ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ Η/Υ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ & ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

ΥΠΟΛΟΓΙΣΤΩΝ

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ ΕΝΟΣ ΟΛΟΚΛΗΡΩΜΕΝΟΥ ΜΕΤΑΓΩΓΕΑ ETHERNET ΜΕ ΚΟΙΝΟΧΡΗΣΤΗ ΜΝΗΜΗ ΚΑΙ ΟΥΡΕΣ ΕΞΟΔΟΥ

ΠΑΠΠΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εξεταστική Επιτροπή:

Πνευματικάτος Διονύσιος (επιβλέπων)

Δόλλας Απόστολος

Κουτσάκης Πολυχρόνης

Χανιά, Σεπτέμβριος 2010

στους γονείς μου

Περιεχόμενα

Εικόνες

1 Εισαγωγή

1.1 Γενικά

Στις μέρες μας το internet είναι ένας από τους πιο ραγδαία αναπτυσσομένους τεχνολογικούς τομείς. Το internet είναι ένα παγκόσμιο δίκτυο επικοινωνίας, το οποίο διασυνδέει δισεκατομμύρια υπολογιστές. Η τεχνολογία του είναι κυρίως βασισμένη στην διασύνδεση επιμέρους δικτύων ανά τον κόσμο μέσω της εφαρμογής πολυάριθμων τεχνολογικών πρωτοκόλλων.

Η πρώτη εμφάνιση ενός δικτύου υπολογιστών έγινε στα τέλη της δεκαετίας του 1960, με το δίκτυο ARPANET που συνέδεε τέσσερις υπολογιστές ανάμεσα σε τέσσερα αμερικανικά πανεπιστήμια. Από τότε μέχρι και σήμερα το διαδίκτυο αναπτύσσεται με ραγδαίους ρυθμούς, ενώ αυτή η ανάπτυξη αναμένεται να συνεχιστεί καθώς νέες τεχνολογίες εμφανίζονται συνεχώς, παρέχοντας στους χρήστες όλο και περισσότερες ευκολίες. Ο μέσος καταναλωτής έχει την δυνατότητα να απολαμβάνει υπηρεσίες φωνής, βίντεο και internet, όχι μόνο χρησιμοποιώντας τον προσωπικό υπολογιστή του, αλλά και ένα πλήθος άλλων συσκευών που μπορούν να συνδεθούν στο διαδίκτυο.

Η ραγδαία ανάπτυξη, όμως, του internet και γενικότερα των δικτύων των υπολογιστών, για να αποδώσει στο μέγιστο και παράλληλα να μπορεί να θεωρείται αξιόπιστη, απαιτεί την ύπαρξη ενός συνόλου συσκευών οι οποίες θα διασυνδέουν σωστά τα επιμέρους τμήματα των δικτύων, εφαρμόζοντας κάποιους κοινά αποδεκτούς κανόνες επικοινωνίας.

Οι συσκευές οι οποίες αποτελούν το δομικό λίθο διασύνδεσης των δικτύων είναι τα Switches (Μεταγωγείς) και τα Routers (Δρομολογητές). Η χρήση των συσκευών αυτών παρέχει αδιάλειπτη σύνδεση δεδομένων, ενώ, παράλληλα δίνει λύση στον ανταγωνισμό εξόδου, το βασικότερο πρόβλημα κάθε πολύπλευρης (multiparty) επικοινωνίας. Οι μοντέρνοι μεταγωγείς

υψηλών επιδόσεων πρέπει να παρέχουν συνολική διαμεταγωγή της τάξεως του terabit, ανά δευτερόλεπτο κάτι το οποίο αποτελεί πρόκληση τόσο για την αρχιτεκτονική, όσο και για την τεχνολογία υλοποίησής τους.

Το ρόλο των κανόνων επικοινωνίας διαδραματίζουν τα πρωτόκολλα επικοινωνίας, την εξασφάλιση, δηλαδή, των σωστών συνθηκών για την ομαλή, αδιάκοπη και αξιόπιστη λειτουργία των δικτύων. Αξίζει να σημειωθεί, ότι οι πρώτες γενεές των πρωτοκόλλων ήταν προσαρτημένες στα τότε λειτουργικά συστήματα των υπολογιστών, κάτι που με την εξέλιξη της τεχνολογίας τόσο σε επίπεδο λογισμικού, όσο και σε επίπεδο υλικού ξεπεράστηκε, με αποτέλεσμα πολύ σύντομα η υλοποίηση τους να στραφεί και σε άλλες κατευθύνσεις.

1.2 Ανάπτυξη μεταγωγέων

Η συνεχόμενη αύξηση των αναγκών για δίκτυα υψηλών ταχυτήτων, έχει κάνει τους μεταγωγείς ένα από τα πιο σημαντικά στοιχεία για την δημιουργία δρομολογητών υψηλών επιδόσεων. Επομένως, η ερευνητική και εμπορική δραστηριότητα γύρω από την τεχνολογία των μεταγωγέων είναι ιδιαίτερα αυξημένη.

Σημαντικό ρόλο στην σχεδίαση, ανάλυση και εφαρμογή των τεχνολογιών μεταγωγέων διαδραματίζει η προσομοίωσή της λειτουργίας τους στο στάδιο της ανάπτυξης, σαν μια οικονομική και αποδοτική μέθοδος. Οι προσομοιώσεις χρησιμοποιούνται για τον έλεγχο της λειτουργικότητας, για την εκτίμηση της απόδοσης και γενικότερα για την καθοδήγηση της διαδικασίας σχεδίασης νέων μεταγωγέων.

Ένας μεταγωγέας μπορεί να προσομοιωθεί με χρήση γλωσσών προγραμματισμού γενικής χρήσης, με την χρήση προγραμμάτων προσομοίωσης, όπως το Sim του πανεπιστημίου Stanford (http://yuba.stanford.edu/tools/SIM/) ή σε επίπεδο hardware με μια γλώσσα περιγραφής υλικού όπως η VHDL.

Οι παραπάνω προσεγγίσεις επικεντρώνονται σε χαρακτηριστικά του μεταγωγέα όπως η δομή του ή ο αλγόριθμος δρομολόγησης, χωρίς να μπορούν να προσομοιώσουν την απόδοση του μεταγωγέα σε διάφορα

δικτυακά σενάρια, πχ. Internet. Έτσι, κάθε σύστημα που αναπτύσσεται με τον συγκεκριμένο τρόπο πρέπει να περάσει από μια διαδικασία αξιολόγησης και επιβεβαίωσης της λειτουργίας του. Αυτή η διαδικασία περιλαμβάνει την δημιουργία δεδομένων αναφοράς (δεδομένα εισόδου και τα αντίστοιχα αναμενόμενα δεδομένα εξόδου) και στην συνέχεια τον έλεγχο του συστήματος με βάση αυτά, με στόχο την επιβεβαίωση της ορθής λειτουργίας του υλικού, κάτι που αποτέλεσε το αρχικό θέμα αυτής της εργασίας όπως θα περιγραφεί στην παράγραφο 1.4.

1.3 Προσομοιωτές Δικτύων

Για την εκτίμηση της απόδοσης σε ένα δίκτυο είναι απαραίτητο να μπορούμε να προσομοιώσουμε τοπολογίες δικτύων και σενάρια κίνησης δεδομένων στο δίκτυο, κάτι που είναι δύσκολο να εφαρμοστεί και έχει αρκετά υψηλές απαιτήσεις σε υπολογιστική ισχύ. Οι υπάρχοντες προσομοιωτές δικτύων παρέχουν ένα ευρύ φάσμα λειτουργιών που καλύπτουν τις παραπάνω απαιτήσεις. Επομένως, είναι απαραίτητη η χρήση κάποιου προσομοιωτή δικτύου για την προσομοίωση και εκτίμηση της απόδοσης του μεταγωγέα σε ένα δίκτυο.

Ένα από τα πλέον διαδεδομένα συστήματα προσομοίωσης για έρευνα πάνω σε δίκτυα είναι το Network Simulator - ns-2 (http://www.isi.edu/nsnam/ns/). Η δυνατότητα τροποποίησης της σχεδίασης του (προσθήκη επιπλέον λειτουργικότητας) σε συνδυασμό με την πλήρη βιβλιογραφία, μας οδήγησαν στο να επιλέξουμε το NS2 για την υλοποίηση της προσομοίωσης ενός μεταγωγέα.

Το σύστημα NS2 παρέχει λειτουργικότητα για προσομοίωση ενός μεγάλου φάσματος εφαρμογών, πρωτοκόλλων δικτύων, τύπων δικτύων και μοντέλων κίνησης δεδομένων σε ένα δίκτυο. Όλα αυτά είναι «αντικείμενα» που μπορεί να προσομοιώσει το σύστημα. Για παράδειγμα, υποστηρίζει προσομοιώσεις διαφόρων τύπων κόμβων, συνδέσεων, ουρών, πακέτων ή πρωτοκόλλων. Όμως το σύστημα NS2 δεν υποστηρίζει εύκολα παραμετροποιήσιμους μεταγωγείς. Ένας κόμβος μπορεί να συνδεθεί με ένα πλήθος άλλων κόμβων και να λειτουργήσει σαν δρομολογητής πακέτων, ο

οποίος όμως μπορεί να παρομοιαστεί με μια σήραγγα στην οποία εισέρχονται πακέτα, και χωρίς καμία προσωρινή αποθήκευση ή δρομολόγησή τους, εξέρχονται και οδηγούνται στον επόμενο κόμβο. Η διαδικασία αυτή εκτελείται από όλους τους ενδιάμεσους κόμβους σε μία τοπολογία, έως ότου τα πακέτα φτάσουν στον κόμβο παραλήπτη. Για την υποστήριξη και προσομοίωση ενός μεταγωγέα, έπρεπε να δημιουργηθεί μια επέκταση στην βιβλιοθήκη αντικειμένων του NS2. Ο προσομοιωτής δικτύων NS2 είναι ανοιχτό λογισμικό, ελεύθερο για τροποποίηση από τον χρήστη ενώ επιπλέον παρέχεται και η απαραίτητη βιβλιογραφία για την ανάπτυξη επεκτάσεων στο σύστημα με χρήση των ήδη υπαρχόντων αντικειμένων του συστήματος.

Υπάρχει ήδη ένα πλήθος επεκτάσεων ή τροποποιήσεων του NS2 διαθέσιμο προς χρήση. Η παρούσα εργασία παρέχει μια επέκταση στο πεδίο της σχεδίασης, προσομοίωσης και ανάλυση απόδοσης μεταγωγέων με χρήση του NS2.

1.4 Περιγραφή και συνεισφορά εργασίας

Ο αρχικός στόχος της εργασίας ήταν η δημιουργία δεδομένων αναφοράς για έναν μεταγωγέα Ethernet με ουρές εξόδου, που θα υλοποιούνταν σε γλώσσα VHDL, και στην συνέχεια επιβεβαίωση της σωστή λειτουργίας του υλικού καθώς και η εξαγωγή συμπερασμάτων κατά την προσομοίωση της λειτουργίας του υπό διάφορες συνθήκες. Η υλοποίηση του μεταγωγέα αποτελούσε θέμα χρονικά παράλληλης μεταπτυχιακής διατριβής, η οποία όμως δεν ολοκληρώθηκε. Αναπόφευκτα, έγινε τροποποίηση του θέματος της εργασίας.

Έτσι, στόχος αυτής της εργασίας είναι η μοντελοποίηση και προσομοίωση λειτουργίας ενός ολοκληρωμένου μεταγωγέα Ethernet, όπως αυτός έχει περιγραφεί στην μεταπτυχιακή διατριβή του κ. Ερμή Ιωάννη, με τίτλο «Σχεδιασμός και υλοποίηση ενός 8x8 Ethernet switch». Η παρούσα εργασία μπορεί μελλοντικά να χρησιμοποιηθεί για την σύγκριση αντίστοιχων αποτελεσμάτων προσομοίωσης του υλοποιημένου σε VHDL μεταγωγέα, με στόχο την επιβεβαίωση της σωστής λειτουργίας του. Με μικρές τροποποιήσεις, η εργασία μπορεί να χρησιμοποιηθεί για την προσομοίωση

οποιασδήποτε σχεδίασης μεταγωγέα, ενώ μπορεί επίσης να χρησιμοποιηθεί για την υλοποίηση περισσότερο περίπλοκων δικτυακών προσομοιώσεων. Παράλληλα, αποτελεί βελτίωση για τον προσομοιωτή δικτύων NS-2, ο οποίος στην βασική του έκδοση δεν υποστηρίζει παραμετροποιήσιμους μεταγωγείς.

Η ανάπτυξη της εργασίας έγινε με χρήση του προσομοιωτή NS-2 εγκατεστημένου στο περιβάλλον προσομοίωσης UNIX/Linux σε Windows, Cygwin. Για την συγγραφή του κώδικα, χρησιμοποιήθηκε το περιβάλλον ανάπτυξης Eclipse IDC για C/C++. Τέλος, για την επεξεργασία των αποτελεσμάτων και την εξαγωγή γραφικών παραστάσεων χρησιμοποιήθηκε το εργαλείο Matlab.

1.5 Η οργάνωση αυτής της αναφοράς

Το επόμενο κεφάλαιο αποτελεί μια συνοπτική περιγραφή του τρόπου λειτουργίας του συστήματος NS2 και παρουσίαση των κυριότερων δομικών του στοιχειών.

Στο κεφάλαιο 3 περιγράφεται η υλοποίηση του μεταγωγέα, που περιελάμβανε την δημιουργία νέων κλάσεων για το NS2 και την τροποποίηση υπαρχόντων. Επίσης, παρουσιάζεται ο τρόπος ανάπτυξης προσομοιώσεων.

Στο κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα των προσομοιώσεων για διαφορετικές περιπτώσεις χρήσης του δικτυού. Παράλληλα γίνονται παρατηρήσεις επί των γραφημάτων που προέκυψαν από την συλλογή των αποτελεσμάτων.

Το κεφάλαιο 5 αναφέρεται στις πιθανές μελλοντικές επεκτάσεις της εργασίας, με σκοπό την βελτίωση της απόδοσης του μοντελοποιημένου μεταγωγέα.

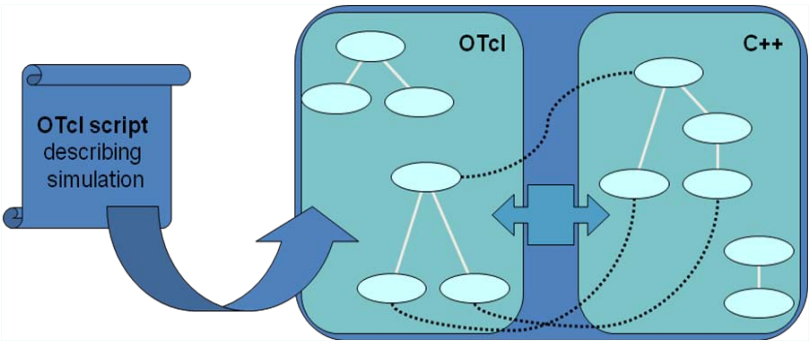
2 Εισαγωγή στην δομή του προσομοιωτή δικτύων NS-2

2.1 Γενικά

Οι επόμενες παράγραφοι περιέχουν μια συνοπτική περιγραφή του τρόπου λειτουργίας του συστήματος NS2.

Το σύστημα NS2 είναι ένας αντικειμενοστραφής προσομοιωτής δικτύων, ο οποίος είναι γραμμένος σε γλώσσα C++ και Otcl (αντικειμενοστραφής έκδοση της γλώσσας TCL). Η TCL (Tool Command Language) είναι μία γλώσσα συγγραφής (scripting language), που πρωτοεμφανίστηκε στα τέλη της δεκαετίας του 1980, με σκοπό την ενσωμάτωση σε εφαρμογές . Χρησιμοποιείται για ταχεία ανάπτυξη λογισμικού (Rapid Application Development), scripted εφαρμογές, Γραφικά Περιβάλλοντα Χρήστη (GUIs), σε ενσωματωμένα συστήματα ενώ είναι πολύ δημοφιλής σε εφαρμογές προσομοιώσεων δικτύων.

Ο προσομοιωτής NS2 αποτελείται από μια ιεραρχία κλάσεων C++ και από μία αντίστοιχη ιεραρχία κλάσεων OTcl. Οι δύο αυτές ιεραρχίες κλάσεων συσχετίζονται μεταξύ τους, αφού ουσιαστικά υπάρχει μια ένα προς ένα αντιστοιχία ανάμεσα σε κάθε κλάση OTcl και C++. Έτσι, ένα δικτυακό πρόβλημα μπορεί να εκφραστεί μόνο με την Otcl, μόνο με την C++ ή και με συνδυασμό των δύο. Η Otcl χρησιμοποιείται για την διεπαφή δημιουργίας προσομοιώσεων και συνήθως για την παραμετροποίηση του προβλήματος. Η εικόνα 1 απεικονίζει σχηματικά την αντιστοίχηση C++ και OTcl.



Εικόνα 1: Αντιστοίχηση C++ και OTcl στην δομή του συστήματος NS2

Το σύστημα NS2 χρησιμοποιεί δύο γλώσσες προγραμματισμού γιατί εκτελεί δύο διαφορετικές λειτουργίες:

1. Οι λεπτομερείς προσομοιώσεις πρωτοκόλλων δικτύων απαιτούν μια γλώσσα προγραμματισμού που να μπορεί να διαχειριστεί bytes, επικεφαλίδες πακέτων και να εφαρμόσει αλγόριθμους που πρέπει να τρέξουν για μεγάλες δομές δεδομένων. Για αυτές τις λειτουργίες είναι απαραίτητο να χρησιμοποιηθεί μια γρήγορη γλώσσα προγραμματισμού. Η C++ είναι μια τέτοια γρήγορη γλώσσα που όμως απαιτεί χρόνο για αλλαγές στον κώδικά (αποσφαλμάτωση, recompile, re-run). Για αυτό χρησιμοποιείται για την δημιουργία και λεπτομερή εφαρμογή πρωτοκόλλων.
2. Ένα μεγάλο κομμάτι της έρευνας πάνω σε δίκτυα εμπλέκει την ελαφριά μεταβολή παραμέτρων και ρυθμίσεων στο δίκτυο, ή την ανάγκη για προσομοίωση πολλών διαφορετικών σεναρίων. Σε αυτές τις περιπτώσεις, ο χρόνος που θα χρειαστεί για την αλλαγή του μοντέλου και την νέα προσομοίωση είναι σημαντικός. Από την στιγμή που κατασκευή ενός σεναρίου προσομοίωσης τρέχει μόνο μια φορά (στην αρχή), ο χρόνος που απαιτείται για αυτή την λειτουργία δεν είναι σημαντικός. Η Tcl είναι πιο αργή από την C++ αλλά μπορεί να τροποποιηθεί με ευκολία (ακόμα και κατά την διάρκεια της προσομοίωσης), κάνοντάς την ιδανική για την διασύνδεση του προσομοιωτή με τον χρήστη και την διαμόρφωση των προσομοιώσεων.

2.2 Κλάσεις Otcl

Οι κλάσεις Otcl που χρησιμοποιούνται από το σύστημα ns2 είναι οι παρακάτω:

- Η κλάση ClassTcl ενσωματώνει τον μεταγλωττιστή Otcl και περιέχει τις μεθόδους εκείνες που χρησιμοποιεί ο κώδικας C++ ώστε να έχει πρόσβαση στον μεταγλωττιστή.
- Η TclObject είναι η βασική κλάση για όλες σχεδόν τις κλάσεις του συστήματος (Tcl και C++). Κάθε αντικείμενο της TclObject δημιουργείται από τον χρήστη μέσα από τον μεταγλωττιστή Tcl. Για κάθε αντικείμενο Otcl, ένα αντίστοιχο C++ αντικείμενο δημιουργείται. Τα δύο αντικείμενα είναι στενά συνδεδεμένα μεταξύ τους.
- Η κλάση TclClass ορίζει την ιεραρχία της κλάσεων Otcl καθώς και τις μεθόδους μέσω των οποίων ο χρήστης δημιουργεί στιγμιότυπα αντικειμένων Otcl (TclObjects).
- Η κλάση TclCommand ορίζει απλές καθολικές (global) εντολές για τον μεταγλωττιστή Otcl.
- Η κλάση EmbeddedTcl περιέχει μεθόδους για την χρήση δομικών εντολών υψηλού επιπέδου, καθιστώντας ευκολότερες τις προσομοιώσεις.
- Η classInstVar περιέχει μεθόδους για την διασύνδεση μεταβλητών-μελών κλάσεων C++ με τις αντίστοιχες μεταβλητές-μέλη των Otcl κλάσεων. Μέσω αυτής της διασύνδεσης, η τιμή μιας μεταβλητής μπορεί να τροποποιηθεί είτε μέσω του μεταγλωττιστή είτε από τον κώδικα C++. Αυτό γίνεται με την κλήση των αντίστοιχων συναρτήσεων get() και set() σε επίπεδο Otcl και C++ κάθε φορά.

