    snode->group_GW_id=node_id;
    GP->Substrate_N[node_id]=snode;
    node_id++;
}

GP->Substrate_L=calloc(max_link_num,sizeof(SubstrateLink));
int link_id=0;
int link_num=0;
int rand_router_id;

for (i=0;i<term_num;i++)
{
    link_num++;
    if (link_num>max_link_num)
    {
        printf("Very small max_link_num!\n");
        return NULL;
    }
    SubstrateLink slink=malloc(sizeof(struct SubstrateLink_struct));
    if (slink==NULL)
        return NULL;
    rand_router_id=(int)roundf(u_space_random(term_num,GP->node_num-1));
    slink->id=link_id;
    slink->res_BW=u_space_random(0,max_bw);
    slink->end1=GP->Substrate_N[i];
    slink->end2=GP->Substrate_N[rand_router_id];
    slink->end1->group_GW_id=slink->end2->id;
    GP->Substrate_L[link_id]=slink;
    link_id++;
}

int j=0;
int router_for_con_id;
int con[GP->node_num][GP->node_num];
for (i=0;i<GP->node_num;i++)
    for (j=0;j<GP->node_num;j++)
        con[i][j]=0;
float connect_prob;
int enough_links_flag=0;

for(i=0;i<router_num;i++)
{
    for (j=0;j<router_num;j++)
    {
        connect_prob=u_random();
        if ((connect_prob>0.5)&&(j!=i))
        {
            router_for_con_id=term_num+j;
            if (con[i+term_num][router_for_con_id]==0)
            {
                con[i+term_num][router_for_con_id]=1;
                con[router_for_con_id][i+term_num]=1;
                link_num++;
                if (link_num>max_link_num)
                {
                    link_num--;
                    enough_links_flag=1;
                    break;
                }
                SubstrateLink slink=malloc(sizeof(struct SubstrateLink_struct));
                if (slink==NULL)
                    return NULL;
                slink->id=link_id;
                slink->res_BW=u_space_random(0,max_bw);
                slink->end1=GP->Substrate_N[i+term_num];
                slink->end2=GP->Substrate_N[router_for_con_id];
                GP->Substrate_L[link_id]=slink;
                link_id++;
            }
        }
```

```
        }
        if (enough_links_flag==1)
          break;
    }

    GP->link_num=link_num;
    for (i=link_num;i<max_link_num;i++)
    {
      free(GP->Substrate_L[i]);
    }

    return GP;
}


//print the substrate (graphviz processing file)
void print_GP(SubstrateGraph GP, FILE* fp)
{
    /*printf("\nSubstrateGraph id = %d \n",GP->id);
    printf("SubstrateGraph Terminals number = %d \n",GP->term_num);
    printf("SubstrateGraph Routers number = %d \n",GP->router_num);
    printf("SubstrateGraph total Nodes number = %d \n",GP->node_num);
    printf("SubstrateGraph Links number = %d \n",GP->link_num);
    printf("\n");*/

    fprintf(fp,"Graph Substrate {\n");
    int i=0;
    SubstrateNode n1;
    SubstrateNode n2;
    for (i=0;i<GP->link_num;i++)
    {
      n1=GP->Substrate_L[i]->end1;
      n2=GP->Substrate_L[i]->end2;

      if (n1->type==0)
      {
        fprintf(fp,"terminal%d_cpu_%d_s_%d -- ",n1->id,(int)(n1->res_CPU),n1->res_slice_num);
      }
      else
      {
        fprintf(fp,"router%d_s%d -- ",n1->id,n1->res_slice_num);
      }

      if (n2->type==0)
      {
        fprintf(fp,"terminal%d_CPU_s%d ",n2->id,(int)(n2->res_CPU),n2->res_slice_num);
      }
      else
      {
        fprintf(fp,"router%d_s%d ",n2->id,n2->res_slice_num);
      }
      fprintf(fp,"[ label=%.2f ];\n",GP->Substrate_L[i]->res_BW);
    }
    fprintf(fp,"}");
}


//deallocate memory occupied by substrate
void destroy_GP(SubstrateGraph GP)
{
    int i=0;
    for (i=0;i<GP->link_num;i++)
    {
      free(GP->Substrate_L[i]);
    }
    free(GP->Substrate_L);
    for (i=0;i<GP->node_num;i++)
    {
      free(GP->Substrate_N[i]);
    }
    free(GP->Substrate_N);
    free(GP);
}
```

```
//create a new virtual graph (terminal stars around routers, arbitrary router-router connection)
VNGraph new_VNGraph(int id, int router_num, int term_num, int max_link_num, float max_bw, float term_max_cpu, int max_hops)
{
   VNGraph GV=malloc(sizeof(struct VNGraph_struct));
   if (GV==NULL)
      return NULL;
   GV->id=id;
   GV->term_num=term_num;
   GV->router_num=router_num;
   GV->node_num=router_num+term_num;
   GV->VN_N=calloc(GV->node_num,sizeof(VNNode));
   if (GV->VN_N==NULL)
      return NULL;
   int node_id=0;
   int i=0;

   for (i=0;i<term_num;i++)
   {
      VNNode vnode=malloc(sizeof(struct VNNode_struct));
      if (vnode==NULL)
         return NULL;
      vnode->id=node_id;
      vnode->type=0;
      vnode->CPU_req=u_space_random(0,term_max_cpu);
      vnode->SNode_req=NULL;
      vnode->req_group_GW_id=-1;
      GV->VN_N[node_id]=vnode;
      node_id++;
   }

   for (i=0;i<router_num;i++)
   {
      VNNode vnode=malloc(sizeof(struct VNNode_struct));
      if (vnode==NULL)
         return NULL;
      vnode->id=node_id;
      vnode->type=1;
      vnode->SNode_req=NULL;
      vnode->req_group_GW_id=-1;
      vnode->CPU_req=0;
      GV->VN_N[node_id]=vnode;
      node_id++;
   }

   GV->VN_L=calloc(max_link_num,sizeof(VNLink));
   if (GV->VN_L==NULL)
      return NULL;
   int link_id=0;
   int link_num=0;
   int rand_router_id;

   for (i=0;i<term_num;i++)
   {
      link_num++;
      if (link_num>max_link_num)
      {
         printf("Very small max_link_num!\n");
         return NULL;
      }
      VNLink vlink=malloc(sizeof(struct VNLink_struct));
      if (vlink==NULL)
         return NULL;
      rand_router_id=(int)roundf(u_space_random(term_num,GV->node_num-1));
      vlink->id=link_id;
      vlink->BW_req=u_space_random(0,max_bw);
      vlink->max_hops=max_hops;
      vlink->end1=GV->VN_N[i];
      vlink->end2=GV->VN_N[rand_router_id];
      vlink->end1->next_vrouter=vlink->end2;
      GV->VN_L[link_id]=vlink;
      link_id++;
   }
```

```
    int j=0;
    int router_for_con_id;
    int con[GV->node_num][GV->node_num];
    for (i=0;i<GV->node_num;i++)
        for (j=0;j<GV->node_num;j++)
            con[i][j]=0;
    float connect_prob;
    int enough_links_flag=0;

    for(i=0;i<router_num;i++)
    {
        for (j=0;j<router_num;j++)
        {
            connect_prob=u_random();
            if ((connect_prob>0.5)&&(j!=i))
            {
                router_for_con_id=term_num+j;
                if (con[i+term_num][router_for_con_id]==0)
                {
                    con[i+term_num][router_for_con_id]=1;
                    con[router_for_con_id][i+term_num]=1;
                    link_num++;
                    if (link_num>max_link_num)
                    {
                        link_num--;
                        enough_links_flag=1;
                        break;
                    }
                    VNLink vlink=malloc(sizeof(struct VNLink_struct));
                    if (vlink==NULL)
                        return NULL;
                    vlink->id=link_id;
                    vlink->BW_req=u_space_random(0,max_bw);
                    vlink->max_hops=max_hops;
                    vlink->end1=GV->VN_N[i+term_num];
                    vlink->end2=GV->VN_N[router_for_con_id];
                    GV->VN_L[link_id]=vlink;
                    link_id++;
                }
            }
        }
        if (enough_links_flag==1)
            break;
    }

    GV->link_num=link_num;
    for (i=link_num;i<max_link_num;i++)
    {
        free(GV->VN_L[i]);
    }

    return GV;
}


//print the virtual network (graphviz processing file)
void print_GV(VNGraph GV, FILE* fp)
{
    /*printf("\nVNGraph id = %d \n",GV->id);
    printf("VNGraph Terminals number = %d \n",GV->term_num);
    printf("VNGraph Routers number = %d \n",GV->router_num);
    printf("VNGraph total Nodes number = %d \n",GV->node_num);
    printf("VNGraph Links number = %d \n",GV->link_num);
    printf("\n");*/

    fprintf(fp,"Graph Virtual {\n");
    int i=0;
    VNNode n1;
    VNNode n2;
    for (i=0;i<GV->link_num;i++)
    {
        n1=GV->VN_L[i]->end1;
        n2=GV->VN_L[i]->end2;
```

```
        if (n1->type==0)
        {
          fprintf(fp,"terminal%d -- ",n1->id);
        }
        else
        {
          fprintf(fp,"router%d -- ",n1->id);
        }

        if (n2->type==0)
        {
          fprintf(fp,"terminal%d ",n2->id);
        }
        else
        {
          fprintf(fp,"router%d ",n2->id);
        }
        fprintf(fp,"[ label=%.2f ];\n",GV->VN_L[i]->BW_req);
    }
    fprintf(fp,"}");
}


//deallocate memory occupied by virtual network
void destroy_GV(VNGraph GV)
{
    int i=0;
    for (i=0;i<GV->link_num;i++)
    {
        free(GV->VN_L[i]);
    }
    free(GV->VN_L);
    for (i=0;i<GV->node_num;i++)
    {
        free(GV->VN_N[i]);
    }
    free(GV->VN_N);
    free(GV);
}


//create a new subgraph of the virtual network
VNGraph new_GsubV(VNGraph GV)
{
     VNGraph GsubV=malloc(sizeof(struct VNGraph_struct));
     if (GsubV==NULL)
     {
        printf("Out of memory!\n");
        leave();
     }
     GsubV->id=GV->id;
     GsubV->term_num=0;
     GsubV->router_num=0;
     GsubV->node_num=0;
     GsubV->link_num=0;
     GsubV->VN_N=calloc(GV->node_num,sizeof(VNNode));
     if (GsubV->VN_N==NULL)
     {
        printf("Out of memory!\n");
        leave();
     }
     GsubV->VN_L=calloc(GV->link_num,sizeof(VNLink));
     if (GsubV->VN_L==NULL)
     {
        printf("Out of memory!\n");
        leave();
     }
     return GsubV;
}
```

```
//deallocate memory occupied by virtual subgraph
void destroy_GsubV(VNGraph GsubV)
{
    free(GsubV->VN_N);
    free(GsubV->VN_L);
    free(GsubV);
}