2.3 Προσομοιωτής (Simulator)

Ο προσομοιωτής του συστήματος NS2 περιγράφεται από την κλάση Tcl `Simulator`. Παρέχει ένα σύνολο διεπαφών για την διαχείριση μιας προσομοίωσης και για την επιλογή του τύπου του χρονοπρογραμματιστή γεγονότων (scheduler) που θα χρησιμοποιηθεί για την εκτέλεση της

προσομοίωσης. Ο κώδικας μιας προσομοίωσης, αρχικά περιλαμβάνει την δημιουργία ενός στιγμιότυπου της κλάσης αυτής και στην συνέχεια κλήσεις σε μεθόδους που δημιουργούν κόμβους, συνδέσεις και άλλα χαρακτηριστικά των προσομοιώσεων.

Ως γεγονός ορίζεται η αποστολή ή η λήψη ενός πακέτου από έναν κόμβο του δικτύου. Η εκτέλεση της προσομοίωσης γίνεται σε βήματα, κατά τα οποία ο προσομοιωτής αναγνωρίζει το αμέσως επόμενο γεγονός που θα συμβεί στο δίκτυο, εκτελεί την ενέργεια του γεγονότος (αποστολή/ λήψη πακέτου, κτλ.) και στην συνέχεια επιστρέφει για αναγνωρίσει ποιο θα είναι το επόμενο γεγονός. Η χρονική μονάδα που χρησιμοποιεί ο προσομοιωτής του NS2 είναι το δευτερόλεπτο (και οι υποδιαιρέσεις του) ενώ το ρολόι της προσομοίωσης απεικονίζεται σε μια μεταβλητή τύπου double. Σύμφωνα με την βιβλιογραφία του συστήματος NS2 μπορούν να απεικονιστούν με ακρίβεια χρονικές υποδιαιρέσεις έως και 2^{40}.

Στην έκδοση του συστήματος που χρησιμοποιήθηκε, ο προσομοιωτής δεν μπορεί να εκτελέσει ταυτόχρονα δύο γεγονότα. Αν ένα γεγονός έχει οριστεί ότι πρέπει να εκτελεστεί, ενώ το αμέσως προηγούμενο δεν έχει ολοκληρωθεί τότε το σύστημα χρησιμοποιεί πολιτική FIFO - το πρώτο γεγονός που προγραμματίσθηκε είναι αυτό που θα ολοκληρωθεί πρώτο. Έτσι, διασφαλίζεται ο παραπάνω περιορισμός και δεν επιτρέπεται «μερική» ολοκλήρωση ενός γεγονότος ή ακύρωσή του.

Ένα γεγονός χαρακτηρίζεται από το χρόνο στον οποίο εκτελείται (event time), το id που το χαρακτηρίζει μοναδικά (event id) καθώς και από την συνάρτηση που το διαχειρίζεται (handler function). Υπάρχουν δύο τύποι αντικειμένων που πηγάζουν από την βασική κλάση `Event`:

- Packet-events: γεγονότα που σχετίζονται με πακέτα (θα περιγραφούν λεπτομερέστερα στην παράγραφο 2.5)
- “At-events”: γεγονότα που σχετίζονται με αλλαγές στην τοπολογία του δικτύου.

Ένα γεγονός του δεύτερου τύπου αντιστοιχεί στην εκτέλεση κάποιας λειτουργίας σε συγκεκριμένο χρόνο, και χρησιμοποιήθηκε συχνά στις προσομοιώσεις που έγιναν. Για παράδειγμα:

```
set ns [new Simulator]
$ns use-scheduler Heap
```

```
$ns at 300.5 "finish"
```

Το παραπάνω κομμάτι Tcl κώδικα, αρχικά δημιουργεί ένα αντικείμενο προσομοιωτή, στην συνέχεια ορίζει τον τύπο του χρονοπρογραμματιστή που θα χρησιμοποιηθεί (heap σε αυτή την περίπτωση), και στην τελευταία γραμμή ορίζει την εκτέλεση της συνάρτησης “finish” (at-event) στον χρόνο 300.5 (seconds)

Τα “at-events” υλοποιούνται σαν γεγονότα των οποίων ο διαχειριστής (handler) ουσιαστικά είναι η εκτέλεση κάποιας εντολής από τον μεταγλωττιστή Tcl.

Ο ορισμός της κλάσης `Event` βρίσκεται στο αρχείο *~ns/scheduler.h*.

2.4 Χρονοπρογραμματιστές (Schedulers)

Υπάρχουν τέσσερις τύποι χρονοπρογραμματιστών (schedulers), ο καθένας από τους οποίους υλοποιείται με χρήση διαφορετικών δομών δεδομένων: ένας χρονοπρογραμματιστής απλά συνδεδεμένων λιστών, ένας χρονοπρογραμματιστής σωρού (heap), ένας χρονοπρογραμματιστής ουράς ημερολογίου (Calendar queue) και ένας χρονοπρογραμματιστής πραγματικού χρόνου.

- *List scheduler*: Υλοποίηση του χρονοπρογραμματιστή με μία απλά συνδεδεμένη λίστα. Η λίστα κρατάει τα γεγονότα σε χρονολογική σειρά (από το παλαιότερο στο νεότερο), έτσι κάθε εγγραφή ή διαγραφή από την λίστα γίνεται μόνο αφού βρεθεί το γεγονός που αντιστοιχεί σε κάθε περίπτωση. Με αυτόν τον τρόπο διασφαλίζεται η χρήση πολιτικής FIFO.
- *Heap scheduler*: Υλοποιεί τον χρονοπρογραμματιστή χρησιμοποιώντας μία δομή heap. Αυτός ο τρόπος υλοποίησης υπερτερεί της υλοποίησης με λίστα για μεγάλο αριθμό γεγονότων, καθώς οι χρόνοι εισαγωγής και διαγραφής είναι της τάξης $O(\log N)$ για *N* γεγονότα.
- *Calendar queue Scheduler:* Υλοποιεί τον χρονοπρογραμματιστή με την χρήση μιας δομής δεδομένων ανάλογης με ένα ετήσιο επιτραπέζιο

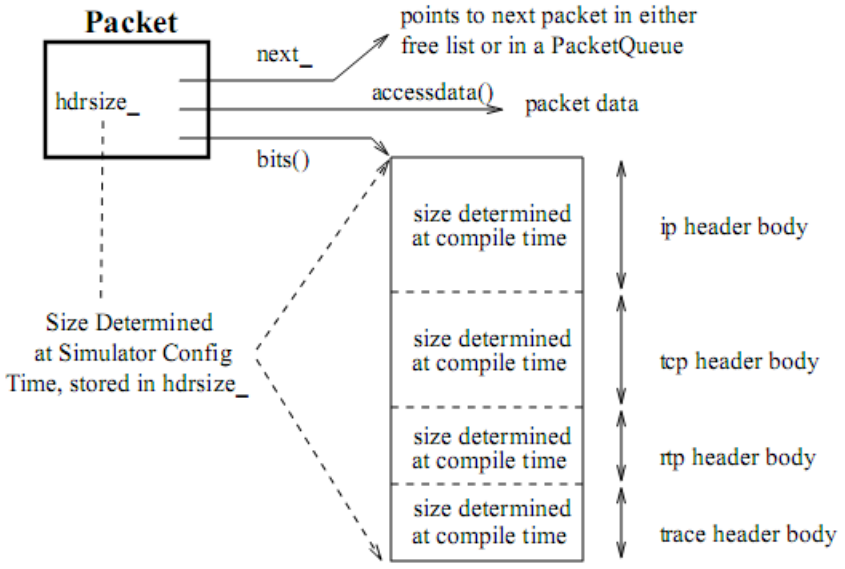
ημερολόγιο, σύμφωνα με την οποία γεγονότα που συμβαίνουν την ίδια μέρα και μήνα αλλά σε διαφορετικά έτη, μπορούν να αποθηκευθούν στην ίδια μέρα.

- *Real-Time Scheduler*: Ο χρονοπρογραμματιστής πραγματικού χρόνου, έχει στόχο να συγχρονίσει την εκτέλεση των γεγονότων σε πραγματικό χρόνο. Όμως προσωρινά βρίσκεται ακόμα σε στάδιο ανάπτυξης και μπορεί να χρησιμοποιηθεί μόνο σε δίκτυα με σχετικά μικρούς ρυθμούς κίνησης δεδομένων στο δίκτυο

Κατά την εκτέλεση των προσομοιώσεων χρησιμοποιήθηκε ο χρονοπρογραμματιστής σωρού (heap scheduler), καθώς μας ενδιέφερε η απόδοση της προσομοίωσης δηλαδή ο απαιτούμενος χρόνος για την ολοκλήρωση κάθε προσομοίωσης.

2.5 Πακέτα (Packets)

Όπως σε κάθε δίκτυο επικοινωνιών η μεταφορά δεδομένων γίνεται με χρήση πακέτων, έτσι και το σύστημα NS2 προσομοιώνει πακέτα, για την ανταλλαγή δεδομένων ανάμεσα στα αντικείμενα της προσομοίωσης. Υπάρχουν τέσσερις κλάσεις C++ σχετικές με την διαχείριση των πακέτων και των επικεφαλίδων τους, οι `Packet`, `p_info`, `PacketHeader` και `PacketHeaderManager`.



Εικόνα 2: Η δομή ενός αντικειμένου Packet

Η κλάση Packet ορίζει τον τύπο όλων των πακέτων στην προσομοίωση. Είναι υποκλάση της `Event`, έτσι τα πακέτα μπορούν εύκολα να χρονοπρογραμματιστούν (πχ. για μια καθυστερημένη παράδοση σε μια ουρά). Η μορφή της απεικονίζεται στην εικόνα 2 και περιέχει:

- σύνδεση του πακέτου με μία λίστα πακέτων ίδιου τύπου
- μια αναφορά σε μια δομή που περιέχει τις πληροφορίες επικεφαλίδας του πακέτου, διαφορετική για κάθε πρωτόκολλο δικτύου, καθώς και
- μια αναφορά σε ένα χώρο μνήμης που αποτελεί τα δεδομένα του πακέτου.

Η υλοποίηση νέων πρωτοκόλλων, γίνεται με τον ορισμό νέου τύπου επικεφαλίδας πακέτων ή με την προσθήκη νέων πεδίων πληροφορίας σε υπάρχον τύπο επικεφαλίδας.

Η κλάση `packet_info` χρησιμοποιείται για την αντιστοίχηση των αριθμητικών τιμών των διάφορων τύπων πακέτων με τα συμβολικά τους ονόματα.

Η κλάση `PacketHeader` παρέχει μια βασική κλάση για κάθε επικεφαλίδα πακέτου της προσομοίωσης. Κάθε πακέτο μπορεί να αντιστοιχίζεται σε ένα πλήθος επικεφαλίδων, όμως όλα τα πακέτα κατά την προσομοίωση έχουν ένα κοινό τύπο επικεφαλίδας ο οποίος ορίζεται από την δομή `hdr_cmn`. Η δομή αυτή περιέχει βασικές πληροφορίες για κάθε πακέτο όπως, τον τύπου του πακέτου (`ptype_`), το μέγεθός του (`size_`) σε bytes και άλλες, οι οποίες χρησιμοποιήθηκαν εκτενώς στην υλοποίηση της εργασίας.

Η κλάση `PacketHeaderManager` χρησιμοποιείται για τον έλεγχο της συλλογής των τρεχόντων επικεφαλίδων που χρησιμοποιούνται στην προσομοίωση.

Όλα τα παραπάνω περιγράφονται στα αρχεία *~ns/tcl/lib/ns-lib.tcl*, *~ns/tcl/lib/ns-packet.tcl*, και *~ns/packet.{cc,h}*.

2.6 Κόμβοι (Nodes)

Όπως αναφέρθηκε παραπάνω, η κλάση `Simulator`, χρησιμοποιείται για την αρχικοποίηση και έλεγχο της προσομοίωσης. Η κλάση περιέχει

βασικές συναρτήσεις για την δημιουργία και την διαχείριση κόμβων της τοπολογίας του δικτύου. Η κλάση `Node`, παρέχει συναρτήσεις για την διαχείριση των επιμέρους χαρακτηριστικών κάθε κόμβου ανάλογα με τον τύπο του.

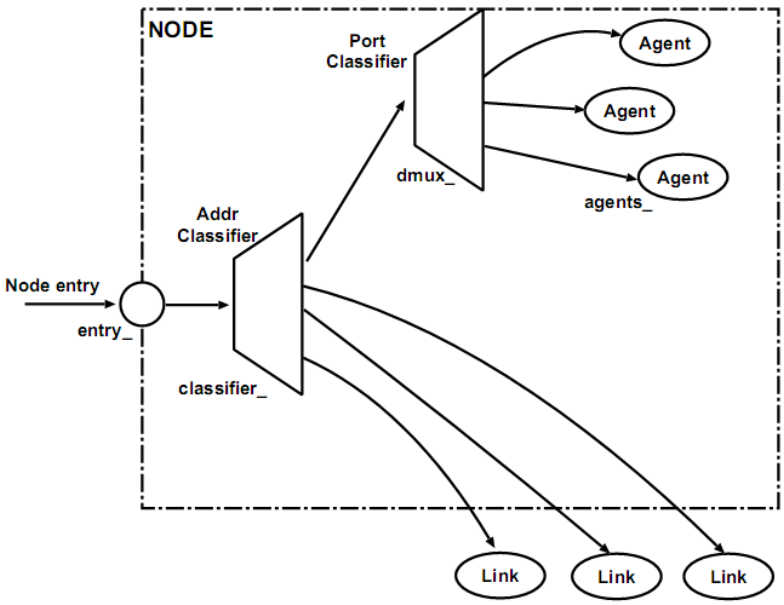
Οι μέθοδοι και οι συναρτήσεις που περιγράφονται παρακάτω βρίσκονται στα αρχεία:

~ns/tcl/lib/ns-lib.tcl, ~ns/tcl/lib/ns-node.tcl, ~ns/tcl/lib/ns-rtmodule.tcl, ~ns/rtmodule.cc, ~ns/rtmodule.h, ~ns/classifier.cc, ~ns/classifier.h, ~ns/classifier-addr.cc.

Ένας νέος κόμβος για το δίκτυο δηλώνεται ως εξής:

```
set ns [new Simulator]
$ns node
```

Η χρήση της συνάρτησης `node` δημιουργεί έναν κόμβο, ο οποίος αποτελείται από απλούστερα αντικείμενα τύπου ταξινομητή (classifier), τα οποία θα επεξηγηθούν στην συνέχεια. Ο κόμβος είναι ένα αυτόνομο αντικείμενο Tcl, όπως και τα συστατικά του στοιχεία. Η τυπική δομή ενός κόμβου, όπως δίνεται από την βιβλιογραφία του NS2, φαίνεται στην εικόνα 3.



Εικόνα 3: Η εσωτερική δομή ενός unicast κόμβου

Ο κάθε κόμβος (node) αποτελείται από δύο αντικείμενα Tcl (TclObjects):

α) έναν ταξινομητή διευθύνσεων (address classifier) (`classifier_`)

β) έναν ταξινομητή θυρίδων (port classifier) (`dmux_`).

Η λειτουργία αυτών των ταξινομητών είναι η κατανομή των εισερχόμενων πακέτων στον κόμβο, είτε στον σωστό πράκτορα (agent), για πακέτα που έχουν προορισμό τον ίδιο τον κόμβο είτε στην σωστή σύνδεση (link) για πακέτα τα οποία έχουν προορισμό άλλους κόμβους του δικτύου, και επομένως πρέπει να προωθηθούν.

Όλοι οι κόμβοι αποτελούνται από τα ακόλουθα στοιχεία:

- την διεύθυνση (`id_`) – είναι ένας ακέραιος που χαρακτηρίζει μοναδικά κάθε κόμβο της τοπολογίας, αρχίζοντας από το 0 και αυξάνοντας κατά 1 για κάθε κόμβο που δημιουργείται.
- μια λίστα με τους γειτονικούς κόμβους (`neighbor_`)
- μία λίστα με τους πράκτορες του κόμβου (`agent_`)
- έναν προσδιοριστή τύπου του (`nodetype_`)
- μια συνάρτηση δρομολόγησης

Εξορισμού οι κόμβοι στο NS2 υποστηρίζουν μόνο unicast προσομοιώσεις. Επομένως, για την προσομοίωση broadcast αποστολών, ήταν απαραίτητο να γίνουν τροποποιήσεις σε έναν τύπο κόμβου ώστε να δημιουργεί broadcast πακέτα, κάτι που θα συζητηθεί σε επόμενο κεφάλαιο.

2.7 Ταξινομητές (Classifiers)

Η βασική λειτουργία ενός κόμβου όταν λάβει ένα πακέτο είναι να εξετάσει τα πεδία του, κυρίως τις διευθύνσεις του παραλήπτη και σε πολλές περιπτώσεις την διεύθυνση του αποστολέα. Στην συνέχεια, πρέπει να αντιστοιχίσει τις τιμές που μόλις ανέγνωσε με την αντίστοιχη διεπαφή εξόδου, στην οποία είναι συνδεδεμένος ο επόμενος κόμβος-παραλήπτης του πακέτου.

Στο NS2 αυτή η λειτουργία πραγματοποιείται από έναν ταξινομητή – ένα απλό αντικείμενο τύπου `Classifier`. Πολλαπλά αντικείμενα `Classifier`, το καθένα ελέγχοντας διαφορετικά πεδία του πακέτου λειτουργούν σε κάθε κόμβο, για την σωστή προώθηση του πακέτου.

Ένας ταξινομητής παρέχει μια μέθοδο για την ταυτοποίηση ενός πακέτου με βάση συγκεκριμένα κριτήρια με σκοπό την δημιουργία μιας αναφοράς σε ένα άλλο αντικείμενο της προσομοίωσης. Για να γίνει αυτό, κάθε

ταξινομητής, περιέχει έναν πίνακα-ευρετήριο με τα όλα τα αντικείμενα της προσομοίωσης, τα οποία δεικτοδοτούνται με έναν αριθμό θυρίδας. Η λειτουργία του ταξινομητή είναι να καθορίσει το αριθμό θυρίδας που σχετίζεται με κάθε πακέτο που λαμβάνει και στην συνέχεια να το προωθήσει στο αντίστοιχο αντικείμενο. Η C++ κλάση Classifier (αρχείο *~ns/classifier.h*) είναι η βασική κλάση από την οποία κληρονομούνται όλοι οι ταξινομητές. Ακολουθεί μια σύντομη περιγραφή των μεθόδων που ορίζονται για την κλάση `Classifier`

Η μέθοδος `alloc()` δεσμεύει δυναμικά χώρο σε έναν πίνακα-ευρετήριο με δεδομένο αριθμό θυρίδων. Οι μέθοδοι `install()` και `clear()` προσθέτουν ή αφαιρούν αντικείμενα από τον πίνακα.

Τα αντικείμενα της προσομοίωσης, μπορούν να αναγνωρίσουν μόνο “Packet-events”, δηλαδή την παραλαβή ενός πακέτου. Όταν καλείται η μέθοδος `recv()`, σημαίνει ότι ο κόμβος έχει παραλάβει ένα πακέτο. Αυτή με την σειρά της καλεί την μέθοδο `classify()` δίνοντας σαν όρισμα το ίδιο το πακέτο. Για κάθε διαφορετικό τύπο ταξινομητή, εκτελείται και διαφορετικός κώδικας για την `classify()`.

Η βασική λειτουργία της μεθόδου είναι να ελέγχει το ευρετήριο με τους δείκτες και να βρίσκει τον αριθμό της θυρίδας. Αν ο δείκτης έχει βρεθεί στο ευρετήριο, και είναι αντιστοιχισμένος σε ένα υπαρκτό αντικείμενο Tcl, τότε ο ταξινομητής θα φροντίσει να προωθήσει το πακέτο στο αντικείμενο αυτό, καλώντας την μέθοδο `recv()` του νέου αντικειμένου. Αν ο δείκτης δεν είναι έγκυρος, ο ταξινομητής θα εμφανίσει ένα μήνυμα λάθους και θα τερματίσει την εκτέλεση της προσομοίωσης.

Η μέθοδος `command()` χρησιμοποιείται για την επικοινωνία Tcl - C++. Ουσιαστικά μεταφράζει τις εντολές που προέρχονται από την Tcl στο κατάλληλο κώδικα C++.

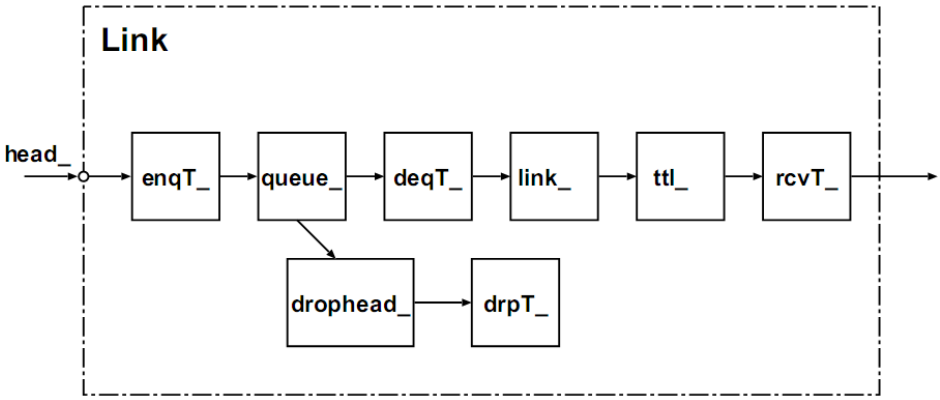
Για την δημιουργία του μεταγωγέα της εργασίας έγιναν αλλαγές στις μεθόδους `recv()` και `command()` οι οποίες περιγράφονται σε επόμενο κεφάλαιο.

2.8 Συνδέσεις (Simple Links)

Μετά την δημιουργία των κόμβων σε μια τοπολογία, πρέπει να οριστεί με ποιόν τρόπο είναι συνδεδεμένοι οι κόμβοι, ώστε να μπορεί να προσομοιωθεί το δίκτυο. Για τον σκοπό αυτό χρησιμοποιούνται οι συνδέσεις (links).

Θα περιγραφούν περιληπτικά οι απλές συνδέσεις σημείου προς σημείο (point to point links), παρότι το σύστημα NS2 υποστηρίζει και άλλα είδη (multi-access LANs, ασύρματες συνδέσεις, κτλ) αφού μόνο οι απλές συνδέσεις ήταν απαραίτητες για την προσομοίωση λειτουργίας του μεταγωγέα.

Όπως ένας κόμβος αποτελείται από ένα σύνολο ταξινομητών (classifiers), έτσι και μια σύνδεση αποτελείται από μια ακολουθία συνδέσμων (connectors). Η δομή της εμφανίζεται στην εικόνα 4.



Εικόνα 4: Δομή μιας μονόδρομης σύνδεσης

Η κλάση `Link` είναι μία αυτόνομη Tcl κλάση, η οποία παρέχει ορισμένα θεμελιώδη χαρακτηριστικά των συνδέσεων. Η κλάση `SimpleLink` παρέχει την δυνατότητα διασύνδεσης δύο κόμβων με μια σύνδεση σημείο προς σημείο. Με την χρήση της συνάρτησης `simplex-link` ο προσομοιωτής του συστήματος δημιουργεί μια μονόδρομη σύνδεση ανάμεσα σε δυο κόμβους, σύμφωνα και με την ακόλουθη σύνταξη:

```
$ns simplex-link <node0> <node1> <bandwidth> <delay>
<queue_type>
```

Έτσι δημιουργείται μια σύνδεση από τον κόμβο `<node0>` στον κόμβο `<node1>` όπου, `<bandwidth>` είναι το εύρος της ζώνης της σύνδεσης σε bits/second και `<delay>` είναι καθυστέρηση στην μετάδοση των δεδομένων

που προσθέτει η σύνδεση σε seconds. Η σύνδεση χρησιμοποιεί μία ουρά για την προσωρινή αποθήκευση των πακέτων που στέλνει ο κόμβος <node0>. Η μέθοδος simplex-link προσθέτει επίσης έναν ελεγκτή χρόνου ζωής πακέτων (TTL checker) στην σύνδεση.

Για κάθε στιγμιότυπο σύνδεσης, ορίζονται οι ακόλουθες μεταβλητές που την χαρακτηρίζουν:

- head_ - Δείκτης στο πρώτο αντικείμενο μέσα στην σύνδεση.
- queue_ - Αναφορά στο βασικό στοιχείο ουράς (queue) της σύνδεσης. Κάθε απλή σύνδεση έχει συνήθως μία ουρά ανά σύνδεση, σε αντίθεση με τις πιο σύνθετους τύπους συνδέσεων.
- link_ - Αναφορά στο αντικείμενο που μοντελοποιεί την σύνδεση, σε σχέση με τα βασικά χαρακτηριστικά της (εύρος ζώνης και καθυστέρηση).
- ttl_ - Αναφορά στον διαχειριστή του χρόνου ζωής καθενός πακέτου στην ουρά.
- drophead_ - Αναφορά στο αντικείμενο που είναι στην κεφαλή της ουράς των αντικειμένων που έχουν απορριφθεί από την σύνδεση (λόγω υπερχείλισης).

Η χρήση της Tcl μεθόδου duplex-link στην κατασκευή της τοπολογίας, δημιουργεί μια αμφίδρομη σύνδεση με την χρήση δύο απλών μονόδρομων συνδέσεων.

2.9 Καθυστερήσεις σε συνδέσεις

Κατά την διάρκεια των προσομοιώσεων, πακέτα δεδομένων μεταδίδονται από κόμβο σε κόμβο μέσω των συνδέσεων. Οι καθυστερήσεις αναπαριστούν τον χρόνο που χρειάζεται ένα πακέτο για να διασχίσει μια σύνδεση ανάμεσα σε δύο κόμβους. Το χρονικό αυτό διάστημα ορίζεται ως $\frac{s}{b} + d$ όπου *s* είναι το μέγεθος του πακέτου (όπως έχει ορισθεί στην επικεφαλίδα του – IP header), *b* είναι η ταχύτητα μεταφοράς δεδομένων της

σύνδεσης σε $\frac{bits}{\sec}$ και *d* είναι η καθυστέρηση της σύνδεσης σε δευτερόλεπτα, η οποία έχει οριστεί κατά την αρχικοποίηση της προσομοίωσης.

2.10 Ουρές (Queues)

Οι ουρές αναπαριστούν περιοχές στις οποίες πακέτα κρατούνται προσωρινά (ή απορρίπτονται). Το σύστημα NS2 παρέχει διάφορες μεθόδους διαχείρισης ουρών (πολιτική FIFO, Round-Robin, διαχείριση με βάση την IP προτεραιότητα και άλλες).

Συνήθως, όπου υπάρχει ένα στοιχείο καθυστέρησης (delay) σε μία σύνδεση, η αντίστοιχη ουρά μπορεί να είναι μπλοκαρισμένη, έως ότου επανενεργοποιηθεί από τον επόμενο γειτονικό κόμβο. Όταν είναι μπλοκαρισμένη, δεν μπορούν να εξαχθούν πακέτα από την δομή της αντίστοιχης ουράς. Με αυτόν τον μηχανισμό προσομοιώνονται οι καθυστερήσεις μετάδοσης. Για την περίπτωση απόρριψης πακέτων λόγω υπερχείλισης , κάθε ουρά περιέχει ένα «προορισμό απορριπτέων πακέτων», που είναι ένα αντικείμενο που παραλαμβάνει τα πακέτα που έχουν απορριφτεί. Αυτό χρησιμοποιείται για την εξαγωγή στατιστικών σχετικά με τα πακέτα που απορριφθήκαν.

Στην ανάπτυξη του μεταγωγέα δεν χρησιμοποιήθηκαν οι ουρές όπως ορίζονται από το σύστημα NS2, αλλά δημιουργήθηκαν νέες κλάσεις που υλοποιούν ουρές προσαρμοσμένες στην λειτουργικότητα του μεταγωγέα και με δυνατότητα παραγωγής επιπλέον στατιστικών στοιχείων, όπως θα δούμε στο επόμενο κεφάλαιο.

2.11 Πράκτορες (Agents)

Οι πράκτορες (agents) για το σύστημα NS2 είναι τα στοιχεία εκείνα που ευθύνονται για την «παραγωγή» ή την «κατανάλωση» των πακέτων σε επίπεδο δικτύου, και χρησιμοποιούνται για τη εφαρμογή των διαφόρων πρωτοκόλλων δικτύων. Για κάθε κόμβο του δικτύου, μπορούν να ορισθούν ένας ή περισσότεροι πράκτορες, οι οποίοι λειτουργούν ταυτόχρονα.

Η κλάση Agent είναι ανεπτυγμένη σε Tcl και C++ (αρχεία *~ns/agent.cc, ~ns/agent.h* και *~ns/tcl/lib/ns-agent.tcl*)

Η C++ κλάση Agent περιέχει μεταβλητές μέλη που χρησιμοποιούνται για την αρχικοποίηση των τιμών στα χαρακτηριστικά πεδία κάθε πακέτου, πριν από την αποστολή του, σύμφωνα με το πρωτόκολλο που υλοποιεί ο κάθε πράκτορας.

Συγκεκριμένα περιέχει τις παρακάτω:

- `addr_` – διεύθυνση του κόμβου όπου ανήκει ο πράκτορας (διεύθυνση αποστολέα του πακέτου)
- `dst_` – ο κόμβος στον οποίο θα αποσταλούν τα πακέτα
- `size_` – το μέγεθος των πακέτων που δημιουργεί ο πράκτορας σε bytes
- `type_` – τύπος του πακέτου (data, ACK, κτλ)
- `prio_` – επίπεδο προτεραιότητας IP (0-7)
- `flags_` – σημαίες πακέτου
- `defttl_` – εξ' ορισμού χρόνος ζωής πακέτου

Οι διάφοροι τύποι πρακτόρων χρησιμοποιούνται για την υλοποίηση των πρωτοκόλλων δικτύων στα διάφορα επίπεδα δικτύου (σύμφωνα με το μοντέλο OSI). Ανάμεσα σε αυτά διάφορες υλοποιήσεις του πρωτοκόλλου TCP, UDP και άλλα.

Ακολουθεί ένα παράδειγμα κώδικα προσομοίωσης για την δημιουργία ενός πράκτορα που υλοποιεί το πρωτόκολλο TCP:

```
set tcp [new Agent/TCP]
$tcp set prio_ 0
set sink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
```

Αρχικά δημιουργείται ένα στιγμιότυπο TCP πράκτορα με το όνομα `tcp` με επίπεδο IP προτεραιότητας 0, καθώς και ένα στιγμιότυπο πράκτορα τύπου καταβόθρας TCP πακέτων με το όνομα `sink`. Στην συνέχεια τα δύο στιγμιότυπα προσαρτώνται στους κόμβους (nodes) `n0` και `n3` αντίστοιχα, που θεωρούμε ότι έχουν ήδη δημιουργηθεί. Τέλος, ο αποστολέας και ο παραλήπτης πρέπει να ενωθούν για να δημιουργηθεί η TCP σύνδεση.

2.12 Χρονοδιακόπτες (Timers)

Οι χρονοδιακόπτες στο σύστημα NS2 χρησιμοποιούνται για την προσομοίωση καθυστερήσεων, ορίζοντας την λήξη του χρονοδιακόπτη σε χρόνο ίσο με την επιθυμητή καθυστέρηση.

Σε επίπεδο C++, οι χρονοδιακόπτες κληρονομούνται από την αφηρημένη κλάση `TimerHandler` (αρχείο *~ns/timer-handler.h*). Η κλάση περιέχει τις παρακάτω δημόσιες συναρτήσεις μέλη:

- `void sched(double delay)` – προγραμματίζει έναν ανενεργό χρονοδιακόπτη να λήξει μετά την πάροδο τόσων δευτερολέπτων όσων ορίζονται στην μεταβλητή `delay`
- `void resched(double delay)` – επαναπρογραμματίζει τον χρονοδιακόπτη (όπως και η `sched()`, αλλά σε αυτήν την περίπτωση μπορεί να γίνει ακόμα και αν ο διακόπτης ήταν ενεργός).
- `void cancel()` – ακυρώνει έναν ενεργό χρονοδιακόπτη.
- `int status()` – επιστρέφει την τρέχουσα κατάσταση του χρονοδιακόπτη.

Οι χρονοδιακόπτες είναι μια μηχανές πεπερασμένων καταστάσεων, οι οποίες λειτουργούν με τις ακόλουθες καταστάσεις: TIMER_IDLE -> TIMER_PENDING -> TIMER_HANDLING. Αρχικά ο χρονοδιακόπτης είναι ανενεργός (idle). Όταν ο χρονοδιακόπτης δεχθεί ένα γεγονός, βρίσκεται στην κατάσταση pending και περιμένει την εκτέλεση του γεγονότος. Όταν λήξει ο απαιτούμενος χρόνος, ο χρονοδιακόπτης είναι σε κατάσταση εκτέλεσης του γεγονότος (handling), και στην συνέχεια καλείται η μέθοδος expire() που τον επιστρέφει στην ανενεργή κατάσταση.

Επίσης, η κλάση `TimerHandler` περιέχει τις παρακάτω ιδιωτικές συναρτήσεις μέλη:

- `virtual void expire(Event* e) = 0` – Η virtual μέθοδος `expire()` πρέπει να οριστεί για κάθε κλάση τύπου timer που κληρονομείται από την αφηρημένη βασική κλάση.
- `virtual void handle(Event* e)` – εκτελεί ένα γεγονός και στην συνέχεια καλεί την `expire()` ώστε να τεθεί ο χρονοδιακόπτης στην σωστή κατάσταση.

- `int status_` – κρατάει την τρέχουσα κατάσταση του χρονοδιακόπτη
- `Event event_` – το γεγονός που πρέπει να εκτελεστεί με την λήξη του χρονοδιακόπτη.

Τέλος, ορίζονται και οι δύο παρακάτω ιδιωτικές inline μέθοδοι:

- ```
  inline void _sched(double delay){
  (void)Scheduler::instance().schedule(this,
  &event_,delay); }
  ```
- ```
  inline void _cancel() {
  (void)Scheduler::instance().cancel(&event_); }
  ```

Όπως φαίνεται και παραπάνω, οι χρονοδιακόπτες χρησιμοποιούν μεθόδους της κλάσης των χρονοπρογραμματιστών (schedulers) (τις μεθόδους `schedule()` και `cancel()`). Οι χρονοδιακόπτες χρησιμοποιήθηκαν εκτενώς στην ανάπτυξη του μεταγωγέα της εργασίας.

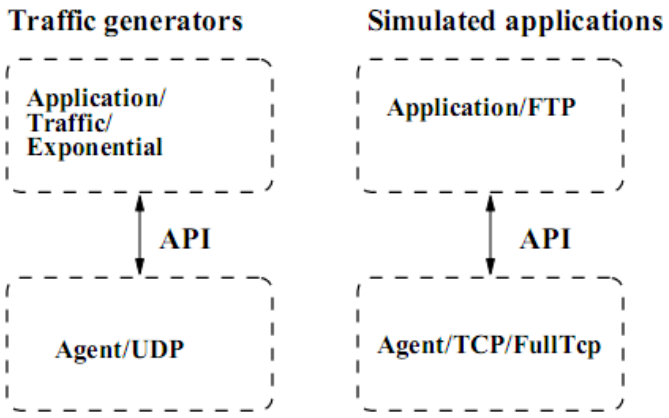
2.13 Εφαρμογές (applications)

Οι εφαρμογές (applications) είναι δομικά στοιχεία του συστήματος NS2 που προσαρτώνται σε κάθε πράκτορα (agent) και είναι υπεύθυνα για την καθαυτού παραγωγή κίνησης δεδομένων στο δίκτυο, σύμφωνα με το πρωτόκολλο δικτύου που υλοποιεί ο αντίστοιχος πράκτορας.

Υπάρχουν δύο βασικοί τύποι εφαρμογών:

- Παραγωγοί κίνησης στο δίκτυο
- Προσομοιωτές πραγματικών εφαρμογών

Η ακόλουθη εικόνα επεξηγεί την σχέση των δύο παραπάνω τύπων εφαρμογών με τον τύπο του πράκτορα στον οποίο προσαρτώνται.



Εικόνα 5: Αντιστοίχηση τύπων πρακτόρων με εφαρμογές

Η βασική κλάση για τις εφαρμογές που παράγουν κίνηση στο δίκτυο είναι η `TrafficGenerator`, ενώ υπάρχουν τέσσερις κλάσεις που την κληρονομούν και παράγουν δεδομένα συμφωνά με κατανομές:

1. EXPOO_Traffic – υλοποιεί “On/Off” περιόδους λειτουργίας με βάση μια εκθετική κατανομή.
2. POO_Traffic – υλοποιεί “On/Off” περιόδους λειτουργίας με βάση μια κατανομή Pareto.
3. CBR_Traffic – παράγει δεδομένα με σταθερό ρυθμό
4. TrafficTrace – παράγει δεδομένα με βάσει ένα αρχείο δεδομένων “trace”.

Μια εφαρμογή που δημιουργεί πακέτα με σταθερό ρυθμό 64Kb/s, μέγεθος πακέτου 48 Bytes και έχει προσαρτηθεί στον πράκτορα $udp δημιουργείται με το παρακάτω παράδειγμα κώδικα προσομοίωσης. Η εφαρμογή του παραδείγματος τίθεται σε λειτουργία την χρονική στιγμή 0.01s.

```
set e [new Application/Traffic/CBR]
$e attach-agent $udp
$e set packetSize_ 48
$e set rate_ 64Kb
$ns at 0.01 "$e start"
```

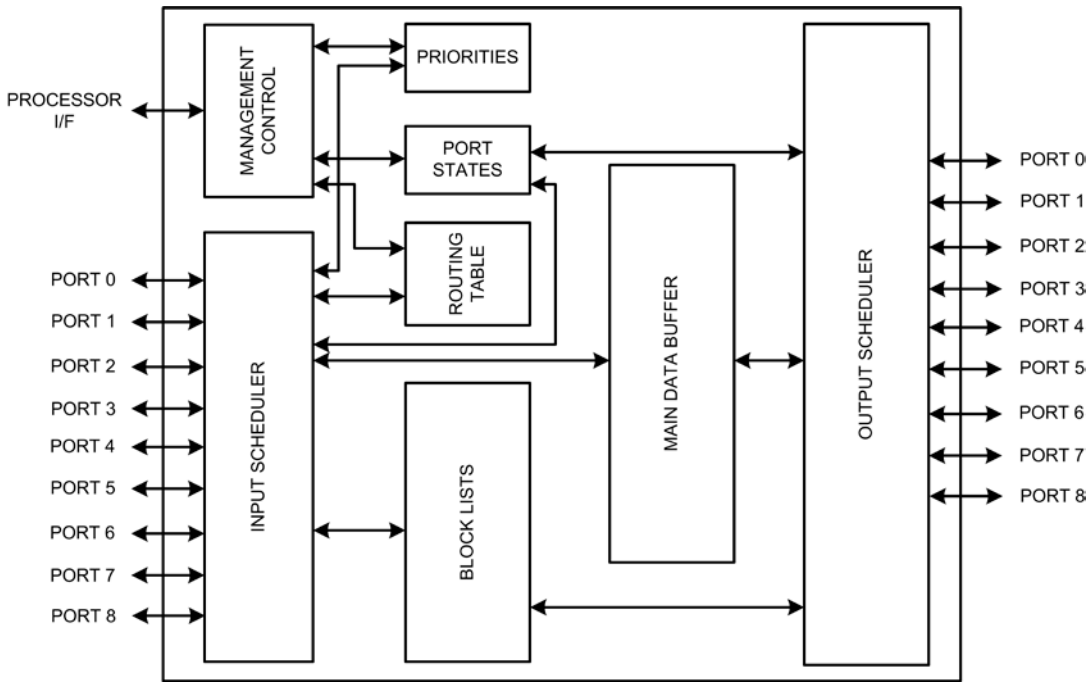
3 Ανάπτυξη μεταγωγέα

3.1 Δομή Μεταγωγέα

Στόχος της εργασίας είναι η προσομοίωση της λειτουργίας ενός μεταγωγέα Ethernet με κοινόχρηστη μνήμη και ουρές εξόδου και στην συνέχεια η προσομοίωση της λειτουργίας του για την εξαγωγή στατιστικών δεδομένων και παρατηρήσεων.