//create a new mapping of virtual->substrate
VNMapping new_VNM(VNGraph GV, int req_id)
{
   VNMapping VNM=malloc(sizeof(struct VNMapping_struct));
   if (VNM==NULL)
   {
       printf("Out of memory!\n");
       leave();
   }
   VNM->id=req_id;
   VNM->complete=0; //0=>incomplete, 1=>complete
   VNM->node_assign_num=0;
   VNM->link_assign_num=0;
   VNM->node_assign=calloc(GV->node_num,sizeof(SubstrateNode));
   if (VNM->node_assign==NULL)
   {
       printf("Out of memory!\n");
       leave();
   }
   int i=0;
   for (i=0;i<GV->node_num;i++)
      VNM->node_assign[i]=NULL;
   VNM->link_assign=calloc(GV->link_num,sizeof(SubstratePath));
   if (VNM->link_assign==NULL)
   {
       printf("Out of memory!\n");
       leave();
   }
   for (i=0;i<GV->link_num;i++)
      VNM->link_assign[i]=NULL;
   return VNM;
}


//print the mapping (graphviz processing file)
void print_VNM(VNMapping VNM, VNGraph GV, FILE* fpn, FILE* fpl)
{
  int i=0;
  fprintf(fpn,"Graph VNMnodes {\n");
  for (i=0;i<GV->node_num;i++)
  {
     if (VNM->node_assign[GV->VN_N[i]->id]==NULL)
     {
        printf("Wrong!\n");
        leave();
     }
     fprintf(fpn,"virtual%d_CPU%.0f   --   physical%d;\n",GV->VN_N[i]->id,GV->VN_N[i]->CPU_req,VNM->node_assign[GV-
>VN_N[i]->id]->id);
  }

  int j=0;
  fprintf(fpl,"Graph VNMlinks {\n");
  for (i=0;i<GV->link_num;i++)
  {
     if (VNM->link_assign[GV->VN_L[i]->id]==NULL)
     {
        printf("Wrong!\n");
        leave();
     }
     else
     {
        fprintf(fpl,"V%d_V%d_BW%.0f -- ",GV->VN_L[i]->end1->id,GV->VN_L[i]->end2->id,GV->VN_L[i]->BW_req);
        for (j=0;j<(VNM->link_assign[GV->VN_L[i]->id]->path_length+1);j++)
        {
           if (VNM->link_assign[GV->VN_L[i]->id]->path_nodes[j]==NULL)
           {
```

```
                printf("Wrong!\n");
                leave();
              }
              fprintf(fpl,"P%d_",VNM->link_assign[GV->VN_L[i]->id]->path_nodes[j]->id);
            }
            fprintf(fpl,";\n");
          }
        }
        fprintf(fpn,"}");
        fprintf(fpl,"}");
}


//deallocate memory occupied by mapping
void destroy_VNM(VNMapping VNM, VNGraph GV)
{
    free(VNM->node_assign);
    int i=0;
    int j=0;
    for (i=0;i<GV->link_num;i++)
    {
       if (VNM->link_assign[GV->VN_L[i]->id]!=NULL)
       {
          free(VNM->link_assign[GV->VN_L[i]->id]->path_nodes);
          free(VNM->link_assign[GV->VN_L[i]->id]->path_links);
          free(VNM->link_assign[GV->VN_L[i]->id]);
       }
    }
    free(VNM->link_assign);
    free(VNM);
}