Βασικά δομικά στοιχεία κάθε μεταγωγέα είναι:

- οι πόρτες εισόδου
- οι πόρτες εξόδου
- ένας πίνακας δρομολόγησης
- συνδέσεις μεταξύ όλων των εισόδων και όλων των εξόδων
- μνήμες για την προσωρινή αποθήκευση πακέτων πριν αποσταλούν
- λογική ελέγχου λειτουργίας των επιμέρους στοιχείων



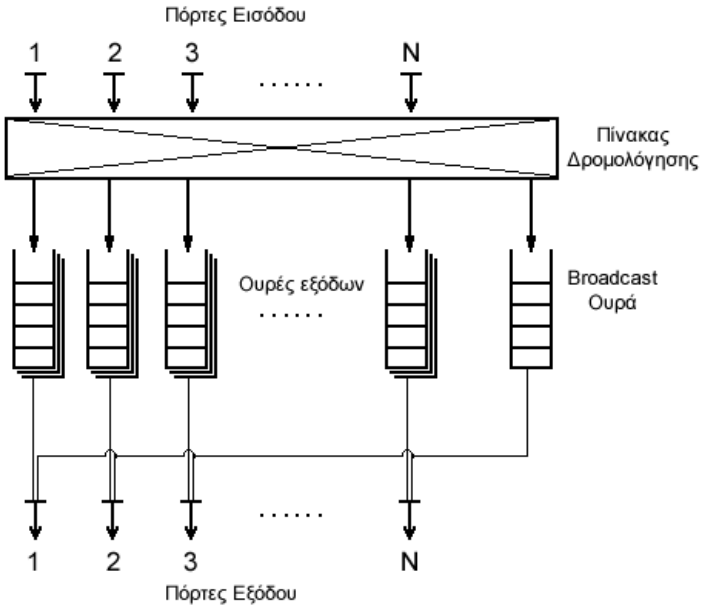
Εικόνα 6: H/W block diagram του πλήρους συστήματος

Στην εικόνα 6, διακρίνεται η αρχιτεκτονική και τα κύρια υποσυστήματα του Ethernet Switch, όπως σχεδιάστηκε στην εργασία «Σχεδιασμός και υλοποίηση ενός 8x8 Ethernet switch». Τα υποσυστήματα του μεταγωγέα χωρίζονται στις παρακάτω κατηγορίες:

- *Input Scheduler.* Το τμήμα αυτό είναι υπεύθυνο για την προώθηση των εισερχόμενων πακέτων προς την μνήμη προσωρινής αποθήκευσης τους.
- *Περιφερειακά.* Στον τομέα των περιφερειακών συμπεριλαμβάνονται τα συστήματα που εξυπηρετούν την επεξεργασία που λαμβάνει χώρα στον Input Scheduler. Πιο συγκεκριμένα, τα υποσυστήματα Priority, Port States, Block Lists, Main Data Buffer και Routing Table, ανήκουν σε αυτήν την κατηγορία.
- *Output Scheduler.* Αντίστροφα με τον Input Scheduler, ο Output Scheduler είναι υπεύθυνος για την ανάγνωση των αποθηκευμένων πακέτων από την μνήμη και την αποστολή τους στις πόρτες εξόδου.
- *Management Interface.* Στην κατηγορία αυτή ανήκει ένα υποσύστημα υπεύθυνο για την ρύθμιση/αρχικοποίηση βασικών παραμέτρων του συστήματος, όπως η κατάσταση των πορτών, οι προτεραιότητες και ο πίνακας δρομολόγησης.

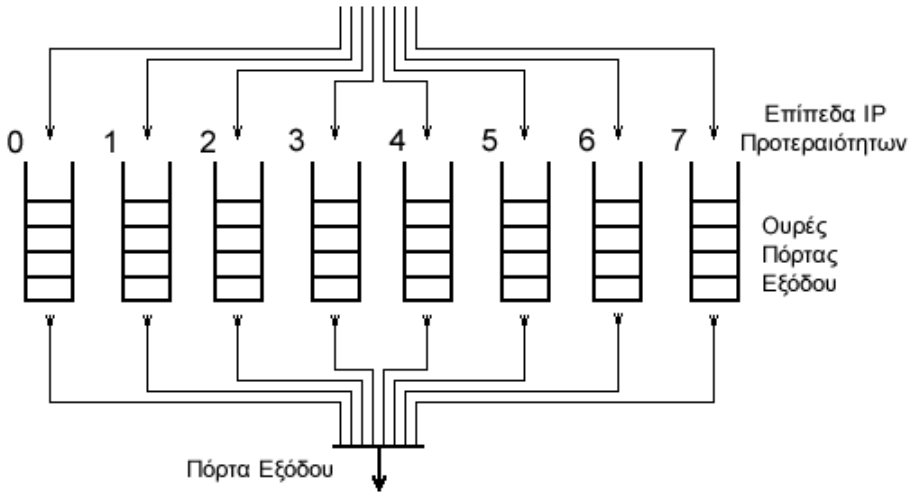
Κατά την ανάπτυξη του προσομοιωτή του μεταγωγέα, ορισμένα κομμάτια δεν ήταν απαραίτητο να υλοποιηθούν ενώ η λειτουργίες άλλων συνενώθηκαν σε ένα υπερσύστημα. Συγκεκριμένα, ο Input Scheduler μαζί με το υποσύστημα Routing Table υλοποιήθηκαν σαν ένα ενιαίο σύστημα δρομολόγησης των εισερχόμενων πακέτων. Ο Output Scheduler υλοποιήθηκε μαζί με τα υποσυστήματα Main Data Buffer και Priority, σαν μια δομή εξόδου με μνήμη. Τα υποσυστήματα Port States και τμήματα του Management Interface υλοποιήθηκαν σαν ένα ανώτερο επίπεδο ελέγχου των προηγούμενων, ενώ το υποσύστημα Block Lists δεν υλοποιήθηκε γιατί σχετίζεται με την μορφή της μνήμης σε επίπεδο υλικού.

Το ακόλουθο διάγραμμα απεικονίζει την γενική μορφή του μεταγωγέα που αναπτύχθηκε και προσομοιώθηκε.



Εικόνα 7: Εσωτερική δομή του μεταγωγέα που αναπτύχθηκε

Ορίζουμε N το πλήθος των εισόδων/εξόδων του μεταγωγέα. Κάθε πόρτα εισόδου δέχεται συνεχώς πακέτα από τους συνδεδεμένους σε αυτή κόμβους, ενώ κάθε πόρτα εξόδου είναι συνδεδεμένη με μια μνήμη προσωρινής αποθήκευσης των πακέτων που πρέπει να σταλούν από αυτή την έξοδο. Η μνήμη αυτή πρέπει να έχει την μορφή ουράς. Συγκεκριμένα θα είναι χωρισμένη σε 8 επιμέρους ουρές με βάση τις IP προτεραιότητες, όπως φαίνεται στην εικόνα 8.



Εικόνα 8: Εσωτερική δομή μίας εξόδου

3.1.1. Δρομολόγηση πακέτων

Ο πίνακας δρομολόγησης είναι υπεύθυνος για την σωστή κίνηση των πακέτων, από την είσοδο στην οποία καταφθάνει κάθε πακέτο, μέσω των συνδέσεων μεταξύ εισόδων και εξόδων, προς την αντίστοιχη ουρά εξόδου για κάθε πακέτο. Για να εκτελεστεί αυτή η λειτουργία, ο πίνακας δρομολόγησης αποτελείται από μια δομή που αντιστοιχίζει σε κάθε πόρτα εξόδου, την διεύθυνση IP του κόμβου που είναι συνδεδεμένος σε αυτή. Όταν το σύστημα δεχθεί ένα εισερχόμενο πακέτο, διαβάζει την διεύθυνση προορισμού του από την IP επικεφαλίδα του και στην συνέχεια επιλέγει, με βάση τον πίνακα δρομολόγησης, προς ποια έξοδο πρέπει να προωθηθεί το πακέτο. Στον προσομοιωτή που δημιουργήθηκε δεν υλοποιήθηκε δομή πίνακα δρομολόγησης, αφού ουσιαστικά, η κάθε έξοδος διευθυνσιοδοτείται με βάση την διεύθυνση του κόμβου του συστήματος NS2 ο οποίος είναι συνδεδεμένος με αυτή (όπως περιγράφηκε στην παράγραφο 2.6). Επομένως, σε αυτή την περίπτωση ο πίνακας δρομολόγησης θα περιείχε εγγραφές της μορφής «πόρτα: 1 -> διεύθυνση IP: 1, πόρτα: 2 -> διεύθυνση IP: 2». Ο τρόπος υλοποίησης της δρομολόγησης των πακέτων εντός του μεταγωγέα προς την σωστή έξοδο περιγράφεται στην υποπαράγραφο 3.3.4.

3.1.2. Εξυπηρέτηση unicast πακέτων

Κάθε εισερχόμενο πακέτο, αφού προωθηθεί προς την κατάλληλη έξοδο πρέπει να αποθηκευθεί προσωρινά σε κάποια μνήμη, έως ότου η έξοδος είναι διαθέσιμη για να αρχίσει την αποστολή του. Όπως φαίνεται στην εικόνα 8, η μνήμη αυτή είναι χωρισμένη σε 8 ισομεγέθεις ουρές δεδομένων, οι οποίες δέχονται πακέτα ίδιας IP προτεραιότητας η κάθε μια. Έτσι, ουσιαστικά κατά την δρομολόγηση, εκτός από την επιλογή της σωστής εξόδου, πρέπει να γίνει και επιλογή της αντίστοιχης ουράς, μέσω της ανάγνωσης του πεδίου IP προτεραιότητας της επικεφαλίδας κάθε πακέτου.

Η αποστολή των unicast πακέτων, γίνεται με βάση την προτεραιότητα. Όσο μια ουρά υψηλότερης προτεραιότητας περιέχει πακέτα προς αποστολή, εξυπηρετείται, δηλαδή η έξοδος επιλέγει να στέλνει πακέτα από την συγκεκριμένη ουρά, ενώ οι υπόλοιπες βρίσκονται σε αναμονή. Σε αυτή την

κατάσταση μπορούν να δέχονται πακέτα, μέχρις ότου υπάρξει υπερχείλιση. Σε αυτή την περίπτωση, τα πακέτα που καταφθάνουν στον μεταγωγέα και προορίζονται για την συγκεκριμένη έξοδο και ανήκουν στο επίπεδο προτεραιότητας του οποίου η ουρά έχει υπερχειλίσει, απορρίπτονται.

3.1.3. Εξυπηρέτηση broadcast πακέτων

Για την εξυπηρέτηση broadcast αποστολών πακέτων, στην απλούστερη περίπτωση ο μεταγωγέας θα μπορούσε να δημιουργεί και να εισάγει ένα αντίγραφο του broadcast πακέτου σε κάθε ουρά εξόδου, το οποίο και τελικά θα αποστελλόταν στον προορισμό του με βάση την χρονική προτεραιότητα των πακέτων σε κάθε επιμέρους ουρά. Σε αυτή την περίπτωση θα έπρεπε αρχικά να δημιουργηθούν N αντίγραφα του εισερχόμενου broadcast πακέτου και στην συνέχεια να γίνουν N εισαγωγές πακέτων στις ουρές. Όμως, αν για παράδειγμα κατά την διάρκεια της προσομοίωσης ο μεταγωγέας παραλάβει N broadcast πακέτα, θα πρέπει να γίνουν N^2 εισαγωγές, κάτι το οποίο θα επέφερε μεγάλο φόρτο εργασίας στο σύστημα και θα δημιουργούσε θέμα ταχύτητας.

Επομένως, η εξυπηρέτηση των broadcast αποστολών, υλοποιήθηκε με διαφορετική φιλοσοφία. Σύμφωνα με αυτή, ο μεταγωγέας χρησιμοποιεί μια επιπλέον ουρά προσωρινής αποθήκευσης broadcast πακέτων, η οποία είναι συνδεδεμένη απευθείας με όλες τις εξόδους, όπως φαίνεται στην εικόνα 8. Όταν ένα πακέτο για broadcast παραληφθεί από μια είσοδο, προωθείται μόνο στην broadcast ουρά. Στην συνέχεια πρέπει να σταλούν αντίγραφα του πρώτου πακέτου προς αποστολή της συγκεκριμένης ουράς, προς όλες τις εξόδους. Με αυτόν τον τρόπο για κάθε broadcast πακέτο που λαμβάνει ο μεταγωγέας γίνεται μόνο μια εισαγωγή σε ουρά.

Κάθε έξοδος δέχεται πακέτα από μια από τις 8 ουρές προτεραιοτήτων (εικόνα 8) ή από την broadcast ουρά (εικόνα 7). Η λογική ελέγχου του μεταγωγέα υλοποιεί στην ουσία ένα πολυπλέκτη που είναι υπεύθυνος για την επιλογή του πακέτου που θα εξυπηρετηθεί (unicast ή broadcast). Για να ξεκινήσει την διαδικασία αποστολής ενός broadcast πακέτου, η έξοδος θα πρέπει να είναι ανενεργή, δηλαδή να μην βρίσκεται σε κατάσταση αποστολής

δεδομένων. Αυτό συμβαίνει είτε όταν δεν υπάρχουν καθόλου πακέτα στις ουρές της εξόδου προς αποστολή, είτε στιγμιαία ανάμεσα σε δυο διαδοχικές αποστολές.

Σαφώς, η πρώτη περίπτωση, δηλαδή οι αποστολές broadcast να περιμένουν έως ότου οι ουρές να είναι άδειες δεν είναι καθόλου αποδοτική, αφού τα broadcast θα αποστέλλονται με πολύ μεγάλη καθυστέρηση ή και ποτέ, ανάλογα με τον φόρτο του δικτύου.

Επομένως, οι αποστολές των broadcast από κάθε έξοδο, πρέπει γίνονται ανάμεσα σε διαδοχικές αποστολές unicast. Ο απλούστερος τρόπος για να συμβεί αυτό, είναι κάθε έξοδος να στέλνει ένα αντίγραφο του πρώτου προς αποστολή broadcast πακέτου, αμέσως μόλις ολοκληρωθεί η τρέχουσα unicast αποστολή της. Αυτό όμως, οδηγεί σε μη ταυτόχρονη αποστολή των broadcast πακέτων από τις εξόδους, ενώ χρειάζονται επιπλέον έλεγχοι έτσι ώστε ένα broadcast πακέτο να βγαίνει από την αντίστοιχη ουρά μόνο όταν όλες οι έξοδοι έχουν αποστείλει ένα αντίγραφό του.

Μια βελτίωση του παραπάνω τρόπου εξυπηρέτησης broadcast, ορίζει κάθε έξοδο σε κατάσταση αναμονής αμέσως μετά την ολοκλήρωση της τρέχουσας αποστολής, αν υπάρχουν πακέτα στην broadcast ουρά (ανεξαρτήτως από το αν υπάρχουν unicast πακέτα στις ουρές). Μόλις όλες οι έξοδοι βρεθούν σε κατάσταση αναμονής, αρχίζει η αποστολή του πρώτου broadcast. Μόλις ολοκληρωθεί, αν υπάρχουν και άλλα broadcast, συνεχίζει η εξυπηρέτηση τους, αλλιώς οι έξοδοι επιστρέφουν στην εξυπηρέτηση των unicast πακέτων. Κατά μέσο όρο, η προσέγγιση αυτή προσθέτει στην καθυστέρηση εξυπηρέτησης unicast πακέτων μια επιπλέον καθυστέρηση ίση με το μισό του μέσου απαιτούμενου χρόνου για την αποστολή ενός πακέτου.

Στην υλοποίηση του κώδικα προσομοίωσης, ακολουθήθηκε μια παρεμφερής προσέγγιση. Συγκεκριμένα, αντί να περιμένει έως ότου όλες οι έξοδοι είναι ανενεργές, ο μεταγωγέας σταματάει όλες τις τρέχουσες αποστολές unicast πακέτων από τις εξόδους και τις θέτει σε κατάσταση αναμονής (χωρίς να έχει διαγράψει το πακέτο από την unicast ουρά που βρισκόταν). Στην συνέχεια, ξεκινάει την διαδικασία αποστολή αντιγράφων του πρώτου προς αποστολή πακέτου της broadcast ουράς προς όλες τις εξόδους. Όσο υπάρχουν broadcast πακέτα προς αποστολή, συνεχίζεται η εξυπηρέτηση της broadcast ουράς. Οι unicast ουρές συνεχίζουν να βρίσκονται σε

κατάσταση αναμονής, δηλαδή δέχονται πακέτα (έως ότου υπερχειλίσουν), αλλά δεν στέλνουν. Όταν έχουν αποσταλεί όλα τα broadcast πακέτα και η αντίστοιχη ουρά αδειάσει, γίνεται επανεκκίνηση της εξυπηρέτησης των unicast ουρών. Το πρώτο πακέτο που θα εξυπηρετηθεί είναι αυτό που είχε διακοπεί προηγουμένως. Η προσέγγιση αυτή προσθέτει ακριβώς την ίδια καθυστέρηση, αλλά θεωρήθηκε ευκολότερη σε υλοποίηση προγραμματιστικά.

3.2 Υλοποίηση δομής μεταγωγέα

Για την προσομοίωση της λειτουργίας του μεταγωγέα Ethernet με κοινόχρηστη μνήμη και ουρές εξόδου, ήταν απαραίτητο να αναπτυχθούν κάποιες νέες κλάσεις για την υλοποίηση των επιμέρους στοιχείων που περιγράφηκαν προηγουμένως, καθώς και να τροποποιηθεί κατά περιπτώσεις η βασική λειτουργία ενός κόμβου του συστήματος NS2.

Αρχικά δημιουργήθηκε σε C++ μια κλάση που υλοποιεί ουρές πακέτων, με συγκεκριμένη χωρητικότητα σε bytes, η οποία περιγράφεται στην υποπαράγραφο 3.3.1. Στην συνέχεια δημιουργήθηκε η κλάση που υλοποιεί την εσωτερική δομή του μεταγωγέα, χρησιμοποιώντας στιγμιότυπα των παραπάνω ουρών, όπως περιγράφεται στο 3.3.2. Επιπλέον δημιουργήθηκε μια κλάση για τον έλεγχο των καθυστερήσεων ανά έξοδο που προκαλούνται εντός του κόμβου, με την χρήση χρονοδιακοπτών (timers), η οποία περιγράφεται στην ενότητα 3.3.3.

Για τον έλεγχο της λειτουργίας των παραπάνω κλάσεων, ήταν απαραίτητο να τροποποιηθεί ο τρόπος λειτουργίας του ταξινομητή (classifier) κάθε κόμβου που δηλώνεται στην έναρξη της προσομοίωσης ως κόμβος μεταγωγέα Ethernet (`sw_node`). Αυτό έγινε τροποποιώντας την αντίστοιχη κλάση C++ του συστήματος όπως περιγράφεται στην ενότητα 3.3.4.

Τέλος, έγιναν μικρές αλλαγές σε κώδικα TCL ώστε να υποστηρίζει το σύστημα τον νέο τύπο κόμβου που δημιουργήθηκε (παράγραφος 3.4), ενώ και αναπτύχθηκαν σε TCL και οι κώδικες των προσομοιώσεων, όπως περιγράφονται στην παράγραφο 3.5

3.3 Ανάπτυξη κώδικα σε C++

Το βασικό κομμάτι της υλοποίησης του μεταγωγέα έγινε σε γλώσσα C++. Η δομή του μεταγωγέα (είσοδοι-έξοδοι, ουρές εξόδων κτλ) περιγράφονται στις παραγράφους «Ουρές εξόδου» και «Κόμβος μεταγωγέα Ethernet» ενώ η λειτουργία του (παραλαβή-αποστολή πακέτων, καθυστερήσεις κτλ) περιγράφονται στις παραγράφους «Χρονοδιακόπτες εξόδων» και «Ταξινομητής (Classifier)», που ακολουθούν.

3.3.1. Ουρές εξόδου

Ένα κομμάτι της λειτουργίας του μεταγωγέα είναι να δέχεται πακέτα, από κάθε είσοδο, και να τα εισάγει στην σωστή ουρά εξόδου με βάση το πεδίο που περιγράφει την διεύθυνση προορισμού κάθε πακέτου. Επομένως, βασικό δομικό στοιχείο του μεταγωγέα που αναπτύχθηκε είναι οι ουρές εξόδου. Η λειτουργικότητα των ουρών περιγράφεται στα αρχεία *sw_queue.h* και *sw_queue.cc*.

Η κάθε ουρά αποθηκεύει δομές τύπου sw_packet, οι οποίες ορίζονται ως εξής:

```
typedef struct {
	void* packet;
	void* packet_handler;
	float arrival_time;
	bool used;
} sw_packet;
```

Όπου:

- `packet` είναι το πακέτο δεδομένων, σύμφωνα με την κλάση `Packet` του NS2
- `packet_handler` είναι ο διαχειριστής πακέτων, σύμφωνα με την κλάση `Handler` του NS2
- `arrival_time` είναι η χρονική στιγμή στην οποία το πακέτο κατέφθασε στην αντίστοιχη ουρά (χρησιμοποιείται για την εξαγωγή στατιστικών μετά την προσομοίωση)
- `used`: μια boolean μεταβλητή που ορίζει αν χρησιμοποιείται η αντίστοιχη θέση στην ουρά.

Οι ουρές υλοποιούνται με κυκλικές λίστες οι οποίες λειτουργούν με πολιτική FIFO (first-in first-out). Κάθε στιγμιότυπο της κλάσης ουράς έχει τα παρακάτω δημόσια και ιδιωτικά μέλη:

```
class sw_queue
{
public:
	sw_queue();
	~sw_queue();
	void swInit(int sz);
	bool insertPacket(sw_packet* pack);
	int getQueueSize();
	sw_packet* removePacket();
	int getNumOfPackets();
	int getFirst();
	int getLast();
	bool returnUsed(int i);
	float returnArrivalTime(int i);
	sw_packet* returnSpecPacket(int i);
	int returnBytesInQueue();
	void updateSentNum();
	void updateReceivedNum();
	void updateDroppedNum();
	void updateAcceptedNum();
	int getSentNum();
	int getReceivedNum();
	int getDroppedNum();
	int getAcceptedNum();
	float getDelay();

private:
	sw_packet* packets_queue;
	int queue_size;
	float time_delay;
	int max_bytes;
	int first_packet;
	int last_packet;
	int num_of_received;
	int num_of_accepted;
	int num_of_sent;
	int num_of_dropped;
};
```

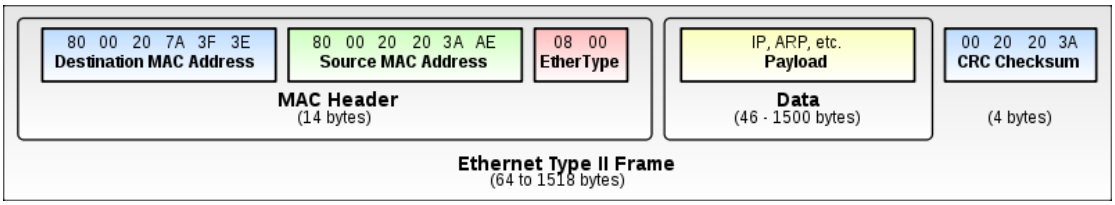
Κάθε στιγμιότυπο της κλάσης `sw_queue`, περιέχει μια ουρά (`packets_queue`) από δομές `sw_packet`. Η μεταβλητή `queue_size` περιέχει το μέγεθος της ουράς σε θέσεις πακέτων, ενώ η `max_bytes` το μέγιστο μέγεθος δεδομένων που μπορεί να δεχτεί μια ουρά σε bytes. Οι μεταβλητές `first_packet` και `last_packet` κρατάνε την θέση του πρώτου (παλαιότερου) και του τελευταίου (νεότερου) πακέτου στην ουρά.

Για κάθε ουρά, καταγράφεται πληροφορία για το πλήθος των πακέτων που κατέφθασαν, το πλήθος όσων έγιναν αποδεκτά, όσων απεστάλησαν

καθώς και όσων απορρίφθηκαν (`num_of_received`, `num_of_accepted`, `num_of_sent` και `num_of_dropped` αντίστοιχα) .

Κατά την έναρξη της προσομοίωσης, ο χρήστης αρχικοποιεί το μέγεθος της ουράς σε bytes. Στην πραγματικότητα όμως, οι ουρές κρατάνε πακέτα δεδομένων. Επομένως, για την σωστή αρχικοποίηση κάθε ουράς (δέσμευση χώρου στην μνήμη), πρέπει να υπολογισθεί το μέγιστο πλήθος πακέτων που μπορεί να δεχθεί η ουρά για το δεδομένο από τον χρήστη μέγεθος σε bytes. Το πλήθος αυτό μεγιστοποιείται όταν η ουρά λαμβάνει πακέτα με το μικρότερο δυνατό μέγεθος μόνο.

Στο πρωτόκολλο Ethernet, το ελάχιστο μέγεθος πακέτου είναι 64 bytes. Από αυτά 26 bytes χρησιμοποιούνται από τα πεδία διευθύνσεων αποστολέα και παραλήπτη, το πεδίο που ορίζει τον τύπο του πακέτου, καθώς και για τον έλεγχο των δεδομένων (CRC Checksum), όπως φαίνεται και στην εικόνα 9.



Εικόνα 9: Δομή ενός πακέτου Ethernet

Επομένως, η κάθε ουρά θα πρέπει να αρχικοποιηθεί με qs/64 θέσεις πακέτων, όπου qs είναι το δεδομένο από τον χρήστη μέγεθος ουράς σε bytes.

Η μεταβλητή `time_delay` κρατάει το άθροισμα των καθυστερήσεων για όλα τα πακέτα που έχουν αποσταλεί από την ουρά. Στο τέλος της προσομοίωσης, διαιρώντας την συνολική καθυστέρηση με το πλήθος των πακέτων που απεστάλησαν, προκύπτει η μέση καθυστέρηση ανά πακέτο.

Οι μέθοδοι της κλάσης των ουρών εξόδου είναι οι παρακάτω:

- `void swInit(int sz)`: αρχικοποιεί την ουρά, δημιουργώντας sz θέσεις πακέτων.
- `bool insertPacket(sw_packet* pack)`: καλείται κάθε φορά που η ουρά πρέπει να παραλάβει ένα πακέτο. Δέχεται σαν όρισμα το πακέτο και επιστρέφει true όταν το πακέτο εισαχθεί με επιτυχία στη ουρά. Για τον έλεγχο υπερχείλισης, πριν γίνει μια εισαγωγή γίνεται έλεγχος αν το συνολικό μέγεθος των δεδομένων στην ουρά, μετά την

εισαγωγή, θα υπερβεί την τιμή `max_bytes`. Σε αυτήν την περίπτωση, το πακέτο απορρίπτεται. Αλλιώς, το πακέτο εισάγεται στο τέλος της ουράς και ενημερώνονται τα στατιστικά για το συνολικό πλήθος πακέτων που έχουν γίνει δεκτά από την ουρά.

- `int getQueueSize()`: επιστρέφει το `queue_size`.
- `sw_packet* removePacket()`: επιστρέφει το παλαιότερο εισηγμένο πακέτο εντός της ουράς.
- `sw_packet* returnSpecPacket(int i)`: επιστρέφει το πακέτο που βρίσκεται στην θέση `i`.
- `int getNumOfPackets()`: επιστρέφει το τρέχον πλήθος πακέτων στην ουρά.
- `int getFirst()`,
  `int getLast()`: επιστρέφουν την θέση του πρώτου και του τελευταίου πακέτου αντίστοιχα.
- `bool returnUsed(int i)`: επιστρέφει true αν η θέση `i` της ουράς περιέχει πακέτο.
- `float returnArrivalTime(int i)`: επιστρέφει την χρονική στιγμή `arrival_time`, που παραλήφθηκε το πακέτο στην θέση `i`.
- `int returnBytesInQueue()` : επιστρέφει το άθροισμα των bytes όλων των πακέτων εντός της ουράς και χρησιμοποιείται στον έλεγχο υπερχείλησης.
- `void updateSentNum()`,
  `void updateReceivedNum()`,
  `void updateDroppedNum()`,
  `void updateAcceptedNum()`: μέθοδοι που καλούνται ώστε να ενημερώσουν τα αντίστοιχα στατιστικά για την ουρά κάθε φορά που ένα πακέτο αποστέλλεται, παραλαμβάνεται, απορρίπτεται ή εισάγεται στην ουρά αντίστοιχα.
- `int getSentNum()`,
  `int getReceivedNum()`,
  `int getDroppedNum()`,
  `int getAcceptedNum()`,

`float getDelay()`: μέθοδοι που επιστρέφουν τα αντίστοιχα στατιστικά στοιχεία για την ουρά.

3.3.2. Κόμβος μεταγωγέα Ethernet

Η κλάση που προσομοιώνει την δομή του μεταγωγέα είναι η sw_node. Η λειτουργικότητα της περιγράφεται στα αρχεία *sw_node.h* και *sw_node.cc*.

Κάθε στιγμιότυπο της κλάσης του μεταγωγέα έχει τα παρακάτω δημόσια και ιδιωτικά μέλη:

```
class sw_node {
public:
	sw_node();
	~sw_node();
	void sw_nodeInit(int inpt, int outpt, int qs, float bndwdth);
	bool sw_insertPacket(int outpt, sw_packet* newpack);
	sw_packet* sw_removePacket(int outpt, int prio);
	bool sw_insertbrdcstPacket(sw_packet* newpack);
	sw_packet* sw_removebrdcstPacket();
	int returnFirstPacketSize(int outpt, int prio);
	int returnFirstPacketSizeBrdcst();
	int getOutPortsNum();
	float getBandwidth();
	int calcCurrNumOfPacks(int inpt, int outpt);
	int calcCurrNumOfPacks(int outpt);
	int calcCurrNumOfPacksBrdcst();
	void printQueueIndex(int src, int dst);
	void printQueueStats(int src, int dst);
	void printQueueStats(int dst);
	void printQueueStats();
	void portIsUsed(int col);
	void addBits(int col, int bts);
	void printSumOfBits();
	bool isQueueUsed(int src, int dst);
	bool isQueueUsed(int dst);
	bool isOutputPortUsed(int dst);
	bool isPriorityQueueUsed(int dst);
	int returnQueueuSumOfBits(int outpt);
	int returnQueueOfPriority(int outpt);


private:
	int input_ports;
	int output_ports;
	sw_queue* node_queues;
	sw_queue brdcst_queue;
	float bandwidth;
	int* bits_per_output;
	int* used_ports;
	int bits_of_brdcst_queue;
	int retQueueNumber(int outpt, int prio);
	int sizeBits;
};
```

Κάθε κόμβος αποτελείται από 8*n+1 ουρές τύπου sw_queue, όπου n είναι το πλήθος των εξόδων, όπως αυτό έχει δηλωθεί στην αρχικοποίηση της προσομοίωσης. Για κάθε πόρτα εξόδου υπάρχουν 8 ουρές, μια για κάθε επίπεδο προτεραιότητας (node_queues) καθώς και μια κοινή ουρά για την εξυπηρέτηση broadcast πακέτων (brdcst_queue). Έτσι, για παράδειγμα, οι ουρές node_queues[0] έως node_queues[7], εξυπηρετούν την πόρτα εξόδου 0 για πακέτα IP προτεραιοτήτων 0 έως 7 αντίστοιχα.

Ο πίνακας bits_per_output και η μεταβλητή bits_of_brdcst_queue κρατάνε συνεχώς το συνολικό πλήθος των bits όλων των πακέτων που βρίσκονται κάθε στιγμή σε κάθε επιμέρους ουρά, για τον έλεγχο υπερχείλισης. Υπερχείλιση συμβαίνει όταν το συνολικό μέγεθος ξεπεράσει την τιμή sizeBits που έχει οριστεί στην αρχικοποίηση της προσομοίωσης.

Ο ονομαστικός ρυθμός μεταφοράς δεδομένων του μεταγωγέα, δηλώνεται στην αρχή της προσομοίωσης και για κάθε στιγμιότυπο μεταγωγέα αποθηκεύεται στην μεταβλητή bandwidth.

Οι μέθοδοι της κλάσης του κόμβου είναι οι παρακάτω:

- void sw_nodeInit(int inpt, int outpt, int qs, float bndwdth) : αρχικοποιεί τον κόμβο με inpt πλήθος εισόδων/εξόδων, μέγεθος ουρών qs και ρυθμό μετάδοσης δεδομένων bndwdth σε bits/sec.
- bool sw_insertPacket(int outpt, sw_packet* newpack): εισάγει το πακέτο newpack στην αντίστοιχη ουρά προτεραιότητας της εξόδου outpt με βάση την IP προτεραιότητά του.
- sw_packet* sw_removePacket(int outpt, int prio): επιστρέφει το πρώτο προς αποστολή πακέτο από την ουρά προτεραιότητας prio της εξόδου outpt.
- bool sw_insertbrdcstPacket(sw_packet* newpack): εισάγει το πακέτο newpack στην ουρά broadcast και επιστρέφει true σε επιτυχή εισαγωγή.
- sw_packet* sw_removebrdcstPacket() : επιστρέφει το πρώτο προς αποστολή πακέτο από την broadcast ουρά.

- `int returnFirstPacketSize(int outpt, int prio))`: επιστρέφει το μέγεθος του πρώτου προς αποστολή πακέτου από την ουρά προτεραιότητας `prio` της εξόδου `outpt`.
- `int returnFirstPacketSizeBrdcst())`: επιστρέφει το μέγεθος του πρώτου προς αποστολή πακέτου από την broadcast ουρά.
- `int getOutPortsNum()` : επιστρέφει το πλήθος εισόδων/εξόδων του μεταγωγέα.
- `float getBandwidth()` : επιστρέφει τον ρυθμό μετάδοσης δεδομένων του μεταγωγέα.
- `int calcCurrNumOfPacks(int outpt)` : επιστρέφει το πλήθος πακέτων της ουράς.
- `int calcCurrNumOfPacksBrdcst()` : επιστρέφει το πλήθος πακέτων της ουράς broadcast.
- `void printQueueIndex(int src, int dst)`,
  `void printQueueStats(int src, int dst)`,
  `void printQueueStats(int dst)`,
  `void printSumOfBits()`,
  `void printQueueStats()` : μέθοδοι για την εμφάνιση στατιστικών στο τέλος κάθε προσομοιωσης.
- `void portIsUsed(int col)` : χρησιμοποιείται για την ενημέρωση του πίνακα `used_ports` με το ποιες ουρές χρησιμοποιούνται.
- `void addBits(int col, int bts)` : προσθέτει bts πλήθος bits στην θέση `col` του πίνακα `bits_per_output`, για τον έλεγχο υπερχείλισης της αντίστοιχης ουράς.
- `bool isQueueUsed(int dst)` : επιστρέφει true αν η ουρά `dst` χρησιμοποιείται.
- `bool isOutputPortUsed(int dst)` : επιστρέφει true αν η πόρτα εξόδου `dst` χρησιμοποιείται.
- `bool isPriorityQueueUsed(int dst)` : επιστρέφει true αν η ουρά υψηλότερης προτεραιότητας της πόρτας εξόδου `dst` χρησιμοποιείται.

- `int returnQueueuSumOfBits(int outpt)` : επιστρέφει το συνολικό πλήθος bits της ουράς `outpt`.
- `int returnQueueOfPriority(int outpt)` : επιστρέφει τον αριθμό της ουράς υψηλότερης IP προτεραιότητας της εξόδου `outpt` που περιέχει πακέτα προς αποστολή.

3.3.3. Χρονοδιακόπτες εξόδων

Για την προσομοίωση χρονικών καθυστερήσεων εντός του κόμβου του μεταγωγέα, δημιουργήθηκε η κλάση `sw_timer` η οποία κληρονομεί την βασική κλάση του συστήματος `TimerHandler`, που περιγράφηκε στο προηγούμενο κεφάλαιο. Η λειτουργικότητά τους περιγράφεται στα αρχεία *sw_timer.h* και *sw_timer.cc*.

Κάθε στιγμιότυπο χρονοδιακόπτη εξόδου έχει τα παρακάτω μέλη:

```
class sw_timer : public TimerHandler
{
public:
	sw_timer();
	void InitSwTimer(Classifier *inst,int id);
	int GetTimerId();
protected:
	virtual void expire(Event *e);
private:
	Classifier* sw_classifier;
	int timer_id;

};
```

Η μέθοδος `InitSwTimer()`αρχικοποιεί ένα νέο χρονοδιακόπτη. Η κλήση της μπορεί να γίνει μόνο μέσα από ένα αντικείμενο τύπου ταξινομητή (`Classifier`) με ορίσματα μία αναφορά στο ίδιο το αντικείμενο, η οποία θα αντιστοιχιστεί στο ιδιωτικό μέλος `sw_classifier` και το `id` της εξόδου που θα διαχειρίζεται ο εκάστοτε χρονοδιακόπτης, το οποίο εκχωρείται στην μεταβλητή `timer_id`.

Η μέθοδος `expire()` υπερβαίνει την αντίστοιχη μέθοδο της βασικής κλάσης `TimerHandler`. Όταν καλείται η `expire()` από ένα αντικείμενο `sw_timer`, καλεί την μέθοδο `sw_SendPackets()` του αντικειμένου `sw_classifier` της οποίας η λειτουργία θα αναλυθεί στην συνέχεια.