//leave program abnormally
void leave(void)
{
   printf("\nAbnormal termination!\n");
   getchar();
   printf("Press <enter> or any key to exit\n");
   getchar();
   exit(0);
}
```

Βασικές συναρτήσεις αλγορίθμου vnmFlib basic_vnmFlib_funs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "aux_vnmFlib_funs.c"

//function prototypes

//basic vnmFlib functions according to the original paper
int vnmFlib(VNGraph GsubV, VNMapping VNM, VNGraph GV, SubstrateGraph GP, int req_id);
set_of_pairs genneigh(SubstrateGraph GP, VNGraph GV, VNGraph GsubV, VNMapping VNM);
valid_return valid(VNMapping VNM, VNGraph GsubV, VNNode nV, VNGraph GV, SubstrateNode nP, SubstrateGraph GP);
int create(VNGraph GsubV, VNGraph GV, VNMapping VNM, SubstrateGraph GP, VNNode nV, SubstrateNode nP, valid_return valid_ret, int req_id);
int undo_creation(VNGraph GsubV, VNMapping VNM, SubstrateGraph GP, VNNode nV, SubstrateNode nP, valid_return valid_ret, int req_id);

//global variables
int flib_calls;  //so as to terminate vnmFlib when needed


//basic vnmFlib heuristic backtracking algorithm
int vnmFlib(VNGraph GsubV, VNMapping VNM, VNGraph GV, SubstrateGraph GP, int req_id)
{
   flib_calls++;
   //printf("\n%dh flib call\n",flib_calls);
   if (flib_calls>((GV->node_num)*(GV->node_num+1)/2))
   {
      //printf("Mapping failed because num of flib calls exceeded!\n");
      return 0;
   }
   set_of_pairs C=genneigh(GP, GV, GsubV, VNM);
   int i=0;
   int j=0;
   int flib_done=0;
   int create_OK;
   int undo_create_OK;
   int ret;
   for (i=0;i<(C->pair_num);i++)
   {
      VNNode nV=C->sop[i]->nV;
      SubstrateNode nP=C->sop[i]->nP;
      valid_return valid_ret=valid(VNM, GsubV, nV, GV, nP, GP);
      if (valid_ret!=NULL)
      {
        create_OK=create(GsubV, GV, VNM, GP, nV, nP, valid_ret, req_id);
        ret=vnmFlib(GsubV, VNM, GV, GP, req_id);
        if (flib_calls>((GV->node_num)*(GV->node_num+1)/2))
        {
          for (j=0;j<C->pair_num;j++)
             free(C->sop[j]);
          free(C->sop);
          free(C);
          return 0;
        }
        if (ret==0)
        {
           undo_create_OK=undo_creation(GsubV, VNM, GP, nV, nP, valid_ret, req_id);
        }
        else
        {
           flib_done=1;
           break;
        }
      }
      if (VN_Graphs_Equal(GsubV,GV)==1)
      {
        VNM->complete=1;
        for (j=0;j<C->pair_num;j++)
           free(C->sop[j]);
        free(C->sop);
        free(C);
        return 1;
```

```
      }
   }
   for (j=0;j<C->pair_num;j++)
      free(C->sop[j]);
   free(C->sop);
   free(C);
   if (flib_done==0)
     return 0;
   else return 1;
}


//generate the set C of canditate node mappings
set_of_pairs genneigh(SubstrateGraph GP, VNGraph GV, VNGraph GsubV, VNMapping VNM)
{
   int SubstrateNode_num=GP->node_num;
   int VNNode_num=GV->node_num;
   F_type F=Form_F(GsubV,GV,VNM);
   int i=0;
   int j=0;
   set_of_pairs C=malloc(sizeof(set_of_pairs));
   if (C==NULL)
   {
      printf("Out of memory!\n");
      leave();
   }
   int pair_num=0;
   int C_length;
   if (F->length==0)
   {
      C_length=VNNode_num*SubstrateNode_num;
      C->sop=calloc(C_length,sizeof(pair));
      if (C->sop==NULL)
      {
         printf("Out of memory!\n");
         leave();
      }

      for (i=0;i<VNNode_num;i++)
      {
         for (j=0;j<SubstrateNode_num;j++)
         {
            C->sop[pair_num]=malloc(sizeof(struct pair_struct));
            if (C->sop[pair_num]==NULL)
            {
               printf("Out of memory!\n");
               leave();
            }
            C->sop[pair_num]->nV=GV->VN_N[i];
            C->sop[pair_num]->nP=GP->Substrate_N[j];
            pair_num++;
         }
      }
      C->pair_num=pair_num;
   }
   else
   {
      int num_of_assigned_SubsrateNodes=0;
      SubstrateNode current_SubstrateNode;

      for (i=0;i<SubstrateNode_num;i++)
      {
         current_SubstrateNode=GP->Substrate_N[i];
         if (current_SubstrateNode->assigned==1)
         {
           num_of_assigned_SubsrateNodes++;
         }
      }
      C_length=(F->length)*(SubstrateNode_num-num_of_assigned_SubsrateNodes);
      C->sop=calloc(C_length,sizeof(pair));
      if (C->sop==NULL)
      {
         printf("Out of memory!\n");
         leave();
      }
```

```
    for (i=0;i<(F->length);i++)
    {
       for (j=0;j<SubstrateNode_num;j++)
       {
          if (GP->Substrate_N[j]->assigned==0)
          {
            C->sop[pair_num]=malloc(sizeof(struct pair_struct));
            if (C->sop[pair_num]==NULL)
            {
               printf("Out of memory!\n");
               leave();
            }
            C->sop[pair_num]->nV=F->F_mat[i];
            C->sop[pair_num]->nP=GP->Substrate_N[j];
            pair_num++;
          }
       }
    }
    C->pair_num=pair_num;
  }

  set_of_pairs C_opt=optimize(C,GP,GV);
  free(C);
  set_of_pairs C_sorted=sort(C_opt);
  free(C_opt);

  return C_sorted;
}


//check for node and link mapping validity-locate broken connections
valid_return valid(VNMapping VNM, VNGraph GsubV, VNNode nV, VNGraph GV, SubstrateNode nP, SubstrateGraph GP)
{
  int j=0;
  if (VNM->node_assign[nV->id]!=NULL)
  {
    return NULL;
  }
  else
     if ((nP->assigned==1)||(nV->type!=nP->type))
     {
       return NULL;
     }
  else
     if (nV->SNode_req!=NULL)
     {
       if (nP!=nV->SNode_req)
       {
         return NULL;
       }
     }
  else
     if (nV->req_group_GW_id!=-1)
     {
       if (nV->req_group_GW_id!=nP->group_GW_id)
       {
          return NULL;
       }
     }
  else
  {
     if (nV->type==0)
     {
       if ((nV->CPU_req>nP->res_CPU)||(nP->res_slice_num==0))
       {
         return NULL;
       }
     }
     else
     {
        if (nP->res_slice_num==0)
        {
          return NULL;
        }
```

```
    }
}

int i=0;
valid_return val_ret=malloc(sizeof(struct valid_return_struct));
if (val_ret==NULL)
{
    printf("Out of memory!\n");
    leave();
}
VNNeighbours VNNeighb=find_VN_neighbours_in_GsubV(nV,GV,GsubV);
val_ret->VNNeighb=VNNeighb;
val_ret->virtual_links=calloc(VNNeighb->num,sizeof(VNLink));
if (val_ret->virtual_links==NULL)
{
    printf("Out of memory!\n");
    leave();
}

for (i=0;i<VNNeighb->num;i++)
    val_ret->virtual_links[i]=NULL;

val_ret->physical_path=calloc(VNNeighb->num,sizeof(SubstratePath));
if (val_ret->physical_path==NULL)
{
    printf("Out of memory!\n");
    leave();
}

for (i=0;i<VNNeighb->num;i++)
    val_ret->physical_path[i]=NULL;

VNNode cur_VN_neighbour;
SubstrateNode SubstrateNode_assigned_to_neighb;
SubstrateNode path_end1;
SubstrateNode path_end2;
VNLink nV_neighb_link;
float VNLink_BW_req;
SubstratePath path;
int hops;
int k=0;

for (i=0;i<VNNeighb->num;i++)
{
    cur_VN_neighbour=VNNeighb->VNNeighbours_array[i];
    SubstrateNode_assigned_to_neighb=VNM->node_assign[cur_VN_neighbour->id];
    path_end2=SubstrateNode_assigned_to_neighb;
    path_end1=nP;
    nV_neighb_link=find_Vlink(GV,nV,cur_VN_neighbour);
    val_ret->virtual_links[i]=nV_neighb_link;
    VNLink_BW_req=nV_neighb_link->BW_req;
    hops=nV_neighb_link->max_hops;
    path=find_path(path_end1,path_end2,GP,VNLink_BW_req,hops);
    if (path==NULL)
    {
        for (k=0;k<i;k++)
        {
            cur_VN_neighbour=VNNeighb->VNNeighbours_array[k];
            nV_neighb_link=find_Vlink(GV,nV,cur_VN_neighbour);
            path=val_ret->physical_path[k];
            for (j=0;j<path->path_length;j++)
                path->path_links[j]->res_BW=path->path_links[j]->res_BW+nV_neighb_link->BW_req;
        }
        free(val_ret->VNNeighb->VNNeighbours_array);
        free(val_ret->VNNeighb);
        free(val_ret->virtual_links);
        for (j=0;j<i;j++)
        {
            free(val_ret->physical_path[j]->path_nodes);
            free(val_ret->physical_path[j]->path_links);
            free(val_ret->physical_path[j]);
        }
        free(val_ret->physical_path);
        free(val_ret);
        return NULL;
```

```
        }
        val_ret->physical_path[i]=path;
        for (j=0;j<path->path_length;j++)
        {
           path->path_links[j]->res_BW=path->path_links[j]->res_BW-nV_neighb_link->BW_req;
           if (path->path_links[j]->res_BW<0)
           {
              printf("BW<0! Something went wrong in valid!\n");
              leave();
           }
        }
     }

     for (i=0;i<VNNeighb->num;i++)
     {
        cur_VN_neighbour=VNNeighb->VNNeighbours_array[i];
        nV_neighb_link=find_Vlink(GV,nV,cur_VN_neighbour);
        path=val_ret->physical_path[i];
        for (j=0;j<path->path_length;j++)
           path->path_links[j]->res_BW=path->path_links[j]->res_BW+nV_neighb_link->BW_req;
     }
     return val_ret;
}


//update virtual subgraph and mapping, allocate substrate resources
int create(VNGraph GsubV, VNGraph GV, VNMapping VNM, SubstrateGraph GP, VNNode nV, SubstrateNode nP, valid_return
valid_ret, int req_id)
{
    GsubV->VN_N[GsubV->node_num]=nV;
    GsubV->node_num++;
    if (nV->type==0)
       GsubV->term_num++;
    else
       GsubV->router_num++;

    int i=0;
    for (i=0;i<valid_ret->VNNeighb->num;i++)
    {
       GsubV->VN_L[GsubV->link_num]=valid_ret->virtual_links[i];
       GsubV->link_num++;
    }

    if ((VNM->node_assign_num>GV->node_num)||(VNM->link_assign_num>GV->link_num))
    {
      printf("\nSomething went wrong!\n");
      leave();
    }
    else if ((GsubV->node_num>GV->node_num)||(GsubV->link_num>GV->link_num))
    {
      printf("\nSomething went wrong!\n");
      leave();
    }

    VNM->node_assign[nV->id]=nP;
    VNM->node_assign_num++;
    //printf("\nnV %d - nP %d  added to VNM!\n",nV->id,nP->id);
    int j=0;
    for (i=0;i<valid_ret->VNNeighb->num;i++)
    {
       VNM->link_assign[valid_ret->virtual_links[i]->id]=valid_ret->physical_path[i];
       VNM->link_assign_num++;
       //printf("virtual link %d",valid_ret->virtual_links[i]->id);
       //printf(" between %d and %d \n",nV->id,valid_ret->VNNeighb->VNNeighbours_array[i]->id);
       //printf("by physical path: ");
       //for (j=0;j<valid_ret->physical_path[i]->path_length+1;j++)
          //printf("%d ",valid_ret->physical_path[i]->path_nodes[j]->id);
       //printf("between %d and %d\n",valid_ret->physical_path[i]->path_nodes[0]->id,valid_ret->physical_path[i]->path_nodes[j-
1]->id);
       //printf("added to VNM\n");
    }
    //printf("Now GsubV terminals are %d\n",GsubV->term_num);
    //printf("Now GsubV routers are  %d\n",GsubV->router_num);
    //printf("Now GsubV links are %d\n",GsubV->link_num);
```

```
    if (nP->type==0)
    {
       nP->res_CPU=nP->res_CPU-nV->CPU_req;
    }
    if (nP->sliced==0)
      nP->res_slice_num--;
    nP->assigned=1;
    nP->sliced++;

    SubstrateNode cur_node;
    SubstrateLink cur_link;
    for (i=0;i<valid_ret->VNNeighb->num;i++)
    {
       for (j=1;j<(valid_ret->physical_path[i]->path_length);j++)
       {
          cur_node=GP->Substrate_N[valid_ret->physical_path[i]->path_nodes[j]->id];
          if (cur_node->sliced==0)
            cur_node->res_slice_num--;
          cur_node->sliced++;
       }
       for (j=0;j<valid_ret->physical_path[i]->path_length;j++)
       {
          cur_link=GP->Substrate_L[valid_ret->physical_path[i]->path_links[j]->id];
          cur_link->res_BW=cur_link->res_BW-valid_ret->virtual_links[i]->BW_req;
          if (cur_link->res_BW<0)
          {
             printf("BW<0! Something went wrong in creation!\n");
             leave();
          }
       }
    }
    free(valid_ret->VNNeighb->VNNeighbours_array);

    return 1;
}


//undo the previous update of create, dealocate substrate resources
int undo_creation(VNGraph GsubV, VNMapping VNM, SubstrateGraph GP, VNNode nV, SubstrateNode nP, valid_return
valid_ret, int req_id)
{
   GsubV->node_num--;
   GsubV->VN_N[GsubV->node_num]=NULL;
   if (nV->type==0)
       GsubV->term_num--;
    else
       GsubV->router_num--;

   int i=0;
   for (i=0;i<valid_ret->VNNeighb->num;i++)
   {
      GsubV->link_num--;
      GsubV->VN_L[GsubV->link_num]=NULL;
   }

   int j=0;
   VNM->node_assign_num--;
   VNM->node_assign[nV->id]=NULL;
   //printf("\nnV %d - nP %d removed from VNM!\n",nV->id,nP->id);
   for (i=0;i<valid_ret->VNNeighb->num;i++)
   {
      VNM->link_assign_num--;
      VNM->link_assign[valid_ret->virtual_links[i]->id]=NULL;
      //printf("virtual link %d",valid_ret->virtual_links[i]->id);
      //printf(" between %d and %d \n",nV->id,valid_ret->VNNeighb->VNNeighbours_array[i]->id);
      //printf("by physical path: ");
      //for (j=0;j<valid_ret->physical_path[i]->path_length+1;j++)
         //printf("%d ",valid_ret->physical_path[i]->path_nodes[j]->id);
      //printf("between %d and %d\n",valid_ret->physical_path[i]->path_nodes[0]->id,valid_ret->physical_path[i]->path_nodes[j-
1]->id);
      //printf("removed from VNM\n");
   }
   //printf("Now GsubV terminals are %d\n",GsubV->term_num);
   //printf("Now GsubV routers are  %d\n",GsubV->router_num);
   //printf("Now GsubV links are %d\n",GsubV->link_num);
```

```
    if (nP->type==0)
    {
       nP->res_CPU=nP->res_CPU+nV->CPU_req;
    }
    if (nP->sliced==1)
      nP->res_slice_num++;
    nP->assigned=0;
    nP->sliced--;

    SubstrateNode cur_node;
    SubstrateLink cur_link;
    for (i=0;i<valid_ret->VNNeighb->num;i++)
    {
       for (j=1;j<(valid_ret->physical_path[i]->path_length);j++)
       {
          cur_node=GP->Substrate_N[valid_ret->physical_path[i]->path_nodes[j]->id];
          if (cur_node->sliced==1)
            cur_node->res_slice_num++;
          cur_node->sliced--;
       }
       for (j=0;j<valid_ret->physical_path[i]->path_length;j++)
       {
          cur_link=GP->Substrate_L[valid_ret->physical_path[i]->path_links[j]->id];
          cur_link->res_BW=cur_link->res_BW+valid_ret->virtual_links[i]->BW_req;
       }
    }
    for (i=0;i<valid_ret->VNNeighb->num;i++)
    {
       free(valid_ret->physical_path[i]->path_nodes);
       free(valid_ret->physical_path[i]->path_links);
       free(valid_ret->physical_path[i]);
    }
    free(valid_ret->virtual_links);
    free(valid_ret->VNNeighb);
    free(valid_ret);

    return 1;
}
```

Βοηθητικές συναρτήσεις αλγορίθμου vnmFlib aux_vnmFlib_funs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "graph_funs.c"

//function prototypes

//for misc actions
int VNNode_in_VNGraph(VNNode nV, VNGraph GV);
int SubstrateNode_in_MGsubV(SubstrateNode nP, VNMapping VNM);
VNNeighbours find_VN_neighbours_in_GsubV(VNNode nV, VNGraph GV, VNGraph GsubV);
int VN_Graphs_Equal(VNGraph GsubV, VNGraph GV);
SubstrateLink find_Plink(SubstrateGraph GP, SubstrateNode end1, SubstrateNode end2);
VNLink find_Vlink(VNGraph GV, VNNode end1, VNNode end2);

//for optimization stage in genneigh
F_type Form_F(VNGraph GsubV, VNGraph GV, VNMapping VNM);
set_of_pairs optimize(set_of_pairs C, SubstrateGraph GP, VNGraph GV);
set_of_pairs sort(set_of_pairs C);

//for path finding in valid
SubstratePath find_path(SubstrateNode sn1, SubstrateNode sn2, SubstrateGraph GP, float VNink_BW_req, int max_hops);

//for resource deallocation
void release_resources(SubstrateGraph GP, VNMapping VNM, VNGraph GV);


//is a virtual node inside a virtual graph?
int VNNode_in_VNGraph(VNNode nV, VNGraph GV)
{
   if ((GV==NULL)||(GV->node_num==0))
   {
      return 0;
   }
   else
   {
      int is_in_flag=0;
      int VNNode_num=GV->node_num;
      VNNode current_VNNode;
      int i=0;
      for (i=0;i<VNNode_num;i++)
      {
         current_VNNode=GV->VN_N[i];
         if ((current_VNNode!=NULL)&&(nV==current_VNNode))
         {
            is_in_flag=1;
            break;
         }
      }
      return is_in_flag;
   }
}


//is a physical host already mapped and registered?<==>nP->assigned==1?
int SubstrateNode_in_MGsubV(SubstrateNode nP, VNMapping VNM)
{
   if ((VNM==NULL)||(VNM->node_assign_num==0))
   {
      return 0;
   }
   else
   {
      int is_in_flag=0;
      int VNNode_assigned_num=VNM->node_assign_num;
      SubstrateNode current_SubstrateNode;
      int i=0;

      for (i=0;i<VNNode_assigned_num;i++)
      {
         current_SubstrateNode=VNM->node_assign[i];
         if ((current_SubstrateNode!=NULL)&&(nP==current_SubstrateNode))
```

```
        {
          is_in_flag=1;
          break;
        }
      }
      return is_in_flag;
    }
}