3.3.4. Ταξινομητής (Classifier)

Όπως έχει περιγραφεί σε προηγούμενη παράγραφο, η δρομολόγηση των πακέτων μέσα στο δίκτυο που προσομοιώνεται από το σύστημα NS2 γίνεται από τους ταξινομητές (classifiers) κάθε κόμβου. Για την υλοποίηση του μεταγωγέα ήταν απαραίτητη η τροποποίηση της λειτουργία του ταξινομητή, όταν ο κόμβος έχει δηλωθεί σαν κόμβος μεταγωγέα (sw_node). Μέσω αυτής της τροποποίησης, υλοποιήθηκε η λογική ελέγχου λειτουργίας των επιμέρους στοιχείων του μεταγωγέα. Ο κώδικας που ακολουθεί είναι κομμάτι του αρχείου *~ns/classifier/classifier.cc*

Αρχικά, κάθε δήλωση νέου κόμβου στο NS2, καλεί την μέθοδο `command()`. Για κάθε αποδεκτή εντολή του συστήματος, υπάρχει ένα αντίστοιχο κομμάτι κώδικα που πρέπει να εκτελεστεί. Έτσι όταν ο μεταγλωττιστής Tcl διαβάσει την δεσμευμένη λέξη “`Switch_mode`”, σε επίπεδο C++ η μέθοδος `command()` αρχικοποιεί τον μεταγωγέα, δεσμεύοντας χώρο μνήμης για τις ουρές, σύμφωνα με το πλήθος πορτών εισόδου-εξόδου και το μέγεθος ουράς που έχει ορισθεί από την προσομοίωση. Επίσης, αρχικοποιείται ο ονομαστικός ρυθμός μεταφοράς δεδομένων του μεταγωγέα καθώς και οι χρονοδιακόπτες που ελέγχουν κάθε πόρτα εξόδου (`output_timers[]`).

```
if (strcmp(argv[1], "Switch_mode") == 0){
      int input_rows=atoi(argv[2]);
      int output_cols=atoi(argv[3]);      //not used
      float memory_size=atof(argv[4]); //argv[4]:size of queue in MB,
      float bandwidth_bits_sec = atol(argv[5]); //
      printf("mem:::%f\n",memory_size);
      int cell_size = memory_size*8000000; //cell_size: size of queue
in bits

      if(!(input_rows>0 && output_cols>0 && cell_size>0 &&
bandwidth_bits_sec>0)) {
            printf("Switch node couldn't initilalize properly!\n");
            return (TCL_ERROR);
      }

      switch_node.sw_nodeInit(input_rows, output_cols, cell_size,
bandwidth_bits_sec);


      //SWITCH
      sw_basic_timer.InitSwTimer(this, SW_BASIC_TIMER);
      output_timers = new sw_timer[switch_node.getOutPortsNum()];
      prio_to_send_per_port = new int[switch_node.getOutPortsNum()];

      for(int i=0; i<switch_node.getOutPortsNum(); i++){
```

```
            output_timers[i].InitSwTimer(this, i);
            prio_to_send_per_port[i]=0;
      }
      //SWITCH
      switch_enable = true;

      printf("switch_enabled with : %d %d %d
%f\n",input_rows,output_cols,cell_size,bandwidth_bits_sec);
      return TCL_OK;
}
```

Από την στιγμή που ένας κόμβος έχει οριστεί ως μεταγωγέας, (είναι αληθής η μεταβλητή `switch_enable`), η μέθοδος `recv()`, που καλείται όταν ο κόμβος λάβει ένα πακέτο, εκτελεί τον παρακάτω τροποποιημένο κώδικα, αντί του αρχικού.

```
if(switch_enable){
      /* to neo paketo dimiourgei ena antikeimeno sw_packet pou
       * orizetai sto sw_queue
       */
      tmpPackHandler.packet = p;
      tmpPackHandler.packet_handler = h;

      //anagnwsi source kai dest
      int src = (int)(HDR_IP(p)->saddr());
      int dst = (int)(HDR_IP(p)->daddr());

      if (dst==-1){
            if(switch_node.sw_insertbrdcstPacket(&tmpPackHandler)) {
                  sw_SendPackets(SW_BASIC_TIMER);
            }
            else{
                  Packet::free(p);
            }
            return;
      }

      if(switch_node.sw_insertPacket(dst, &tmpPackHandler)) {
            sw_SendPackets(SW_BASIC_TIMER);
      }
      else{
            Packet::free(p);
      }
}
```

Αρχικά πρέπει να αναγνωστεί η διεύθυνση προορισμού του πακέτου. Αν η διεύθυνση έχει την τιμή “-1”, έχει οριστεί κατά σύμβαση ότι το πακέτο είναι broadcast τύπου και πρέπει να εισαχθεί στην ουρά που εξυπηρετεί broadcasting στον μεταγωγέα. Αλλιώς, πρέπει να γίνει εισαγωγή του πακέτου στην ουρά εξόδου, που αντιστοιχεί στην διεύθυνση προορισμού του πακέτου.

Σε κάθε περίπτωση, αν η εισαγωγή ήταν επιτυχής, δηλαδή αν υπήρχε διαθέσιμος χώρος στην ουρά, καλείται η μέθοδος `sw_SendPackets()` η οποία είναι υπεύθυνη για τον χρονοπρογραμματισμό της αποστολής του πακέτου.

Η μέθοδος `sw_SendPackets()` έχει προστεθεί στην βασική λειτουργικότητα του ταξινομητή. Η `sw_SendPackets()` καλείται πάντα με όρισμα ακέραιο αριθμό, ο οποίος αντιπροσωπεύει το id ενός χρονοδιακόπτη. Οι πιθανές περιπτώσεις κλήσης της είναι τρεις:

- αμέσως μετά την εισαγωγή ενός πακέτου σε κάποια ουρά, με όρισμα το `SW_BASIC_TIMER` που έχει αρχικοποιηθεί στην τιμή -1 (αρχείο *sw_timer.h*)
- αμέσως μετά την ολοκλήρωση της αποστολής ενός πακέτου από κάποια ουρά με όρισμα το id του χρονοδιακόπτη της ουράς, αν η ουρά περιέχει και άλλα πακέτα προς αποστολή
- αμέσως μετά την ολοκλήρωση της αποστολής ενός πακέτου από κάποια ουρά με όρισμα το `SW_BASIC_TIMER` (-1), αν η ουρά έχει αδειάσει

```
void Classifier::sw_SendPackets(int timer_id){
      int packet_size = 0;
      double delta_t = 0.0;
      int col_num = switch_node.getOutPortsNum();
```

Όταν η `sw_SendPackets()` κληθεί με όρισμα -1, γίνεται έλεγχος σε όλες τις ουρές του μεταγωγέα για το αν περιέχονται πακέτα προς αποστολή. Συγκεκριμένα, για κάθε πόρτα εξόδου, αν ο χρονοδιακόπτης της βρίσκεται σε ανενεργή κατάσταση (`TIMER_IDLE`), δηλαδή δεν υπάρχει κάποια αποστολή σε εξέλιξη, τότε αρχικά επιλέγεται ποια από τις 8 ουρές διαφορετικών IP προτεραιοτήτων πρέπει να εξυπηρετηθεί από την πόρτα αυτή.

Αυτό γίνεται με την κλήση της μεθόδου `returnQueueOfPriority()`, η οποία ελέγχει τις ουρές της πόρτας και επιστρέφει την τιμή 0-7 ανάλογα με το υψηλότερο επίπεδο πακέτου που βρίσκεται σε αναμονή.

Στην συνέχεια, η μέθοδος `returnFirstPacketSize()` επιστρέφει το μέγεθος (σε bits) του πρώτου προς αποστολή πακέτου για την συγκεκριμένη πόρτα και για το συγκεκριμένο επίπεδο προτεραιότητας.

Διαιρώντας το μέγεθος του πακέτου με τον ρυθμό μετάδοσης δεδομένων του μεταγωγέα, ο οποίος έχει ορισθεί στην αρχή της προσομοίωσης, υπολογίζεται ο χρόνος που απαιτείται από την έναρξη ως την ολοκλήρωση της μετάδοσης του πακέτου. Αν αυτός ο χρόνος δεν είναι μηδενικός (ουσιαστικά αν υπάρχει πακέτο προς μετάδοση), τότε προγραμματίζεται η λήξη του χρονοδιακόπτη της πόρτας, μετά από καθυστέρηση ίση με τον χρόνο που υπολογίστηκε προηγουμένως.

Πλέον η πόρτα εξόδου θεωρείται ότι βρίσκεται σε κατάσταση αποστολής.

```
if (timer_id==SW_BASIC_TIMER){
      for(int col_itr=0; col_itr<(col_num*8-1); col_itr=col_itr+8) {
            if(output_timers[col_itr/8].status() ==
TimerHandler::TIMER_IDLE ){
                  prio_to_send_per_port[col_itr/8] =
switch_node.returnQueueOfPriority(col_itr/8);

                  packet_size =
switch_node.returnFirstPacketSize(col_itr/8,
prio_to_send_per_port[col_itr/8]);

                  delta_t = packet_size/switch_node.getBandwidth();

                  if(delta_t!=0){
                        output_timers[col_itr/8].resched(delta_t);
                  }
            }
      }
```

Η ίδια διαδικασία που περιγράφηκε προηγουμένως, πρέπει να εκτελεστεί και για τον χρονοδιακόπτη ελέγχου των broadcast αποστολών.

```
if(output_timers[col_num-1].status()==TimerHandler::TIMER_IDLE ){
      packet_size = switch_node.returnFirstPacketSizeBrdcst();
      delta_t = packet_size/switch_node.getBandwidth();

      if(delta_t!=0){
            output_timers[col_num-1].resched(delta_t);

            for(int col_itr2=0; col_itr2<(col_num-1); col_itr2++){

                  if (output_timers[col_itr2].status() ==
            TimerHandler::TIMER_PENDING) {
                        output_timers[col_itr2].cancel();
                  }
            }

      }
}
}
```

Όταν η sw_SendPackets() κληθεί με όρισμα (col_num-1), που είναι η τιμή που αντιστοιχεί στον χρονοδιακόπτη που ελέγχει τις broadcast αποστολές, πρέπει να γίνει η αποστολή του πρώτου πακέτου της broadcast ουράς προς όλες τις πόρτες εξόδου. Επομένως, αρχικά γίνεται ανάκτηση του πακέτου

```
else
      if (timer_id==(col_num-1)){
            sw_packet* removed_packet;

            removed_packet = switch_node.sw_removebrdcstPacket();
            NsObject* node = NULL;
            Packet* p = (Packet*)removed_packet->packet;
            Handler* h = (Handler*)removed_packet->packet_handler;
```

Στην συνέχεια με μια επανάληψη για όλες τις πόρτες εξόδου του μεταγωγέα, γίνεται έλεγχος αν η κάθε επιμέρους έξοδος χρησιμοποιείται (αν υπάρχει κόμβος του δικτύου που να είναι συνδεδεμένος με τον μεταγωγέα σε αυτή την έξοδο). Αν η έξοδος δεν χρησιμοποιείται, δεν γίνεται τίποτα και η επανάληψη συνεχίζει για τις υπόλοιπες πόρτες εξόδου. Αν η έξοδος χρησιμοποιείται, τότε πρέπει να τροποποιηθεί η διεύθυνση προορισμού του πακέτου με νέα τιμή την διεύθυνση του κόμβου που είναι συνδεδεμένος στην έξοδο. Από αυτό το σημείο και μετά πρέπει να ακολουθηθεί η τυπική διαδικασία αποστολής πακέτου από κόμβο για το NS2, έτσι, πρέπει να βρεθεί ο κόμβος στο δίκτυο με την χρήση της μεθόδου find() και να αντιστοιχιστεί στο αντικείμενο node τύπου NsObject. Στην συνέχεια πρέπει να κληθεί η μέθοδος recv() από αυτόν τον κόμβο, έχοντας σαν ορίσματα ένα αντίγραφο του πακέτου και τον διαχειριστή του πακέτου (Handler).

```
      for (int i = 0; i < (col_num*8-1); i=i+8) {
            hdr_ip* ipret = hdr_ip::access(p);
            if(switch_node.isOutputPortUsed(i/8)){
                  ipret->daddr() = i/8;
                  node = find(p);
                  node->recv(p->copy(), h);
                  node=NULL;
            }
      }
```

Αφού γίνει η αποστολή προς όλες τις εξόδους, πρέπει να προγραμματιστεί η επόμενη αποστολή για την broadcast ουρά.

Αν υπάρχουν ακόμα broadcast πακέτα για αποστολή, ο χρονοδιακόπτης ελέγχου της ουράς ορίζεται να λήξει σε χρόνο ίσο με `delta_t` (όπως προηγουμένως). Επίσης ελέγχεται η περίπτωση να εκκρεμούν αποστολές από τις υπόλοιπες ουρές, δηλαδή κάποιος από τους υπόλοιπους χρονοδιακόπτες να βρίσκεται σε ενεργή κατάσταση (`TIMER_PENDING`). Αν ναι, τότε επειδή οι broadcast αποστολές έχουν υψηλότερη προτεραιότητα, όλοι οι υπόλοιποι χρονοδιακόπτες ακυρώνονται.

Αν δεν υπάρχουν άλλα πακέτα για broadcast, καλείται εκ νέου η `sw_SendPackets()` με όρισμα -1, έτσι ώστε να ξεκινήσει η διαδικασία αποστολής από την αρχή.

```
      packet_size = switch_node.returnFirstPacketSizeBrdcst();
      delta_t = packet_size/switch_node.getBandwidth();
      if(delta_t!=0){
            output_timers[col_num-1].resched(delta_t);
            for(int col_itr2=0; col_itr2<(col_num-1); col_itr2++){
                  if (output_timers[col_itr2].status() ==
TimerHandler::TIMER_PENDING) {
                        output_timers[col_itr2].cancel();
                  }
            }
      }else{
            sw_SendPackets(SW_BASIC_TIMER);
      }

}
```

Τέλος, η `sw_SendPackets()` μπορεί να κληθεί με όρισμα έναν ακέραιο που αντιστοιχεί στον χρονοδιακόπτη μιας συγκεκριμένης εξόδου που μόλις έληξε. Σε αυτήν την περίπτωση, βρίσκεται ο κόμβος στον οποίο πρέπει να γίνει η αποστολή του πακέτου. Αν ο κόμβος υπάρχει καλείται η μέθοδος `recv()` ώστε να γίνει η παραλαβή του πακέτου από τον κόμβο.

Στην συνέχεια ελέγχεται αν στην συγκεκριμένη ουρά υπάρχουν και άλλα πακέτα για αποστολή. Αν ναι επαναπρογραμματίζεται ο χρονοδιακόπτης της ουράς να λήξει σε χρόνο ίσο με το μέγεθος του πακέτου προς τον ρυθμό μετάδοσης δεδομένων του μεταγωγέα.

```
else{
      sw_packet* removed_packet;
      removed_packet = switch_node.sw_removePacket(timer_id,
prio_to_send_per_port[timer_id]);

      NsObject* node = find((Packet*)(removed_packet->packet));
```

```
	if (node == NULL) {
		Packet::free((Packet*)(removed_packet->packet));
	}else{
		node->recv((Packet*)(removed_packet-
>packet),(Handler*)(removed_packet->packet_handler));
	}

	prio_to_send_per_port[timer_id] =
switch_node.returnQueueOfPriority(timer_id);

	packet_size = switch_node.returnFirstPacketSize(timer_id,
prio_to_send_per_port[timer_id]);

	delta_t = packet_size/switch_node.getBandwidth();
	if(delta_t!=0){
		output_timers[timer_id].resched(delta_t);
	}
}

}
```

3.3.5. Άλλες τροποποιήσεις του συστήματος NS2

Για την υλοποίηση προσομοιώσεων που περιέχουν και broadcast αποστολές πακέτων ήταν απαραίτητο να δημιουργηθεί ένα νέο πρωτόκολλο για το NS2 το οποίο να δημιουργεί κατά σύμβαση πακέτα με διεύθυνση προορισμού -1. Η τιμή -1 στην διεύθυνση προορισμού ενός πακέτου, ορίστηκε να σηματοδοτεί ότι το συγκεκριμένο πακέτο, αφού ληφθεί από τον κόμβο-μεταγωγέα πρέπει να προωθηθεί σε όλες τις εξόδους του.

Για την δημιουργία ενός τέτοιου πρωτοκόλλου, τροποποιήθηκε το ήδη υπάρχον πρωτόκολλο-παράδειγμα του NS2 που ορίζεται από την κλάση `PingAgent`. Ο πράκτορας που οριζόταν από την συγκεκριμένη κλάση χρησιμοποιούνταν για την υλοποίηση της μεθόδου ping (δημιουργία και αποστολή πακέτων δεδομένων σε έναν κόμβο του δικτύου και στην συνέχεια αναμονή για την απάντηση από τον κόμβο). Το αρχείο που τροποποιήθηκε είναι το *~ns/apps/ping.cc*.

Συγκεκριμένα στον κώδικα της μεθόδου `command()` προστέθηκαν οι ακόλουθες γραμμές, αμέσως μετά την δημιουργία ενός νέου πακέτου με τυχαία δεδομένα:

```
hdr_ip* iph = HDR_IP(pkt);
iph->daddr() = IP_BROADCAST;
iph->dport() = iph->sport();
```

Έτσι, ο πράκτορας μετά από κάθε δημιουργία πακέτου κατά την διάρκεια της προσομοίωσης, τροποιεί την διεύθυνση προορισμού του πακέτου στην τιμή IP_BROADCAST που έχει οριστεί σε -1. Στην συνέχεια, γίνεται η αποστολή του πακέτου χωρίς τροποποιήσεις.

Η μέθοδος recv() παρέμεινε ίδια εκτός του ότι απενεργοποιήθηκε η αποστολή πακέτου-απάντησης.

3.4 Τροποποιήσεις σε TCL

Για την υποστήριξη του νέου τύπου κόμβου που δημιουργήθηκε, έπρεπε να τροποποιηθεί το αρχείο *~ns/tcl/lib/ns-node.tcl* έτσι ώστε το σύστημα να υποστηρίζει την εντολή Switch_mode με την οποία θα δημιουργείται ένας κόμβος τύπου μεταγωγέα κατά την υλοποίηση της προσομοίωσης σε Tcl. Έτσι προστέθηκαν οι ακόλουθες γραμμές κώδικα:

```
Node instproc Switch_mode { row cell_size bandwidth }
{
     set bw_bits_per_second [expr [bw_parse $bandwidth]]
     [$self set classifier_] Switch_mode $row $cell_size
$bw_bits_per_second
}
```

Η νέα εντολή Switch_mode δέχεται 3 ορίσματα, τα οποία είναι με την ακόλουθη σειρά, το πλήθος των εισόδων/εξόδων του μεταγωγεά, το μέγεθος των ουρών σε megabytes και ο ρυθμός λειτουργίας του σε bits/sec.

Επίσης ορίστηκε η εντολή switch_stats μέσω της οποίας καλείται η μέθοδος printQueueStats() της κλάσης switch_node, για την εμφάνιση των αποτελεσμάτων στο τέλος της προσομοίωσης:

```
Node instproc switch_stats { info_id } {
     [$self set classifier_] switch_stats $info_id
}
```

3.5 Ανάπτυξη κώδικα προσομοίωσης σε TCL

Για την δημιουργία προσομοιώσεων, το σύστημα NS2 χρησιμοποιεί την γλώσσα Tcl. Έτσι αναπτύχθηκε σε Tcl μια τοπολογία δικτύου με κόμβους που παράγουν πακέτα, συνδεδεμένους στις πόρτες εισόδου ενός κόμβου-

μεταγωγέα και κόμβους που δέχονται τα πακέτα συνδεδεμένους στις πόρτες εξόδου του. Για την παραγωγή broadcast πακέτων στο δίκτυο, χρειάστηκε ένας επιπλέον κόμβος προορισμένος για αυτή την λειτουργία.

Μετά την δημιουργία των κόμβων και την διασύνδεσή τους, πρέπει να αντιστοιχιστεί ένας πράκτορας σε κάθε κόμβο για την υλοποίηση ενός πρωτοκόλλου δικτύου. Για την δημιουργία κίνησης στο δίκτυο, χρησιμοποιήθηκε ο τύπος πράκτορα (agent) UDP.

Το πρωτόκολλο UDP (User Datagram Protocol) είναι ένα από τα βασικά πρωτόκολλα που χρησιμοποιούνται στο Διαδίκτυο. Ένα από τα κύρια χαρακτηριστικά του UDP είναι ότι δεν εγγυάται αξιόπιστη επικοινωνία. Τα πακέτα UDP που αποστέλλονται από έναν υπολογιστή μπορεί να φτάσουν στον παραλήπτη με λάθος σειρά, διπλά ή να μην φτάσουν καθόλου εάν το δίκτυο έχει μεγάλο φόρτο.

Αντίθετα με το πρωτόκολλο TCP, το πρωτόκολλο UDP δεν διαθέτει μηχανισμούς ελέγχου και επιβολής της αξιοπιστίας. Η έλλειψη των μηχανισμών αυτών, το καθιστά αρκετά πιο γρήγορο και αποτελεσματικό, τουλάχιστον για τις εφαρμογές εκείνες που δεν απαιτούν αξιόπιστη επικοινωνία. Στις προσομοιώσεις που έγιναν, μας ενδιέφερε να μετρήσουμε στατιστικά στοιχεία για την λειτουργία του μεταγωγέα και μόνο (όχι του συνόλου του δικτύου). Επομένως, δεν χρησιμοποιήθηκαν εξωγενείς παράγοντες (θόρυβος, κτλ) που μπορεί να προκαλέσουν απώλειες πακέτων, άρα δεν ήταν απαραίτητος ο έλεγχος αξιοπιστίας. Για αυτό τον λόγο επιλέχθηκε το πρωτόκολλο UDP και χρησιμοποιήθηκε ο αντίστοιχος τύπος πράκτορα (agent) για τους κόμβους που αποστέλλουν πακέτα στον μεταγωγέα.