//are two virtual graphs equal (isomorphic)?
int VN_Graphs_Equal(VNGraph GsubV, VNGraph GV)
{
   if ((GsubV->node_num==0)||(GV->node_num==0)||(GsubV==NULL)||(GV==NULL))
      return 0;
   else if ((GsubV->node_num!=GV->node_num)||(GsubV->link_num!=GV->link_num))
      return 0;
   else
   {
      //the following code for equality check is optional
      //the procedure of constructing the GsubV is right
      //so there is no chance that the following check should fail if the
      //previous checks fail (and GsubV and GV are of equal size)
      //therefore it is omitted due to better run-times (not necessary)
      /*
      int node_equality[GV->node_num];
      int link_equality[GV->link_num];
      int i=0;
      int j=0;

      for (i=0;i<GV->node_num;i++)
      {
         node_equality[i]=0;
         for (j=0;j<GsubV->node_num;j++)
         {
            if (GV->VN_N[i]==GsubV->VN_N[j])
               node_equality[i]=1;
         }
      }

      int node_equal_flag=1;

      for (i=0;i<GV->node_num;i++)
      {
         if (node_equality[i]==0)
         {
            node_equal_flag=0;
            break;
         }
      }
      if (node_equal_flag==0)
      {
         leave();
         return 0;
      }

      for (i=0;i<GV->link_num;i++)
      {
         link_equality[i]=0;
         for (j=0;j<GsubV->link_num;j++)
         {
            if (GV->VN_L[i]==GsubV->VN_L[j])
               link_equality[i]=1;
         }
      }

      int link_equal_flag=1;

      for (i=0;i<GV->link_num;i++)
      {
         if (link_equality[i]==0)
         {
            link_equal_flag=0;
            break;
         }
```

```
        }
        if (link_equal_flag==0)
        {
            leave();
            return 0;
        }
        */
        return 1;
    }
}


//find the virtual networks of a virtual node inside a virtual subgraph
VNNeighbours find_VN_neighbours_in_GsubV(VNNode nV, VNGraph GV, VNGraph GsubV)
{
    int i=0;
    int num=0;
    VNNeighbours VNNeighb=malloc(sizeof(struct VNNeighbours_struct));
    if (VNNeighb==NULL)
    {
        printf("Out of memory!\n");
        leave();
    }
    VNNeighb->num=0;
    VNNeighb->VNNeighbours_array=calloc(GV->node_num,sizeof(VNNode));
    if (VNNeighb->VNNeighbours_array==NULL)
    {
        printf("Out of memory!\n");
        leave();
    }
    VNNode VNneigbour;
    VNLink VNneighblink;

    for (i=0;i<GV->link_num;i++)
    {
        VNneighblink=GV->VN_L[i];
        if ((VNneighblink->end1==nV)&&(VNNode_in_VNGraph(VNneighblink->end2,GsubV)==1))
        {
            VNNeighb->VNNeighbours_array[num]=VNneighblink->end2;
            num++;
        }
        else if ((VNneighblink->end2==nV)&&(VNNode_in_VNGraph(VNneighblink->end1,GsubV)==1))
        {
            VNNeighb->VNNeighbours_array[num]=VNneighblink->end1;
            num++;
        }
        VNNeighb->num=num;
    }

    for (i=num;i<(GV->node_num);i++)
    {
        free(VNNeighb->VNNeighbours_array[i]);
    }

    return VNNeighb;
}


//locate a specific physical link of the substrate connecting two physical hosts
SubstrateLink find_Plink(SubstrateGraph GP, SubstrateNode end1, SubstrateNode end2)
{
    int i=0;
    for (i=0;i<GP->link_num;i++)
    {
        if ((GP->Substrate_L[i]->end1==end1)&&(GP->Substrate_L[i]->end2==end2))
            return GP->Substrate_L[i];
        if ((GP->Substrate_L[i]->end2==end1)&&(GP->Substrate_L[i]->end1==end2))
            return GP->Substrate_L[i];
    }
    return NULL;
}
```

```
//locate a specific virtual link of the virtual network connecting two virtual nodes
VNLink find_Vlink(VNGraph GV, VNNode end1, VNNode end2)
{
   int i=0;
   for (i=0;i<GV->link_num;i++)
   {
      if ((GV->VN_L[i]->end1==end1)&&(GV->VN_L[i]->end2==end2))
        return GV->VN_L[i];
      if ((GV->VN_L[i]->end2==end1)&&(GV->VN_L[i]->end1==end2))
        return GV->VN_L[i];
   }
   return NULL;
}


//form the projection set F of virtual nodes as described in the original paper
F_type Form_F(VNGraph GsubV, VNGraph GV, VNMapping VNM)
{
   int VNNode_num=GV->node_num;
   int VNLink_num=GV->link_num;
   F_type F;
   F=malloc(sizeof(struct F_type_struct));
   if (F==NULL)
   {
      printf("Out of memory!\n");
      leave();
   }
   F->F_mat=calloc(VNNode_num,(sizeof(VNNode)));
   if (F->F_mat==NULL)
   {
      printf("Out of memory!\n");
      leave();
   }
   int i=0;
   int F_length=0;
   VNLink current_VNLink;
   VNNode end1;
   VNNode end2;
   int already_in_F[VNNode_num];
   for (i=0;i<VNNode_num;i++)
      already_in_F[i]=0;

   for (i=0;i<VNLink_num;i++)
   {
      current_VNLink=GV->VN_L[i];
      end1=current_VNLink->end1;
      end2=current_VNLink->end2;
      if ((VNM->node_assign[end1->id]==NULL)&&(VNM->node_assign[end2->id]!=NULL)&&(already_in_F[end1->id]==0))
      {
         F->F_mat[F_length]=end1;
         F_length++;
         already_in_F[end1->id]=1;
      }
      else if ((VNM->node_assign[end2->id]==NULL) && (VNM->node_assign[end1->id]!=NULL)&&(already_in_F[end2-
>id]==0))
      {
         F->F_mat[F_length]=end2;
         F_length++;
         already_in_F[end2->id]=1;
      }
   }
   F->length=F_length;

   for (i=F_length;i<VNNode_num;i++)
   {
      free(F->F_mat[i]);
   }

   return F;
}
```

```
//optimization stage of the set C of candidate node mappings (node constraints respected)
set_of_pairs optimize(set_of_pairs C, SubstrateGraph GP, VNGraph GV)
{
   if (C==NULL)
   {
      printf("Fatal error!\n");
      leave();
   }
   int i=0;
   set_of_pairs optimized_C=malloc(sizeof(struct set_of_pairs_struct));
   if (optimized_C==NULL)
   {
      printf("Out of memory!\n");
      leave();
   }
   optimized_C->sop=calloc((C->pair_num),sizeof(pair));
   if (optimized_C->sop==NULL)
   {
      printf("Out of memory!\n");
      leave();
   }
   for (i=0;i<C->pair_num;i++)
   {
      optimized_C->sop[i]=malloc(sizeof(struct pair_struct));
      if (optimized_C->sop[i]==NULL)
      {
        printf("Out of memory!\n");
        leave();
      }
   }
   int pair_num=0;
   int ok_flag=1;
   SubstrateNode nP;
   VNNode nV;
   SubstrateLink pl;
   VNLink vl;

   for (i=0;i<(C->pair_num);i++)
   {
      nP=C->sop[i]->nP;
      nV=C->sop[i]->nV;
      if (nP->assigned==0)
      {
         if ((nP->type)==(nV->type))
         {
            if (nV->SNode_req!=NULL)
            {
              if (nP!=nV->SNode_req)
              {
                 ok_flag=0;
              }
            }
            if (nV->req_group_GW_id!=-1)
            {
               if (nV->req_group_GW_id!=nP->group_GW_id)
               {
                  ok_flag=0;
               }
            }
            if ((nP->type==0))
            {
               if (((nP->res_CPU)<(nV->CPU_req))||(nP->res_slice_num==0))
               {
                  ok_flag=0;
               }
               pl=find_Plink(GP,nP,GP->Substrate_N[nP->group_GW_id]);
               vl=find_Vlink(GV,nV,nV->next_vrouter);
               if ((pl->res_BW)<(vl->BW_req))
                  ok_flag=0;
            }
            else
            {
               if (nP->res_slice_num==0)
                  ok_flag=0;
            }
```

```
            }
            else
               ok_flag=0;
         }
         else
            ok_flag=0;
         if (ok_flag==1)
         {
            optimized_C->sop[pair_num]->nV=C->sop[i]->nV;
            optimized_C->sop[pair_num]->nP=C->sop[i]->nP;
            pair_num++;
         }
         ok_flag=1;
      }

      optimized_C->pair_num=pair_num;
      for (i=pair_num;i<C->pair_num;i++)
      {
         free(optimized_C->sop[i]);
      }

      for (i=0;i<C->pair_num;i++)
        free(C->sop[i]);
      free(C->sop);

      return optimized_C;
   }


   //sorting stage of the set C of candidate node mappings (greedy sort)
   set_of_pairs sort(set_of_pairs C)
   {
      if (C==NULL)
      {
         printf("Fatal error!\n");
         leave();
      }
      int i=0;
      set_of_pairs sorted_C=malloc(sizeof(struct set_of_pairs_struct));
      if (sorted_C==NULL)
      {
         printf("Out of memory!\n");
         leave();
      }
      sorted_C->sop=calloc((C->pair_num),sizeof(pair));
      if (sorted_C->sop==NULL)
      {
         printf("Out of memory!\n");
         leave();
      }
      for (i=0;i<C->pair_num;i++)
      {
         sorted_C->sop[i]=malloc(sizeof(struct pair_struct));
         if (sorted_C->sop==NULL)
         {
            printf("Out of memory!\n");
            leave();
         }
      }
      int j=0;
      int terminal_assign_num=0;
      int router_assign_num=0;

      for (i=0;i<(C->pair_num);i++)
      {
         if (C->sop[i]->nP->type==0)
            terminal_assign_num++;
         else
            router_assign_num++;
      }

      set_of_pairs C_terminal=malloc(sizeof(struct set_of_pairs_struct));
      if (C_terminal==NULL)
      {
         printf("Out of memory!\n");
```

```
    leave();
  }
  C_terminal->sop=calloc(terminal_assign_num,sizeof(pair));
  if ((terminal_assign_num!=0)&&(C_terminal->sop==NULL))
  {
    printf("Out of memory!\n");
    leave();
  }
  for (i=0;i<terminal_assign_num;i++)
  {
    C_terminal->sop[i]=malloc(sizeof(struct pair_struct));
    if (C_terminal->sop[i]==NULL)
    {
      printf("Out of memory!\n");
      leave();
    }
  }
  set_of_pairs C_router=malloc(sizeof(struct set_of_pairs_struct));
  if (C_router==NULL)
  {
    printf("Out of memory!\n");
    leave();
  }
  C_router->sop=calloc(router_assign_num,sizeof(pair));
  if ((router_assign_num!=0)&&(C_router->sop==NULL))
  {
    printf("Out of memory!\n");
    leave();
  }
  for (i=0;i<router_assign_num;i++)
  {
    C_router->sop[i]=malloc(sizeof(struct pair_struct));
    if (C_router->sop[i]==NULL)
    {
      printf("Out of memory!\n");
      leave();
    }
  }
  int term_count=0;
  int rout_count=0;

  for (i=0;i<(C->pair_num);i++)
  {
    if (C->sop[i]->nP->type==0)
    {
      C_terminal->sop[term_count]->nV=C->sop[i]->nV;
      C_terminal->sop[term_count]->nP=C->sop[i]->nP;
      term_count++;
    }
    else
    {
      C_router->sop[rout_count]->nV=C->sop[i]->nV;
      C_router->sop[rout_count]->nP=C->sop[i]->nP;
      rout_count++;
    }
  }

  float a=0.5;
  float weighted_res_resources_j;
  float weighted_res_resources_j_1;
  float req_resources_j;
  float req_resources_j_1;
  int swap_flag;
  pair temp=malloc(sizeof(struct pair_struct));

  for (i=terminal_assign_num-1;i>=0;i--)
  {
    swap_flag=0;
    for (j=0;j<i;j++)
    {
      weighted_res_resources_j=a*(C_terminal->sop[j]->nP->res_CPU)+(1-a)*(float)(C_terminal->sop[j]->nP-
>res_slice_num);
      weighted_res_resources_j_1=a*(C_terminal->sop[j+1]->nP->res_CPU)+(1-a)*(float)(C_terminal->sop[j+1]->nP-
>res_slice_num);
      if (weighted_res_resources_j<weighted_res_resources_j_1)
```

```
        {
            temp->nV=C_terminal->sop[j]->nV;
            temp->nP=C_terminal->sop[j]->nP;
            C_terminal->sop[j]->nV=C_terminal->sop[j+1]->nV;
            C_terminal->sop[j]->nP=C_terminal->sop[j+1]->nP;
            C_terminal->sop[j+1]->nV=temp->nV;
            C_terminal->sop[j+1]->nP=temp->nP;
            swap_flag=1;
        }
    }
    if (swap_flag==0)
        break;
}


for (i=terminal_assign_num-1;i>=0;i--)
{
    swap_flag=0;
    for (j=0;j<i;j++)
    {
        req_resources_j=C_terminal->sop[j]->nV->CPU_req;
        req_resources_j_1=C_terminal->sop[j+1]->nV->CPU_req;
        if (req_resources_j<req_resources_j_1)
        {
            temp->nV=C_terminal->sop[j]->nV;
            temp->nP=C_terminal->sop[j]->nP;
            C_terminal->sop[j]->nV=C_terminal->sop[j+1]->nV;
            C_terminal->sop[j]->nP=C_terminal->sop[j+1]->nP;
            C_terminal->sop[j+1]->nV=temp->nV;
            C_terminal->sop[j+1]->nP=temp->nP;
            swap_flag=1;
        }
    }
    if (swap_flag==0)
        break;
}

for (i=router_assign_num-1;i>=0;i--)
{
    swap_flag=0;
    for (j=0;j<i;j++)
    {
        weighted_res_resources_j=(float)(C_router->sop[j]->nP->res_slice_num);
        weighted_res_resources_j_1=(float)(C_router->sop[j+1]->nP->res_slice_num);
        if (weighted_res_resources_j<weighted_res_resources_j_1)
        {
            temp->nV=C_router->sop[j]->nV;
            temp->nP=C_router->sop[j]->nP;
            C_router->sop[j]->nV=C_router->sop[j+1]->nV;
            C_router->sop[j]->nP=C_router->sop[j+1]->nP;
            C_router->sop[j+1]->nV=temp->nV;
            C_router->sop[j+1]->nP=temp->nP;
            swap_flag=1;
        }
    }
    if (swap_flag==0)
        break;
}
free(temp);
int pair_num=0;

for (i=0;i<terminal_assign_num;i++)
{
    sorted_C->sop[pair_num]->nV=C_terminal->sop[i]->nV;
    sorted_C->sop[pair_num]->nP=C_terminal->sop[i]->nP;
    pair_num++;
}

for (i=0;i<router_assign_num;i++)
{
    sorted_C->sop[pair_num]->nV=C_router->sop[i]->nV;
    sorted_C->sop[pair_num]->nP=C_router->sop[i]->nP;
    pair_num++;
}
```

```
    sorted_C->pair_num=pair_num;
    for (i=pair_num;i<C->pair_num;i++)
    {
        free(sorted_C->sop[i]);
    }

    for (i=0;i<terminal_assign_num;i++)
        free(C_terminal->sop[i]);
    free(C_terminal->sop);
    free(C_terminal);
    for (i=0;i<router_assign_num;i++)
        free(C_router->sop[i]);
    free(C_router->sop);
    free(C_router);
    for (i=0;i<C->pair_num;i++)
        free(C->sop[i]);
    free(C->sop);

    return sorted_C;
}


//find a suitable path connecting two physical hosts, witch satisfies
//the BW and max_hops constraint using Dijkstra's algorithm
SubstratePath find_path(SubstrateNode sn1, SubstrateNode sn2, SubstrateGraph GP, float VNink_BW_req, int max_hops)
{

    if (sn1==sn2)
    {
        printf("same physical???\n");
        leave();
    }

    int* node_label=calloc(GP->node_num,sizeof(int));  //-1 if full router, 0 if unaccounted, 1 if accounted
    if (node_label==NULL)
    {
        printf("Out of memory!\n");
        leave();
    }

    int* current_node_set=calloc(GP->node_num,sizeof(int));
    if (current_node_set==NULL)
    {
        printf("Out of memory!\n");
        leave();
    }

    int* node_pred=calloc(GP->node_num,sizeof(int));
    if (node_pred==NULL)
    {
        printf("Out of memory!\n");
        leave();
    }

    int current_nodes_accounted=0;

    int source=sn1->id;
    int dest=sn2->id;

    //initialization
    int i=0;
    for (i=0;i<GP->node_num;i++)
    {
        node_label[GP->Substrate_N[i]->id]=0;
        node_pred[GP->Substrate_N[i]->id]=-1;
        if ((GP->Substrate_N[i]->type==1)&&(GP->Substrate_N[i]->res_slice_num==0))
        {
            if (GP->Substrate_N[i]->id!=dest)
                node_label[GP->Substrate_N[i]->id]=-1;
        }
    }
    node_label[source]=1;

    int* Dn=calloc(GP->node_num,sizeof(int)); //delay from source
    if (Dn==NULL)
```

```
{
   printf("Out of memory!\n");
   leave();
}

int** dl; //delay of link
dl=calloc(GP->node_num,sizeof(int*));
for (i=0;i<GP->node_num;i++)
   dl[i]=calloc(GP->node_num,sizeof(int));

int j=0;
for (i=0;i<GP->node_num;i++)
   for (j=0;j<GP->node_num;j++)
   {
      dl[GP->Substrate_N[i]->id][GP->Substrate_N[j]->id]=1000; //infinity
   }
for (i=0;i<GP->link_num;i++)
{
   if (GP->Substrate_L[i]->res_BW>VNink_BW_req)
   {
      dl[GP->Substrate_L[i]->end1->id][GP->Substrate_L[i]->end2->id]=1;
      dl[GP->Substrate_L[i]->end2->id][GP->Substrate_L[i]->end1->id]=1;
   }
}

for (i=0;i<GP->node_num;i++)
{
   if (GP->Substrate_L[i]->id!=source)
   {
     Dn[GP->Substrate_N[i]->id]=dl[source][GP->Substrate_N[i]->id];
     if (Dn[GP->Substrate_N[i]->id]==1)
       node_pred[GP->Substrate_N[i]->id]=source;
   }
}
Dn[source]=0;
node_pred[source]=-1;

current_node_set[0]=source;
current_nodes_accounted=1;

//running
int path_found=0;
int min;
int min_index;

while(node_label[dest]!=1)
{
   min=Dn[dest];
   min_index=dest;
   for (i=0;i<GP->node_num;i++)
   {
      if (node_label[GP->Substrate_N[i]->id]==0)
      {
         if (Dn[GP->Substrate_N[i]->id]<min)
         {
            min_index=GP->Substrate_N[i]->id;
            min=Dn[min_index];
         }
      }
   }
   node_label[min_index]=1;
   current_node_set[current_nodes_accounted]=min_index;
   current_nodes_accounted++;

   if (min>max_hops)
   {
     path_found=0;
     break;
   }

   if (min_index==dest)
   {
      path_found=1;
      break;
   }
```

```
    int l;
    int k;
    for (i=0;i<GP->node_num;i++)
    {
       if (node_label[GP->Substrate_N[i]->id]==0)
       {
          l=GP->Substrate_N[i]->id;
          k=min_index;
          if (Dn[l]>Dn[k]+dl[k][l])
          {
             Dn[l]=Dn[k]+dl[k][l];
             node_pred[l]=k;
          }
       }
    }
}

if (path_found==1)
{
    SubstratePath returnPath=malloc(sizeof(struct SubstratePath_struct));
    if (returnPath==NULL)
    {
       printf("Out of memory\n");
       leave();
    }
    returnPath->path_length=0;
    returnPath->path_links=calloc(min,(sizeof(SubstrateLink)));
    if (returnPath->path_links==NULL)
    {
       printf("Out of memory\n");
       leave();
    }
    returnPath->path_nodes=calloc(min+1,(sizeof(SubstrateNode)));
    if ( returnPath->path_nodes==NULL)
    {
       printf("Out of memory\n");
       leave();
    }
    int formed_path=0;
    int inod=0;
    int ilin=0;

    SubstrateNode curnode=GP->Substrate_N[dest];
    SubstrateNode prevnode;
    SubstrateLink curlink;

    while (formed_path==0)
    {
      returnPath->path_nodes[inod]=curnode;
      inod++;
      if (curnode->id==source)
      {
        formed_path=1;
      }
      else
      {
         prevnode=GP->Substrate_N[node_pred[curnode->id]];
         curlink=find_Plink(GP,curnode,prevnode);
         if (curlink==NULL)
         {
           printf("Path error! Cannot find link!\n");
           leave();
         }
         returnPath->path_links[ilin]=curlink;
         curnode=prevnode;
         ilin++;
         returnPath->path_length++;
      }
    }

    /*for (i=returnPath->path_length+1;i<GP->node_num;i++)
       free(returnPath->path_nodes[i]);
    for (i=returnPath->path_length;i<GP->link_num;i++)
       free(returnPath->path_links[i]);*/
```

```
        free(node_label);
        free(current_node_set);
        free(node_pred);
        free(Dn);
        for (i=0;i<GP->node_num;i++)
            free(dl[i]);
        free(dl);
        return returnPath;
    }
    else
    {
        free(node_label);
        free(current_node_set);
        free(node_pred);
        free(Dn);
        for (i=0;i<GP->node_num;i++)
            free(dl[i]);
        free(dl);
        return NULL;
    }
}