Με τις ακόλουθες γραμμές κώδικα Tcl, δηλώνεται ένας κόμβος στο σύστημα, δηλώνεται ένα πράκτορας τύπου UDP και στη συνέχεια, επισυνάπτεται ο πράκτορας στον κόμβο:

```
set source0 [$ns node]
set   udp1 [new Agent/UDP]
$ns attach-agent $source0 $udp1
```

Έτσι πλέον ο κόμβος έχει οριστεί να δημιουργεί και να στέλνει πακέτα σύμφωνα με το πρωτόκολλο UDP.

Για την καθεαυτού δημιουργία των πακέτων, πρέπει να χρησιμοποιηθεί μια εφαρμογή (application). Ο τύπος εφαρμογής που μπορεί να χρησιμοποιηθεί εξαρτάται από τον πράκτορα του κόμβου, όπως είδαμε στο κεφάλαιο 2. Έτσι χρησιμοποιήθηκε τύπος παραγωγού κίνησης στο δίκτυο (traffic generator). Συγκεκριμένα χρησιμοποιήθηκε ο τύπος CBR που παράγει πακέτα σύμφωνα με έναν σταθερό ρυθμό.

Με τις ακόλουθες γραμμές κώδικα Tcl, δηλώνεται μια νέα εφαρμογή τύπου CBR η οποία προσαρτάται στον ήδη δηλωμένο πράκτορα. Επίσης, ορίζονται το μέγεθος των πακέτων (`packetsize_`) σε bytes, ο χρόνος ανάμεσα σε δύο διαδοχικές αποστολές πακέτων (`interval_`) σε seconds. Τέλος ορίζεται ο χρόνος έναρξης και λήξης της λειτουργίας της εφαρμογής:

```
set e1 [new Application/Traffic/CBR]
$e1 attach-agent $udp1
$e1 set packetSize_ 250
$e1 set interval_ 0.004
$ns at 0.0 "$e1 start"
$ns at $end_time "$e1 stop"
```

Αντί του χρονικού διαστήματος ανάμεσα σε δυο αποστολές, μπορεί να οριστεί ο ρυθμός αποστολής δεδομένων από την εφαρμογή, με την χρήση του χαρακτηριστικού `rate_`. Η χρήση του γίνεται ως εξής:

```
$e1 set rate_ 6000Kb
```

Εκτός από την παραγωγή κίνησης στο δίκτυο, πρέπει να υπάρχουν κόμβοι που θα «καταναλώνουν» την κίνηση, δηλαδή θα λαμβάνουν τα πακέτα που στέλνει ο μεταγωγέας. Για τον σκοπό αυτό, το σύστημα NS2 παρέχει κόμβους-καταβόθρες. Με τις ακόλουθες γραμμές κώδικα, δηλώνεται ένας νέος κόμβος (`$sink1`), προσαρτάται στον κόμβο ένας πράκτορας τύπου UDP και τέλος συνδέεται ο πράκτορας-παραγωγός πακέτων `$udp1` (έχει δηλωθεί προηγουμένως) με τον πράκτορα-δέκτη `$sink1ag`.

```
set sink1 [$ns node]
set sink1ag [new Agent/UDP]
$ns attach-agent $sink1 $sink1ag
$ns connect $udp1 $sink1ag
```

Έτσι έχει οριστεί μια ροή πακέτων (flow) ανάμεσα στους δύο κόμβους, οι οποίοι συνδέονται μέσω του μεταγωγέα.

Ο κόμβος που προσομοιώνει τον μεταγωγέα δηλώνεται ως εξής:

`$sw_node Switch_mode $port_i $queue_size $switch_bw`
Όπου:

- `$port_i` είναι το πλήθος των εισόδων/εξόδων του μεταγωγέα,
- `$queue_size` είναι το μέγεθος κάθε ουράς προτεραιότητας κάθε πόρτας εξόδου σε megabytes
- `$switch_bw` είναι ο ονομαστικός ρυθμός μεταφοράς δεδομένων του μεταγωγέα σε bits/second

Οι συνδέσεις ανάμεσα στους κόμβους του δικτύου και τον μεταγωγέα δηλώνονται ως εξής:
`$ns duplex-link $source0 $sw_node $bw $link_delay DropTail`
Τα δύο πρώτα ορίσματα είναι τα ονόματα των κόμβων που συνδέονται με μια αμφίδρομη σύνδεση (duplex-link).

- `$bw` είναι το εύρος της ζώνης της σύνδεσης σε bits/second
- `$link_delay` είναι καθυστέρηση στην μετάδοση των δεδομένων που προσθέτει η σύνδεση σε seconds και
- `DropTail` ο τύπος ουράς που χρησιμοποιεί η σύνδεση σε περίπτωση υπερχείλισης.

Στις συνδέσεις που χρησιμοποιήθηκαν στις προσομοιώσεις, το εύρος ζώνης τους ορίστηκε σε μια πολύ μεγάλη τιμή και η καθυστέρηση σε πολύ μικρή ούτως ώστε να μην επηρεάζονται οι μετρήσεις από εξωγενείς παράγοντες (εκτός του μεταγωγέα).

4 Πειραματική αξιολόγηση

4.1 Γενικά

Για τον έλεγχο λειτουργίας και την αξιολόγηση του μεταγωγέα, πραγματοποιήθηκε ένα πλήθος προσομοιώσεων πάνω σε δύο διαφορετικά σενάρια λειτουργίας της τοπολογίας προσομοίωσης. Στην πρώτη περίπτωση έγιναν προσομοιώσεις της λειτουργίας σε συνθήκες κανονικού φόρτου εργασίας, με περιόδους χαμηλού, μέσου και υψηλού ρυθμού παραγωγής δεδομένων από τους κόμβους-αποστολείς του δικτύου, ενώ στην δεύτερη περίπτωση προσομοιώθηκε η λειτουργία του μεταγωγέα σε συνθήκες υψηλού φόρτου, δηλαδή όταν δέχεται συνεχώς δεδομένα με ρυθμό μεγαλύτερο από τον ονομαστικό ρυθμό λειτουργίας του.

Το δίκτυο της προσομοίωσης, περιλάμβανε κόμβους-αποστολείς unicast και broadcast πακέτων, οι οποίοι συνδέονταν μέσω ενός κόμβου-μεταγωγέα με κόμβους-παραλήπτες. Οι προσομοιώσεις επαναλήφθηκαν για διαφορετικά ποσοστά κίνησης broadcast πακέτων στο δίκτυο αλλά και για διαφορετικά μεγέθη ουρών.

Τα μεγέθη που μετρήθηκαν κατά τις προσομοιώσεις είναι τα παρακάτω:

- το ποσοστό των πακέτων που απορρίπτονται από κάθε ουρά εξόδου, σε σχέση με το συνολικό πλήθος πακέτων που έπρεπε να παραληφθούν και στην συνέχεια να αποσταλούν από την συγκεκριμένη έξοδο.
- Η μέση καθυστέρηση ανά πακέτο, δηλαδή ο μέσος χρόνος που παραμένει κάθε πακέτο μέσα στην ουρά, από την στιγμή που αφίχθηκε μέχρι την στιγμή που έγινε η αποστολή του

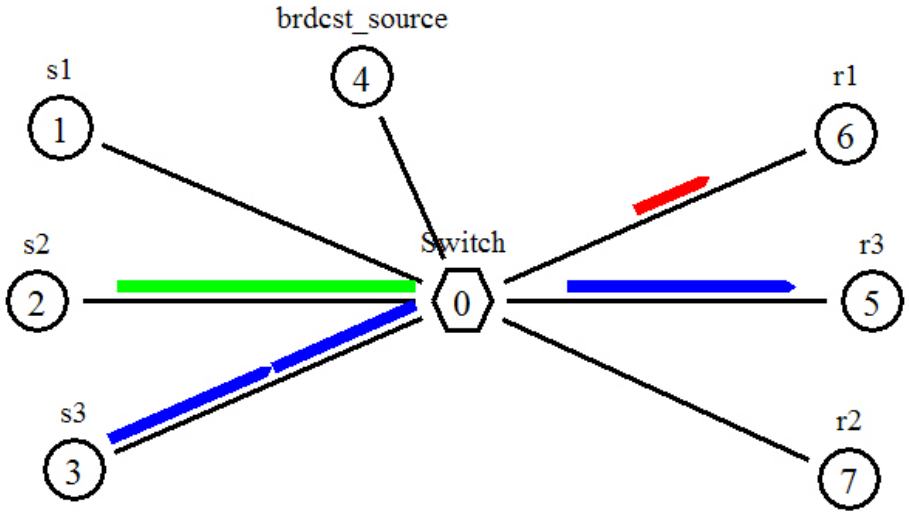
- Το throughput κάθε ουράς, δηλαδή το συνολικό πλήθος των bytes πληροφορίας που αποθηκεύτηκε και στην συνέχεια βγήκε από την ουρά κατά την διάρκεια της προσομοίωσης.
- Το throughput κάθε πόρτας εξόδου, το οποίο είναι το σύνολο των bytes δεδομένων που πέρασαν από την πόρτα (unicast και broadcast πακέτα).

Για την οπτική απεικόνιση της λειτουργίας των δικτύων που προσομοιώθηκαν, χρησιμοποιήθηκε το εργαλείο NAM (Network Animator) που βασίζεται σε TCL και χρησιμοποιείται για την οπτικοποίηση δικτυακών προσομοιώσεων που παράγονται από το NS2. Συγκεκριμένα, το NAM δέχεται σαν είσοδο αρχεία trace, τα οποία περιέχουν πληροφορίες σχετικά με την τοπολογία της προσομοίωσης (κόμβοι, συνδέσεις κτλ) καθώς και πληροφορίες για τις κινήσεις των πακέτων εντός της τοπολογίας. Μέσω της διεπαφής χρήστη, το NAM παρέχει δυνατότητες για τον έλεγχο της προσομοίωσης

4.2 Κανονικός φόρτος εργασίας

Στις προσομοιώσεις λειτουργίας του μεταγωγέα σε κανονικό φόρτο λειτουργίας, η τοπολογία αποτελούνταν από τρείς κόμβους-αποστολείς που έστελναν unicast πακέτα σε τρεις άλλους κόμβους-παραλήπτες, ταυτόχρονα με ένα τέταρτο αποστολέα broadcast πακέτων. Ένα στιγμιότυπο προσομοίωσης της παραπάνω τοπολογίας φαίνεται στην εικόνα 10.

Κάθε ένας από τους τρεις κόμβους παραγωγής unicast πακέτων, ορίστηκε να παράγει πακέτα διαφορετικών μεγεθών. Συγκεκριμένα, ένας κόμβος παρήγαγε μικρά πακέτα μεγέθους 250 bytes, ο δεύτερος κόμβος παρήγαγε μεγάλα πακέτα μεγέθους 1000 bytes και ο τρίτος κόμβος λειτουργούσε πιο ρεαλιστικά, καθώς παρήγαγε πακέτα των οποίων το μέγεθος προέκυπτε από μια κανονική κατανομή με μέση τιμή 500 bytes και τυπική απόκλιση 50 bytes.



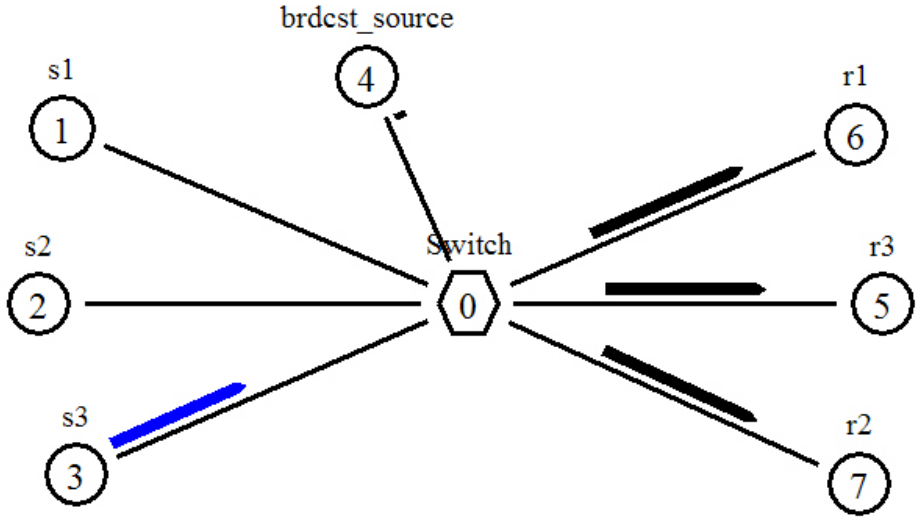
Εικόνα 10: Στιγμιότυπο προσομοίωσης λειτουργίας του μεταγωγέα με φυσιολογικό φόρτο εργασίας

Οι τρείς unicast κόμβοι, δεν παρήγαγαν πακέτα με σταθερό ρυθμό. Ο ρυθμός παραγωγής πακέτων ορίστηκε να αλλάζει κάθε 0.5 δευτερόλεπτα. Αυτό γίνεται με την αλλαγή του χαρακτηριστικού `interval_` κάθε εφαρμογής (χρονικό διάστημα ανάμεσα σε δύο διαδοχικές αποστολές πακέτων), το οποίο ορίστηκε να παίρνει ισοπίθανα τυχαίες τιμές ανάμεσα σε 0.001 και 0.099. Έτσι αντίστοιχα, ο κάθε κόμβος παράγει πακέτα με ρυθμό από 10 έως 1000 πακέτα ανά δευτερόλεπτο. Επομένως, ο μέσος ρυθμός παραγωγής πακέτων είναι περίπου 500 πακέτα ανά δευτερόλεπτο. Με βάση αυτόν τον ρυθμό, ο μέσος ρυθμός αποστολής δεδομένων από κάθε κόμβο στον μεταγωγέα θα είναι 0.125Mbytes/sec για τον κόμβο 1, 0.5Mbyte/sec για τον κόμβο 2 και 0.25Mbyte/sec για τον κόμβο 3 ή 1Mbit/sec, 4Mbit/sec και 2Mbit/sec αντίστοιχα. Τέλος, ο μέγιστος δυνατός ρυθμός αποστολής δεδομένων από κάποιον κόμβο είναι 8Mbit/sec για τον κόμβο 2.

Ο ρυθμός λειτουργίας του μεταγωγέα έχει οριστεί στα 10Mbit/sec για κάθε έξοδο. Επομένως, σύμφωνα με τα παραπάνω, ο μεταγωγέας δέχεται unicast δεδομένα με ρυθμό μικρότερο από τον ρυθμό λειτουργίας του ανά πόρτα.

Ταυτόχρονα, στην τοπολογία λειτουργούσε και ένας τέταρτος κόμβος-αποστολέας broadcast πακέτων. Ο ρυθμός αποστολής πακέτων του κόμβου αυτού ήταν πολύ χαμηλότερος σε σχέση με τον ρυθμό αποστολής των

unicast κόμβων. Συγκεκριμένα, προσομοιώθηκε η λειτουργία του μεταγωγέα όταν το πλήθος των broadcast πακέτων είναι το α)6.7% και β)16.5% των συνολικών πακέτων που κυκλοφορούν στο τοπικό δίκτυο. Η εικόνα 11 παρουσιάζει ένα στιγμιότυπο της προσομοίωσης, αμέσως μετά από μια broadcast αποστολή από τον μεταγωγέα. Το μέγεθος των broadcast πακέτων ήταν 500 bytes.



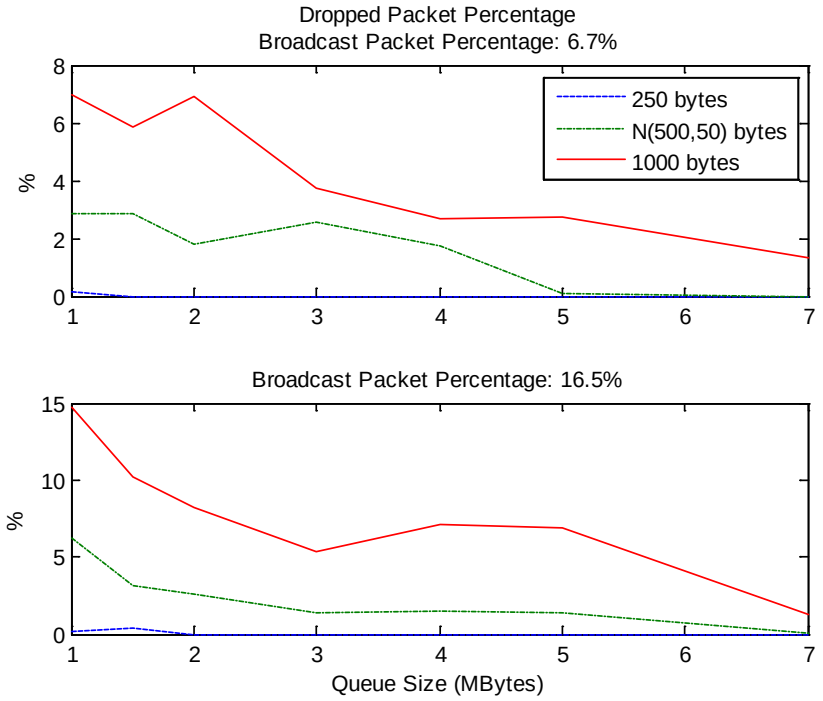
Εικόνα 11: Στιγμιότυπο broadcast αποστολής από τον μεταγωγέα

Οι προσομοιώσεις έγιναν για διάφορα μεγέθη μνήμης για τις ουρές του μεταγωγέα. Συγκεκριμένα προσομοιώθηκε η λειτουργία του μεταγωγέα για μεγέθη 1, 1.5, 2, 3, 4, 5 και 7 mB ανά ουρά. Η διάρκεια των προσομοιώσεων ορίστηκε σε 20 sec.

Επειδή ο ρυθμός παραγωγής πακέτων κάθε κόμβου δεν ήταν σταθερός αλλά έπαιρνε τυχαίες τιμές κατά την διάρκεια της προσομοίωσης, θεωρήθηκε στατιστικά ορθότερο να πραγματοποιηθεί ένα πλήθος προσομοιώσεων και στην συνέχεια να εξαχθεί ο μέσος όρος κάθε μετρούμενου μεγέθους. Έτσι, διεξήχθησαν 20 προσομοιώσεις για κάθε διαφορετική περίπτωση ρυθμίσεων της τοπολογίας (μέγεθος ουρών, ποσοστό broadcast πακέτων). Εξαιτίας αυτής της τυχαίας διαδικασίας μέσω της οποίας προέκυπτε ο ρυθμός παραγωγής πακέτων, το ποσοστό των broadcast πακέτων που παράγονταν ανά προσομοίωση μπορούσε να οριστεί μόνο κατά προσέγγιση κατά την αρχικοποίηση των προσομοιώσεων. Σε συνδυασμό με

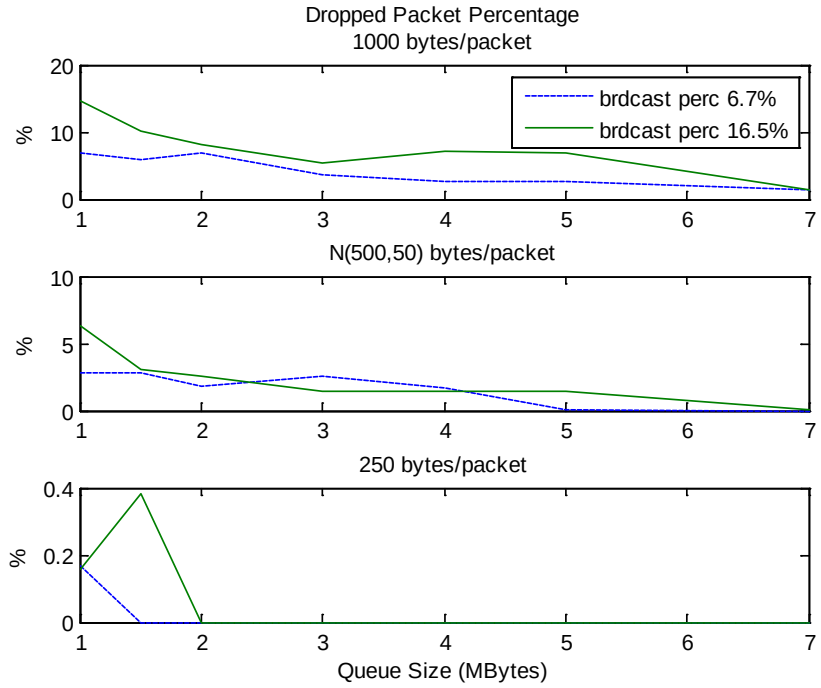
το πλήθος των προσομοιώσεων, προέκυψαν στο τέλος τα συγκεκριμένα (μέσα) ποσοστά broadcast πακέτων (6.7% και 16.5%).