//release the substrate resources that a specific mapping of a virtual network occupies
void release_resources(SubstrateGraph GP, VNMapping VNM, VNGraph GV)
{
    int i=0;
    int j=0;

    int* slice_release_OK=calloc(GP->node_num,sizeof(int));
    for (i=0;i<GP->node_num;i++)
        slice_release_OK[i]=0;

    for (i=0;i<GV->node_num;i++)
    {
        if (VNM->node_assign[GV->VN_N[i]->id]!=NULL)
        {
            VNM->node_assign[GV->VN_N[i]->id]->res_CPU+=GV->VN_N[i]->CPU_req;
            if (slice_release_OK[VNM->node_assign[GV->VN_N[i]->id]->id]==0)
            {
                VNM->node_assign[GV->VN_N[i]->id]->res_slice_num++;
                slice_release_OK[VNM->node_assign[GV->VN_N[i]->id]->id]=1;
            }
        }
    }

    for (i=0;i<GV->link_num;i++)
    {
        if (VNM->link_assign[GV->VN_L[i]->id]!=NULL)
        {
            for (j=0;j<VNM->link_assign[GV->VN_L[i]->id]->path_length;j++)
            {
                VNM->link_assign[GV->VN_L[i]->id]->path_links[j]->res_BW+=GV->VN_L[i]->BW_req;
            }
            for (j=0;j<VNM->link_assign[GV->VN_L[i]->id]->path_length+1;j++)
            {
                if (slice_release_OK[VNM->link_assign[GV->VN_L[i]->id]->path_nodes[j]->id]==0)
                {
                    VNM->link_assign[GV->VN_L[i]->id]->path_nodes[j]->res_slice_num++;
                    slice_release_OK[VNM->link_assign[GV->VN_L[i]->id]->path_nodes[j]->id]=1;
                }
            }
        }
    }
    free(slice_release_OK);
}
```

Συναρτήσεις προσομοίωσης simulation_funs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "basic_vnmFlib_funs.c"

//function prototypes

//general simulation
void simulation(int te_num, int rt_num, float beta, int hops);

//request production
VNRequest new_VNRequest(int id, VNGraph GV, float arrival_time, float duration);

//VNR and event queue management
void insert_VNR_in_to_serve_queue(VNRequest VNR);
void delete_VNR_from_queue(VNRequest VNR, VNR_queueptr queue);
void transfer_VNR_to_serving_queue(VNRequest VNR);
void insert_event(float t, VNRequest VNR, int evtype);
void extract_event(eventptr e, float* t, VNRequest* VNRptr, int* evtype);
void delete_event(VNRequest VNR, int evtype);
eventptr find_event(VNRequest VNR, int evtype);
int there_is_event(int evtype);

//event handling
void arrival(int id, int router_num, int term_num, int max_link_num, float max_bw, float term_max_cpu, int max_hops);  //OK
void check_to_serve_queue(VNRequest VNR);
void serve_VNR(VNRequest VNR);
void abnormal_expiration(VNRequest VNR);
void normal_expiration(VNRequest VNR);

//cost_accounting
float revenue_to_cost_ratio(VNRequest VNR);

//global simulation variables
int s1,s2;
event_queue_ptr event_queue;
VNR_queueptr to_serve_queue;
VNR_queueptr serving_queue;
float clock_timer;
float Dt;
float requests_per_Dt;
int max_requests;
SubstrateGraph SubG;


//simulate uniform distribution in (0,1)
float u_random(void)
{
   int z,k;
   k=s1/53668;
   s1=40014*(s1-k*53668)-k*12211;
   if (s1<0) s1=s1+2147483563;
   k=s2/52774;
   s2=40692*(s2-k*52774)-k*3791;
   if (s2<0) s2=s2+2147483399;
   z=s1-s2;
   if (z<1) z=z+2147483562;
   return z*4.656613e-10;
}


//simulate uniform distribution in (a,b)
float u_space_random(float a, float b)
{
   float value=(b-a)*u_random()+a;
   if(a>b)
      return -1;
   else
      return value;
}
```

```
//create new VNR and initialize its parameters
VNRequest new_VNRequest(int id, VNGraph GV, float arrival_time, float duration)
{
   VNRequest VNReq=malloc(sizeof(struct VNRequest_struct));
   if (VNReq==NULL)
   {
      printf("Out of memory!!!\n");
      leave();
   }
   VNReq->id=id;
   VNReq->VNG=GV;
   VNReq->VNM=NULL;
   VNReq->arrival_time=arrival_time;
   VNReq->duration=duration;
   VNReq->mapped=0;
   VNReq->R_dia_C=0.0;
   return VNReq;
}


//insert VNR in to_serve_queue after arrival
void insert_VNR_in_to_serve_queue(VNRequest VNR)
{
    if (VNR==NULL)
    {
      printf("NULL VNR to insert to to_serve_queue!\n");
      leave();
    }

    if (to_serve_queue->head==NULL)
    {
      VNR->previous=NULL;
      VNR->next=NULL;
      to_serve_queue->head=VNR;
      to_serve_queue->tail=VNR;
      to_serve_queue->length=1;
    }
    else
    {
      VNR->previous=to_serve_queue->tail;
      VNR->next=NULL;
      to_serve_queue->tail->next=VNR;
      to_serve_queue->tail=VNR;
      to_serve_queue->length++;
    }
}


//delete VNR from a queue after expiration (abnormal/normal)
void delete_VNR_from_queue(VNRequest VNR, VNR_queueptr queue)
{
  if (VNR==NULL)
  {
    printf("NULL VNR to delete from queue!\n");
    system("PAUSE");
    exit(0);
  }

  if (queue->length==1)
  {
    queue->head=NULL;
    queue->tail=NULL;
  }
  else if (queue->head==VNR)
  {
    queue->head=VNR->next;
    queue->head->previous=NULL;
  }
  else if (queue->tail==VNR)
  {
    queue->tail=VNR->previous;
    queue->tail->next=NULL;
  }
  else
```

```
  {
    VNRequest prev=VNR->previous;
    VNRequest next=VNR->next;
    prev->next=next;
    next->previous=prev;
  }
  queue->length--;
}


//transfer a VNR to serving_queue after successful mapping
void transfer_VNR_to_serving_queue(VNRequest VNR)
{
    if (VNR==NULL)
    {
      printf("NULL VNR to transfer!\n");
      system("PAUSE");
      exit(0);
    }

    delete_VNR_from_queue(VNR,to_serve_queue);

    if (serving_queue->head==NULL)
    {
      VNR->previous=NULL;
      VNR->next=NULL;
      serving_queue->head=VNR;
      serving_queue->tail=VNR;
      serving_queue->length=1;
    }
    else
    {
      VNRequest temp=serving_queue->head;
      while (temp!=NULL)
      {
        if ((VNR->deployment_time+VNR->duration)<(temp->deployment_time+temp->duration))
        {
          VNR->next=temp;
          if (temp==serving_queue->head)
          {
            serving_queue->head=VNR;
            VNR->previous=NULL;
          }
          else
          {
             temp->previous->next=VNR;
             VNR->previous=temp->previous;
          }
          temp->previous=VNR;
          break;
        }
        else
          temp=temp->next;
      }
      if (temp==NULL)
      {
        VNR->previous=serving_queue->tail;
        VNR->next=NULL;
        serving_queue->tail->next=VNR;
        serving_queue->tail=VNR;
      }
      serving_queue->length++;
    }
}


//insert an event scheduled at specific time with specific VNR and type
void insert_event(float t, VNRequest VNR, int evtype)
{
    eventptr neweventptr=malloc(sizeof(struct event_struct));
    neweventptr->time=t;
    neweventptr->VNR=VNR;
    neweventptr->eventtype=evtype;
    if (VNR!=NULL)
      VNR->event=neweventptr;
```

```
    eventptr temp;

    if (event_queue->head==NULL)
    {
      neweventptr->previous=NULL;
      neweventptr->next=NULL;
      event_queue->head=neweventptr;
      event_queue->tail=neweventptr;
      event_queue->length=1;
    }
    else
    {
       temp=event_queue->head;
       while (temp!=NULL)
       {
          if (t<temp->time)
          {
             neweventptr->next=temp;
             if (temp==event_queue->head)
             {
                event_queue->head=neweventptr;
                neweventptr->previous=NULL;
             }
             else
             {
                temp->previous->next=neweventptr;
                neweventptr->previous=temp->previous;
             }
           temp->previous=neweventptr;
           break;
         }
         else
            temp=temp->next;
       }
       if (temp==NULL)
       {
          neweventptr->previous=event_queue->tail;
          neweventptr->next=NULL;
          event_queue->tail->next=neweventptr;
          event_queue->tail=neweventptr;
       }
       event_queue->length++;
    }
}


//extract an event from the event_queue and update simulation variables
void extract_event(eventptr e, float* t, VNRequest* VNRptr, int* evtype)
{
    if (e==NULL)
    {
      printf("NULL event!!\n");
      system("PAUSE");
      exit(0);
    }

    (*t)=e->time;
    (*VNRptr)=e->VNR;
    (*evtype)=e->eventtype;
    if (e==event_queue->head)
    {
      if (e->next!=NULL)
      {
        event_queue->head=e->next;
        event_queue->head->previous=NULL;
      }
      else
      {
         event_queue->head=NULL;
         event_queue->tail=NULL;
      }
    }
    else if (e==event_queue->tail)
    {
        event_queue->tail=e->previous;
```

```
        event_queue->tail->next=NULL;
    }
    else
    {
       e->next->previous = e->previous;
       e->previous->next = e->next;
    }
    free(e);
}


//delete an event of specific VNR and type from the event_queue
void delete_event(VNRequest VNR, int evtype)
{
    eventptr temp;
    temp=event_queue->head;
    while (temp!=NULL)
       if ((temp->VNR==VNR)&&(temp->eventtype==evtype))
          break;
       else
          temp=temp->next;

     if (temp!=NULL)
     {
       if (temp==event_queue->head)
       {
          if (temp->next!=NULL)
          {
             event_queue->head=temp->next;
             event_queue->head->previous=NULL;
          }
          else
          {
             event_queue->head=NULL;
             event_queue->tail=NULL;
          }
       }
       else if (temp==event_queue->tail)
       {
          event_queue->tail=temp->previous;
          event_queue->tail->next=NULL;
       }
       else
       {
          temp->next->previous=temp->previous;
          temp->previous->next=temp->next;
       }
       free(temp);
     }
}


//find an event regarding a specific VNR and of specific type
eventptr find_event(VNRequest VNR, int evtype)
{
   eventptr temp;
   temp=event_queue->head;
   while (temp!=NULL)
      if ((temp->VNR==VNR)&&(temp->eventtype==evtype))
         return temp;
      else
         temp=temp->next;
   return NULL;
}


//search if there is an event of specific type inside the event_queue
int there_is_event(int evtype)
{
   eventptr temp;
   temp=event_queue->head;
   while (temp!=NULL)
   {
      if (temp->eventtype==evtype)
         break;
```

```
      else
         temp=temp->next;
   }
   if (temp==NULL)
      return 0;
   else
      return 1;
}


//arrival of VNR=> insert VNR to to_serve_queue and schedule future arrivals
void arrival(int id, int router_num, int term_num, int max_link_num, float max_bw, float term_max_cpu, int max_hops)
{
   printf("\nArrival of VNR %d\n",id);
   VNGraph GV=new_VNGraph(id,router_num,term_num,max_link_num,max_bw,term_max_cpu,max_hops);
   VNRequest VNR=new_VNRequest(id,GV,clock_timer,u_space_random(1*Dt,9*Dt));
   printf("VNR %d info:\n",VNR->id);
   printf("routers: %d, terminals: %d, links: %d, max hops: %d\n",router_num,term_num,GV->link_num,max_hops);
   printf("max bandwidth: %.2f, max terminal CPU: %.2f\n",max_bw,term_max_cpu);
   printf("Duration = %.2f sec\n",VNR->duration);
   insert_VNR_in_to_serve_queue(VNR);
   //printf("VNR %d inserted into to_serve_queue\n",VNR->id);
   insert_event(clock_timer+VNR->duration,VNR,3);
   //printf("Abnormal expiration of VNR %d scheduled\n",VNR->id);
   if (id<(max_requests-1))   //if not yet reached request number limit
   {
     //schedule next arrival according to Poisson model
     //(exponentially distributed time spaces between sequential arrivals)
     insert_event((clock_timer-(Dt/requests_per_Dt)*log(u_random())),NULL,0);
     //printf("Arrival of next VNR scheduled\n");
   }
   if (there_is_event(1)==0)  //if next check not scheduled
   {
      //schedule the check of VNR which just arrived
      insert_event(clock_timer+0.000001,VNR,1);
      //printf("Checking of VNR %d scheduled\n",VNR->id);
   }
}


//checking for available mapping
void check_to_serve_queue(VNRequest VNR)
{
   printf("\nChecking VNR %d for available mapping\n",VNR->id);
   float check_time;

   int flib_success;
   time_t time1;
   time_t time2;
   time1=time(NULL);
   VNR->VNM=new_VNM(VNR->VNG,VNR->id);
   VNGraph GsubV=new_GsubV(VNR->VNG);
   flib_calls=0;
   int j=0;
   for (j=0;j<SubG->node_num;j++)
   {
      SubG->Substrate_N[j]->assigned=0;
      SubG->Substrate_N[j]->sliced=0;
   }
   flib_success=vnmFlib(GsubV,VNR->VNM,VNR->VNG,SubG,VNR->id);
   destroy_GsubV(GsubV);
   time2=time(NULL);
   check_time=(float)difftime(time2,time1);
   if (check_time<0.000001)
     check_time=0.00001;

   if ((clock_timer+check_time)>(VNR->arrival_time+VNR->duration))
   {
     printf("VNR %d expired abnormally during mapping stage\n",VNR->id);
     release_resources(SubG,VNR->VNM,VNR->VNG);
     destroy_VNM(VNR->VNM,VNR->VNG);
     delete_event(VNR,3);
     insert_event(clock_timer+check_time,VNR,3);
     printf("Abnormal expiration event of VNR %d rescheduled\n",VNR->id);
   }
```

```
   else if (flib_success==1) //success
   {
      printf("Mapping of VNR %d successful!!!\n",VNR->id);
      delete_event(VNR,3);
      //printf("Abnormal expiration event of VNR %d deleted\n",VNR->id);
      insert_event(clock_timer+check_time,VNR,2);
      //printf("Moving of VNR %d scheduled\n",VNR->id);
      VNR->mapped=1;
   }
   else //failure
   {
      printf("Mapping of VNR %d not successful\n",VNR->id);
      release_resources(SubG,VNR->VNM,VNR->VNG);
      destroy_VNM(VNR->VNM,VNR->VNG);
   }

   //schedule next check
   if ((VNR->next!=NULL)&&(VNR->next->mapped==0))
   {
     insert_event(clock_timer+check_time,VNR->next,1);
     printf("Checking of next VNR %d scheduled\n",VNR->next->id);
   }
   else if ((to_serve_queue->head!=NULL)&&(to_serve_queue->head->mapped==0))
   {
     insert_event(clock_timer+check_time,to_serve_queue->head,1);
     printf("Rechecking of to_serve_queue->head %d scheduled\n",to_serve_queue->head->id);
   }
   else
   {
      printf("Currently no other VNRs to check\n");
   }
}


//serve VNR=> tranfer VNR to serving_queue, schedule its normal expiration
void serve_VNR(VNRequest VNR)
{
   //printf("\nMapping of VNR %d was successful!\n",VNR->id);
   printf("\nReady to serve VNR %d\n",VNR->id);
   VNR->deployment_time=clock_timer;
   transfer_VNR_to_serving_queue(VNR);
   //printf("VNR %d transfered to to_serving_queue\n",VNR->id);
   insert_event(clock_timer+VNR->duration,VNR,4);
   printf("Normal expiration of VNR %d scheduled\n",VNR->id);
   printf("VNR %d is being properly served\n",VNR->id);
}


//abnormal expiration=> delete any events still pending regarding VNR,
//delete VNR form to_serve_queue, obliviate virtual network,
//remove VNR
void abnormal_expiration(VNRequest VNR)
{
   printf("\nVNR %d ready to expire abnormally\n",VNR->id);
   eventptr temp_event=find_event(VNR,1);
   float event_time;
   if (temp_event!=NULL)
   {
     printf("VNR %d was next to be checked and expired before this!\n",VNR->id);
     event_time=temp_event->time;
     delete_event(VNR,1);
     if ((VNR->next!=NULL)&&(VNR->next->mapped==0))
     {
        insert_event(event_time,VNR->next,1);
        printf("Checking of next VNR %d scheduled\n",VNR->next->id);
     }
     else if ((to_serve_queue->head!=NULL)&&(to_serve_queue->head!=VNR)&&(to_serve_queue->head->mapped==0))
     {
        insert_event(event_time,to_serve_queue->head,1);
        printf("Rechecking of to_serve_queue->head %d scheduled\n",to_serve_queue->head->id);
     }
     else if (to_serve_queue->head==VNR)
     {
        printf("Currently no other VNRs to check!\n");
     }
```

```
    }
    delete_event(VNR,2);
    delete_VNR_from_queue(VNR,to_serve_queue);
    printf("VNR %d deleted from to_serve_queue and expired abnormally\n",VNR->id);
    destroy_GV(VNR->VNG);
    free(VNR);
}


//normal expiration=> delete VNR from serving_queue,
//release substrate resources, obliviate the mapping and virtual network,
//remove VNR
void normal_expiration(VNRequest VNR)
{
    printf("\nVNR %d ready to expire normally\n",VNR->id);
    delete_VNR_from_queue(VNR,serving_queue);
    printf("VNR %d deleted from serving_queue and expired normally\n",VNR->id);
    release_resources(SubG,VNR->VNM,VNR->VNG);
    destroy_VNM(VNR->VNM,VNR->VNG);
    destroy_GV(VNR->VNG);
    free(VNR);
}


//take the revenue to cost ratio into accounr
float revenue_to_cost_ratio(VNRequest VNR)
{
    //costs per unit
    float cost_per_CPU=1.0;
    float cost_per_slice=1.0;
    float cost_per_BW=1.0;

    //total mapping costs
    float CPU_cost=0.0;
    float slice_cost=0.0;
    float BW_cost=0.0;
    float cost=0.0;

    //total mapping revenues
    float CPU_rev=0.0;
    float slice_rev=0.0;
    float BW_rev=0.0;
    float rev=0.0;

    //auxiliary variables
    int i=0;
    int j=0;
    int* slice_accounted=calloc(SubG->node_num,sizeof(int));
    for (i=0;i<SubG->node_num;i++)
        slice_accounted[i]=0;

    //costs
    for (i=0;i<VNR->VNG->node_num;i++)
    {
        if (VNR->VNM->node_assign[VNR->VNG->VN_N[i]->id]->type==0)
            CPU_cost=CPU_cost+(VNR->VNG->VN_N[i]->CPU_req)*cost_per_CPU;
        if (slice_accounted[VNR->VNM->node_assign[VNR->VNG->VN_N[i]->id]->id]==0)
        {
            slice_cost+=cost_per_slice;
            slice_accounted[VNR->VNM->node_assign[VNR->VNG->VN_N[i]->id]->id]=1;
        }
    }
    for (i=0;i<VNR->VNG->link_num;i++)
    {
        for (j=0;j<VNR->VNM->link_assign[VNR->VNG->VN_L[i]->id]->path_length;j++)
            BW_cost=BW_cost+(VNR->VNG->VN_L[i]->BW_req)*cost_per_BW;
        for (j=0;j<VNR->VNM->link_assign[VNR->VNG->VN_L[i]->id]->path_length+1;j++)
        {
            if (slice_accounted[VNR->VNM->link_assign[VNR->VNG->VN_L[i]->id]->path_nodes[j]->id]==0)
            {
                slice_cost+=cost_per_slice;
                slice_accounted[VNR->VNM->link_assign[VNR->VNG->VN_L[i]->id]->path_nodes[j]->id]=1;
            }
        }
    }
```

```
    free(slice_accounted);

    //revenues
    CPU_rev=CPU_cost;
    slice_rev=slice_cost;
    for (i=0;i<VNR->VNG->link_num;i++)
    {
       BW_rev=BW_rev+(VNR->VNG->VN_L[i]->BW_req)*cost_per_BW;
    }

    cost=CPU_cost+slice_cost+BW_cost;
    rev=CPU_rev+slice_rev+BW_rev;

    return (rev/cost);
}



//main function for performing Poisson Simulation
void simulation(int te_num, int rt_num, float beta, int hops)
{
    //simulation parameters initialization
    max_requests=20;    //maximum number of requests
    Dt=20.0;            //main time component (for batch means usage)
    requests_per_Dt=5;  //Poisson lamda=pace of arrivals
    clock_timer=0.0;    //main simulation timer

    //auxiliary simulation variables
    eventptr cur_event; //current event to examine
                        //(always the first in event_queue)
    VNRequest cur_VNR;  //current VNR to examine
    int cur_eventtype;  //type of current event
    int events_num=0;   //total number of events occured
    int normal_exp_num=0;  //total number of normal expirations
    int abnormal_exp_num=0; //total number of abnormal expirations
    int req_num=0;  //total number of requests

    //GV parameters
    int id=0;
    int router_num;
    int term_num;
    int max_link_num;
    float max_bw;
    float term_max_cpu;
    int max_hops;

    //R to C ratio
    float R_dia_C_sum=0.0;
    float cur_R_dia_C=0.0;
    float R_dia_C_average=0.0;

    //queue initializations
    //initialize queue of events
    event_queue=malloc(sizeof(struct event_queue_struct));
    if (event_queue==NULL)
    {
       printf("Out of memory!\n");
       leave();
    }
    event_queue->head=event_queue->tail=NULL;
    event_queue->length=0;

    //initialize queue of VNRs waiting to be checked
    to_serve_queue=malloc(sizeof(struct VNR_queue_struct));
     if (to_serve_queue==NULL)
    {
       printf("Out of memory!\n");
       leave();
    }
    to_serve_queue->head=to_serve_queue->tail=NULL;
    to_serve_queue->length=0;

    //initialize queue of VNRs being served
    serving_queue=malloc(sizeof(struct VNR_queue_struct));
     if (serving_queue==NULL)
```

```
{
   printf("Out of memory!\n");
   leave();
}
serving_queue->head=serving_queue->tail=NULL;
serving_queue->length=0;

//initialize substrate
SubG=new_SubstrateGraph(0,30,120,350,100,100,10,50);

//print initial Substrate
FILE *fpsinit;
fpsinit=fopen("substrateinit.gv.txt","w+");
if (fpsinit==NULL)
{
  printf("cannot create file substrateinit\n");
  leave();
}
print_GP(SubG,fpsinit);
fclose(fpsinit);

//schedule initial arrival
insert_event(0.0,NULL,0);

//main loop for event extraction and examination
while ((clock_timer<50*Dt)&&(event_queue->head!=NULL))
{
   //extract first event and update clock
   extract_event(event_queue->head,&clock_timer,&cur_VNR,&cur_eventtype);
   events_num++;
   printf("\nCLOCK=%f\n",clock_timer);

   //arrival
   if (cur_eventtype==0)
   {
      //initialize new GV parameters
      //router_num=(int)roundf(u_space_random(2,8));
      //term_num=4*router_num;
      router_num=rt_num;
      term_num=te_num;
      max_link_num=term_num+router_num*(router_num-1)/2;
      //max_bw=roundf(u_space_random(10,90));
      //term_max_cpu=max_bw;
      max_bw=beta;
      term_max_cpu=beta;
      //max_hops=(int)roundf(u_space_random(2,4));
      max_hops=hops;

      //handle arrival
      arrival(id,router_num,term_num,max_link_num,max_bw,term_max_cpu,max_hops);
      id++;
      req_num++;
   }
   //checking current VNR for mapping (using vnmFlib)
   else if (cur_eventtype==1)
   {
      check_to_serve_queue(cur_VNR);
   }
   //successful mapping=>handle serving
   else if (cur_eventtype==2)
   {
      serve_VNR(cur_VNR);
      cur_R_dia_C=revenue_to_cost_ratio(cur_VNR);
      if (cur_R_dia_C>1.0)
      {
        printf("wrong accounting!\n");
        leave();
      }
      cur_VNR->R_dia_C=cur_R_dia_C;
      R_dia_C_sum=R_dia_C_sum+cur_R_dia_C;
   }
   //abnormal expiration
   else if (cur_eventtype==3)
   {
      abnormal_expiration(cur_VNR);
```

```
            abnormal_exp_num++;
        }
        //normal expiration
        else if (cur_eventtype==4)
        {
            normal_expiration(cur_VNR);
            normal_exp_num++;
        }
    }

    //calculate average R/C
    R_dia_C_average=R_dia_C_sum/req_num;

    //print final Substrate
    FILE *fpsfin;
    fpsfin=fopen("substratefin.gv.txt","w+");
    if (fpsfin==NULL)
    {
        printf("cannot create file substratefin\n");
        leave();
    }
    print_GP(SubG,fpsfin);
    fclose(fpsfin);

    //destroy Substrate
    destroy_GP(SubG);

    printf("\n%d events totally occured\n",events_num);
    printf("%d requests totally processed\n",req_num);

    printf("\n\nTotal simulation time = %f sec\n",clock_timer);
    printf("%d VNRs totally tested\n",id);
    printf("%d VNRs were mapped and expired normally\n",normal_exp_num);
    printf("%d VNRs were not mapped and expired abnormally\n",abnormal_exp_num);
    printf("%d VNRs not mapped - not expired yet\n",to_serve_queue->length);
    printf("%d VNRs successfully mapped - not expired yet\n",serving_queue->length);
    printf("Acceptance ratio = %.2f percent\n",(100*(float)(normal_exp_num+serving_queue->length)/req_num));
    printf("Average R/C ratio = %.2f percent\n",100*R_dia_C_average);
}
```

Κύρια συνάρτηση προγράμματος main_sim.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "simulation_funs.c"

int main()
{
   srand(time(NULL));
   s1=rand();
   srand(time(NULL));
   s2=rand();
   time_t time1;
   time_t time2;
   int single_vs_multiple;
   printf("Type 1 for single VNR, 2 for full-scale Poisson simulation or 0 to exit:\n");
   scanf("%d",&single_vs_multiple);
   if (single_vs_multiple==0)
   {
      printf("BYE");
      exit(0);
   }
   else if (single_vs_multiple==1)
   {
      SubstrateGraph GP;
      GP=new_SubstrateGraph(0,30,70,300,100,100,10,50);
      if (GP==NULL)
      {
         printf("NULL SubstrateGraph! Something went wrong!\n");
         leave();
      }
      else
      {
         //printf("Substrate created and loaded!\n");
         FILE *fpsinit;
         fpsinit=fopen("substrateinit.gv.txt","w+");
         if (fpsinit==NULL)
         {
            printf("cannot create file substrateinit\n");
            leave();
         }
         print_GP(GP,fpsinit);
         fclose(fpsinit);
      }
      VNGraph GV;
      int t_num;
      int r_num;
      float b;
      int hops;
      printf("Give the desired number of terminals:\n");
      scanf("%d",&t_num);
      printf("Give the desired number of routers:\n");
      scanf("%d",&r_num);
      printf("Give the desired beta parameter:\n");
      scanf("%f",&b);
      printf("Give the desired number of hops:\n");
      scanf("%d",&hops);
      int max_link_num=t_num+r_num*(r_num-1)/2;
      GV=new_VNGraph(0,r_num,t_num,max_link_num,b,b,hops);
      if (GV==NULL)
      {
         printf("NULL VNGraph! Something went wrong!\n");
         leave();
      }
      else
      {
         FILE *fpv;
         fpv=fopen("virtual.gv.txt","w+");
         if (fpv==NULL)
         {
            printf("unable to create virtual print_file\n");
            leave();
```

```
        }
        print_GV(GV,fpv);
        fclose(fpv);
    }

    VNGraph GsubV;
    VNMapping VNM;
    GsubV=new_GsubV(GV);
    int req_id=0;
    VNM=new_VNM(GV,req_id);

    flib_calls=0;
    time1=time(NULL);
    int flib_ret=vnmFlib(GsubV,VNM,GV,GP,req_id);
    time2=time(NULL);
    destroy_GsubV(GsubV);
    if (flib_ret==1)
    {
        printf("\nMapping of VN was successful!\n");
        FILE *fpvnmn;
        FILE *fpvnml;
        fpvnmn=fopen("VNMnodes.gv.txt","w+");
        if (fpvnmn==NULL)
        {
            printf("cannot create print_file for node assignments\n");
            leave();
        }
        fpvnml=fopen("VNMlinks.gv.txt","w+");
        if (fpvnml==NULL)
        {
            printf("cannot create print_file for link assignments\n");
            leave();
        }
        print_VNM(VNM,GV,fpvnmn,fpvnml);
        fclose(fpvnmn);
        fclose(fpvnml);
        FILE *fpsfin;
        fpsfin=fopen("substratefin.gv.txt","w+");
        if (fpsfin==NULL)
        {
            printf("cannot create file substratefin\n");
            leave();
        }
        print_GP(GP,fpsfin);
        fclose(fpsfin);
    }
    else
        printf("\nMapping of VN failed!\n");
    printf("\nTotal time for %d calls of flib = %f sec\n",flib_calls,difftime(time2,time1));
    destroy_VNM(VNM,GV);
    destroy_GV(GV);
    destroy_GP(GP);
}
else
{
    srand(time(NULL));
    s1=rand();
    srand(time(NULL));
    s2=rand();
    int t_num;
    int r_num;
    float b;
    int hops;
    printf("Give the desired number of terminals:\n");
    scanf("%d",&t_num);
    printf("Give the desired number of routers:\n");
    scanf("%d",&r_num);
    printf("Give the desired beta parameter:\n");
    scanf("%f",&b);
    printf("Give the desired number of hops:\n");
    scanf("%d",&hops);
    time_t time_start=time(NULL);
    printf("Starting Poisson simulation!\n");
    simulation(t_num,r_num,b,hops);
    time_t time_end=time(NULL);
```

```
        printf("\nReal program time = %f sec\n",difftime(time_end,time_start));
    }
    printf("Exiting normally\n");
    getchar();
    printf("Press <enter> or any key to exit\n");
    getchar();
    return 0;
}
```

Αναφορές

[1] Jens Liscka, Holger Karl, "A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection", in VISA '09 Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, 2009.

[2] Bo Lv., Z. Wang, Tao Huang, J.Chen, Y.Liu, "A Hierarchical Management Architecture for Virtual Network Mapping", 2010 International Conference on Internet Tecnology and Applications, pp. 1- 4, 2010.

[3] Jim Doherty, Neil Anderson, P. D. Maggiora, "Ο οδηγός της Cisco για τη Δικτύωση", 2η αμερικανική έκδοση, εκδόσεις Κλειδάριθμος, σελ. 402-404, 2010

[4] A. Haider, R. Potter, A. Nakao, "Challenges in Resource Allocation in Network Virtualization", in 20th ITC Specialist Seminar, 18.-20. May 2009, Hoi An, Vietnam, 2009.

[5] Wikipedia the free encyclopedia, www.wikipedia.org

[6] N. M. M. K. Chowdhury, M, R. Rahman, R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping", IEEE INFOCOM 2009, pp.783-791, 2009.

[7] M. Yu, Y.Yi, J. Rexford, M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration", ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 17-29, 2008.

[8] W. Szeto, Y. Iraqi, R. Boutaba" A Multi-Commodity Flow Based Approach to Virtual Network Resource Allocation", in IEEE GLOBECOM '03, vol. 6, pp. 3004-3008, 2003.

[9] David G. Andersen, "Theoretical Approaches to Node Assignment", Computer Science Department, Paper 86, http://repository.cmu.edu/compsci/86, 2002.

[10] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a virtual private network: A network design problem for multicommodity flow", in Proceedings of ACM STOC, pp. 389–398, 2001.

[11] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem", ACM SIGCOMM Computer Communication Review, vol. 33, no. 2, pp. 65– 81, April 2003.

[12] Emulab Network Emulation Testbed, http://www.emulab.net/.

[13] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate", Washington University, Tech. Rep. WUCSE-2006-35, 2006.

[14] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components", in Proceedings of IEEE INFOCOM, 2006.

[15] J. Fan and M. Ammar, "Dynamic topology configuration in service overlay networks - a study of reconfiguration policies", in Proceeding of IEEE INFOCOM, 2006.

[16] H. Frank, I. T. Frisch, "Communication, Transmission, and Transportation Networks", Addison-Wesley Series in Electrical Engineering, Addison-Wesley Publishing Company, pp. 56-73, 1971.

[17] F. Balonneau, O. du Merle, J. -P, Vial, "Solving Large-Scale Linear Multicommodity Flow Problems with an Active Set Strategy and Proximal-ACCPM", in Operations Research Journal, vol. 54, no. 1, pp. 184-197, 2006.

[18] Zheng Wang, Jon Crowcroft, "Bandwidth-Delay Based Routing Algorithms", in IEEE GLOBECOM '95, vol.3, pp. 2129-2133, 1995.

[19] I. Houidi, W. Louati, and D. Zeghlache, “A distributed virtual network mapping algorithm”, in Proceedings of IEEE ICC, pp. 5634–5640, 2008.

[20] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, “Algorithms for provisioning virtual private networks in the hose model”, in IEEE/ACM Transactions on Networking, vol. 10, no. 4, pp. 565–578, 2002.

[21] P. Raghavan and C. D. Tompson, “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”, Combinatorica, vol. 7, no. 4, pp. 365–374, 1987.

[22] Planetlab, http://www.planet-lab.org/.

[23] A Virtual Network Infrastructure (VINI), http://www.vini-veritas.net/.

[24] M.R. Garey and D.S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, Freeman, 1979.

[25] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, “An Improved Agorithm for Matching Large Graphs”, 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, pages 149-159, 2001.

[26] Α.Ν. Σταφυλοπάτης, “Ανάλυση επίδοσης υπολογιστικών συστημάτων”, Ε.Μ.Π., Τμήμα Η.Μ.Μ.Υ., Τομέας Πληροφορικής, 1996.

[27] B. W. Kernighan, D. M. Ritchie, “Η γλώσσα προγραμματισμού C”, 2η έκδοση, Prentice Hall Software Series, εκδόσεις Κλειδάριθμος, 2006