Η εικόνα 12 παρουσιάζει την μεταβολή του ποσοστού απορριφθέντων πακέτων, σε συνάρτηση με το μέγεθος ουράς. Όπως ήταν αναμενόμενο, η αύξηση του μεγέθους της κάθε ουράς μειώνει το ποσοστό των πακέτων που απορρίπτονται στην είσοδο του μεταγωγέα, αφού έτσι η κάθε ουρά μπορεί να δεχθεί περισσότερα πακέτα έως ότου υπερχειλίσει.



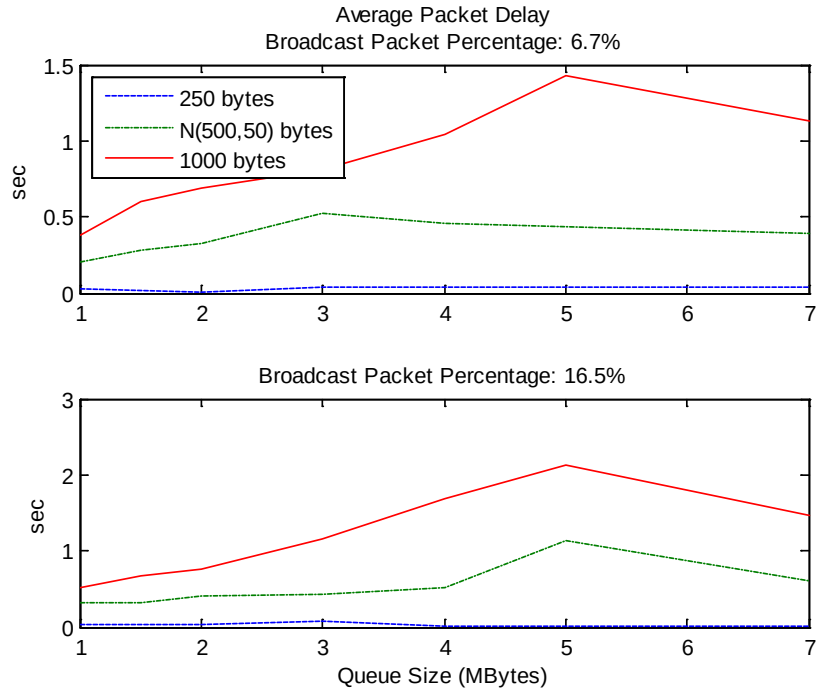
Εικόνα 12: Ποσοστό απορριφθέντων πακέτων για τα ποσοστά broadcast πακέτων που προσομοιώθηκαν

Συγκρίνοντας ανά δύο τις προηγούμενες γραφικές παραστάσεις στην εικόνα 13, για τα μεγέθη unicast πακέτων που χρησιμοποιήθηκαν, παρατηρούμε ότι η απόρριψη πακέτων είναι μεγαλύτερη όταν μία ουρά δέχεται μεγάλα πακέτα



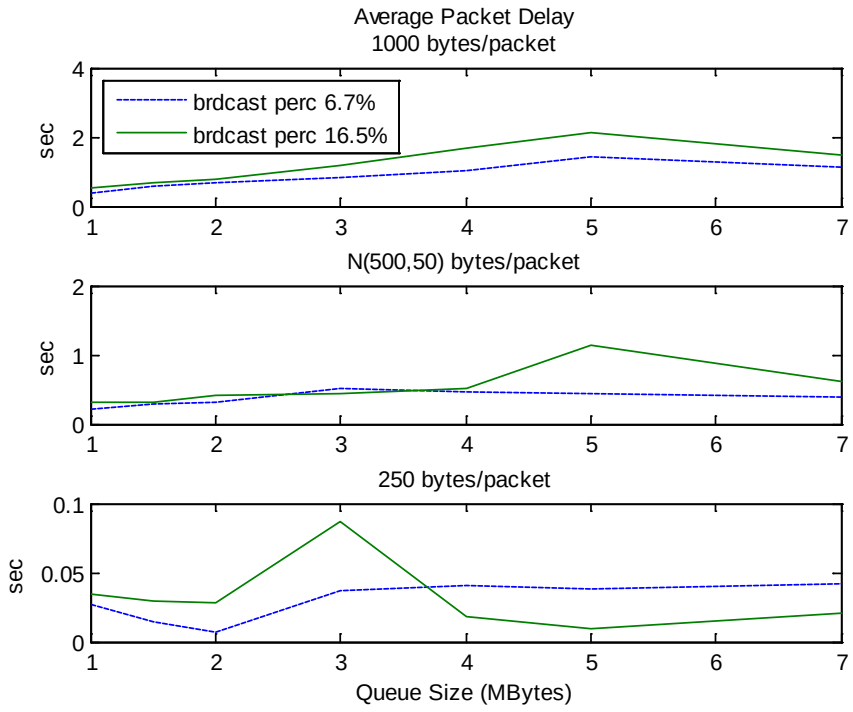
Εικόνα 13: Ποσοστό απορριφθέντων πακέτων για τα μεγέθη unicast πακέτων που προσομοιώθηκαν

Όπως είδαμε στα αποτελέσματα των προσομοιώσεων με υψηλό φόρτο για το σύστημα, ο μέσος χρόνος παραμονής των πακέτων στην ουρά, αυξάνεται με την αύξηση του μεγέθους της. Το ίδιο συμπέρασμα προκύπτει και από την εικόνα 14.



Εικόνα 14: Μέση καθυστέρηση πακέτων για τα διαφορετικά ποσοστά broadcast

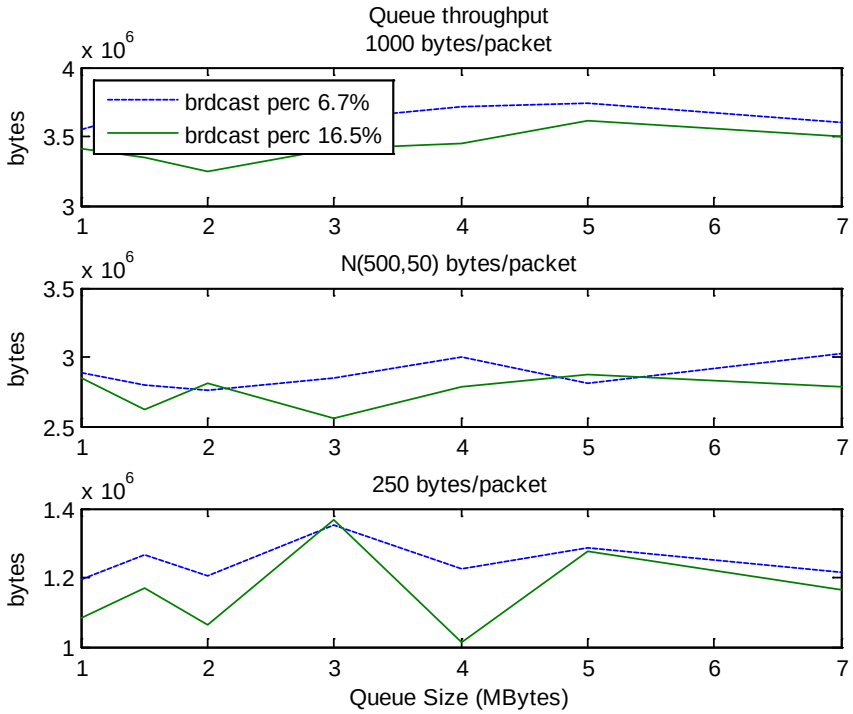
Ενδιαφέρον όμως, προκαλεί το γεγονός ότι η καμπύλη της καθυστέρησης, μετά από κάποιο μέγεθος ουράς σταματάει να έχει αύξουσα μορφή. Για παράδειγμα, παρατηρώντας την εικόνα 15, στην περίπτωση μεγάλων unicast πακέτων (1000 bytes), η καθυστέρηση σταματάει να αυξάνεται όταν η ουρά έχει μέγεθος μεγαλύτερο από 5Mb. Αυτό είναι το κρίσιμο μέγεθος ουράς, για το οποίο η ουρά σταματάει να απορρίπτει πακέτα για τον δεδομένο ρυθμό με τον οποίο τα δέχεται. Για τις συγκεκριμένες συνθήκες που προσομοιώθηκαν, κάθε ουρά με μέγεθος μικρότερο από 5Mb, λειτουργεί σε κατάσταση υπερχείλισης καθόλη την διάρκεια της προσομοίωσης. Όταν το μέγεθος γίνεται μεγαλύτερο από 5Mb, η ουρά σταματάει να απορρίπτει πακέτα, και πλέον ο μέσος χρόνος παραμονής του πακέτου στην ουρά, εξαρτάται μόνο από τον ρυθμό με τον οποίο δέχεται πακέτα και τον ρυθμό λειτουργία του μεταγωγέα. Αντίστοιχη παρατήρηση μπορεί να προκύψει και για τις ουρές που δέχονται μικρότερα πακέτα με μέση τιμή 500 bytes. Οι ανομοιομορφίες που παρουσιάζονται στη αντίστοιχη καμπύλη θεωρήθηκε ότι οφείλονται στον συνδυασμό των γεγονότων ότι και τα μεγέθη των πακέτων αλλά και ο ρυθμός άφιξής τους στην είσοδο της ουράς προκύπτουν τυχαία σε κάθε προσομοίωση. Για μικρά πακέτα (250 bytes), η καθυστέρηση είναι σταθερή, αφού η ουρές δεν λειτουργούν σε υπερχείλιση.



Εικόνα 15: Μέση καθυστέρηση πακέτων για τα διαφορετικά μεγέθη unicast

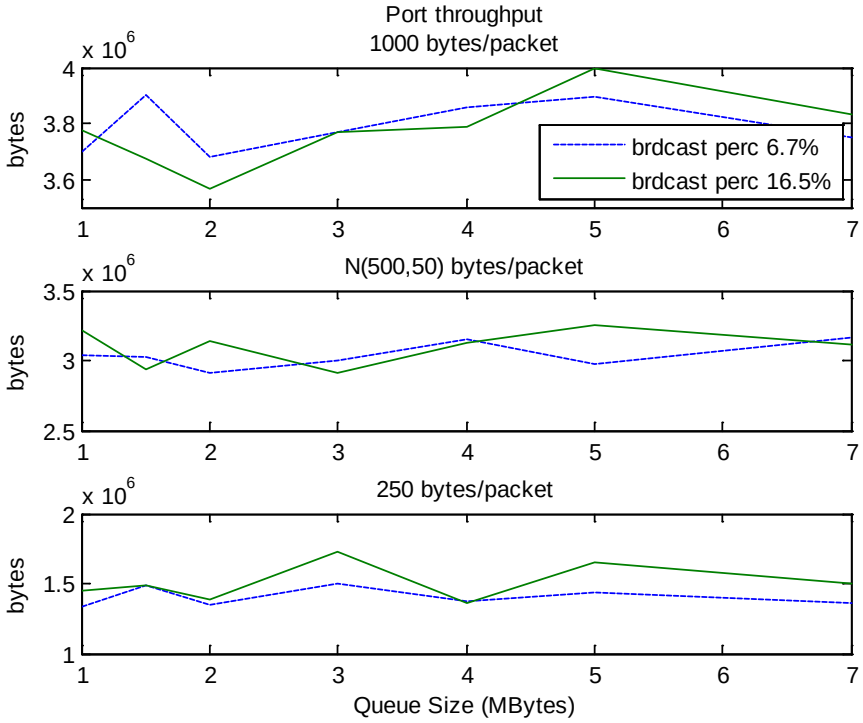
Όσον αφορά το throughput των ουρών του μεταγωγέα, όταν δεν λειτουργούν σε συνθήκες υπερχείλισης, είναι ανεξάρτητο από το μέγεθός τους, όπως μπορεί να γίνει αντιληπτό από την εικόνα 16. Το throughput είναι ανάλογο του ρυθμού με τον οποίο κάθε ουρά δέχεται πακέτα καθώς και ανάλογο του ρυθμού λειτουργίας του μεταγωγέα.

Οι μετρήσεις έχουν γίνει για ίδιο ρυθμό άφιξης πακέτων, διαφορετικού όμως μεγέθους ανά ουρά. Επομένως, ο ρυθμός με τον οποίο κάθε ουρά δέχεται δεδομένα διαφέρει. Άρα δεν έχει νόημα η επιμέρους σύγκριση του throughput για τα διαφορετικά μεγέθη πακέτων. Μπορούμε όμως να συγκρίνουμε την απόδοση των ουρών που δέχονται πακέτα ίδιου μεγέθους για τα δύο διαφορετικά ποσοστά broadcast πακέτων στο δίκτυο, που προσομοιώθηκαν. Όπως φαίνεται στην εικόνα 16, το throughput των ουρών είναι γενικά μικρότερο όταν υπάρχουν περισσότερα broadcast πακέτα στο δίκτυο. Αυτό φαίνεται εντονότερα στις ουρές που δέχονται μεγάλα πακέτα. Η μείωση του throughput των ουρών είναι αναμενόμενη, λόγω της αύξησης των περιόδων αδράνειάς τους, για την εξυπηρέτηση broadcast αποστολών.



Εικόνα 16: Throughput ουρών για τα διάφορα μεγέθη unicast πακέτων της προσομοίωσης ανά μέγεθος ουράς

Τέλος, το throughput των εξόδων του μεταγωγέα (εικόνα 17) φαίνεται να μην επηρεάζεται από την αλλαγή του ποσοστού broadcast πακέτων που καλείται να στείλει το σύστημα, αν παρατηρήσουμε ότι οι γραφικές παραστάσεις, ανά δύο για τα ίδια μεγέθη unicast πακέτων ανά έξοδο, δεν διαφέρουν σημαντικά.



Εικόνα 17: Throughput εξόδων για τα διάφορα μεγέθη unicast πακέτων της προσομοίωσης ανά μέγεθος ουράς

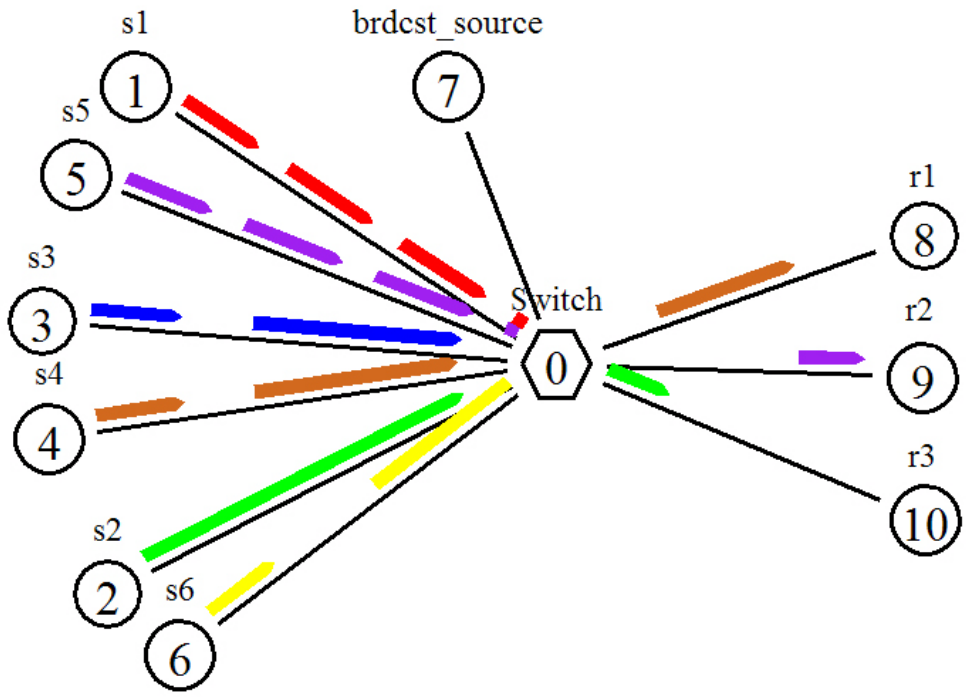
Με βάση τον μέσο ρυθμό παραγωγής δεδομένων από κάθε κόμβο-αποστολέα που υπολογίστηκε στην αρχή αυτής της παραγράφου, καθώς και την χρονική διάρκεια των προσομοιώσεων, μπορεί να υπολογιστεί το σύνολο δεδομένων που παρήγαγε κατά μέσο όρο κάθε κόμβος. Έτσι, ο κόμβος 1 (250bytes/pck) παρήγαγε κατά προσέγγιση 2,5Mbyte, ο κόμβος 2 (1000bytes/pck) 10Mbyte και ο κόμβος 3 (N(500,50)bytes/pck) 5Mbyte αντίστοιχα. Συγκρίνοντας αυτές τις τιμές με το throughput των εξόδων του μεταγωγέα στην παραπάνω εικόνα, παρατηρούμε ότι το throughput είναι ελαφρά μικρότερο από το σύνολο των αντίστοιχων δεδομένων που δέχεται ο μεταγωγέας σε κάθε είσοδο του. Αυτή η διαφορά οφείλεται στην απόρριψη unicast πακέτων, λόγω υπερχείλισης των ουρών των εξόδων, που προκαλείται λόγω της αναστολής εξυπηρέτησής τους με σκοπό την

εξυπηρέτηση των broadcast. Η μείωση αυτή φαίνεται να είναι εντονότερη για μεγάλα μεγέθη πακέτων και μικρά μεγέθη ουρών.

4.3 Υψηλός φόρτος εργασίας

Η τοπολογία της προσομοίωσης αποτελείται από έξι κόμβους-αποστολείς που στέλνουν unicast πακέτα και έναν τέταρτο που στέλνει broadcast πακέτα σε τρεις άλλους κόμβους-παραλήπτες. Κάθε ένας από τους κόμβους παραγωγής unicast πακέτων, ορίστηκε να παράγει πακέτα διαφορετικών μεγεθών. Η εικόνα 18 παρουσιάζει ένα στιγμιότυπο της προσομοίωσης της παραπάνω τοπολογίας, όπως παράγεται από το εργαλείο NAM. Οι κόμβοι 1-6 αποστέλλουν unicast πακέτα ενώ ο κόμβος 7 λειτουργεί σαν παραγωγός broadcast πακέτων.

Προς κάθε κόμβο παραλήπτη, αποστέλλουν πακέτα 2 εφαρμογές που εκτελούνται σε δύο διαφορετικούς κόμβους (applications) με ρυθμό παραγωγής δεδομένων 6000Kb/sec. Επομένως σε κάθε ουρά εξόδου καταφθάνουν δεδομένα με ρυθμό 12000Kb/sec. Ο ρυθμός λειτουργίας του μεταγωγέα έχει οριστεί στα 10Mbits/sec. Έτσι, οι ουρές υπερχειλίζουν αφού ο ρυθμός αποστολής από τις εξόδους υπολείπεται του ρυθμού παραλαβής δεδομένων. Η αύξηση του ποσοστού των broadcast πακέτων, θέτει την εξυπηρέτηση των ουρών προτεραιότητας κάθε πόρτας σε κατάσταση αναμονής όλο και περισσότερο, έτσι ώστε να εξυπηρετηθεί η broadcast ουρά. Αυτό οδηγεί σε αύξηση του ποσοστού unicast πακέτων που απορρίπτει κάθε ουρά καθώς και σε αύξηση της μέσης καθυστέρησης ανά πακέτο.

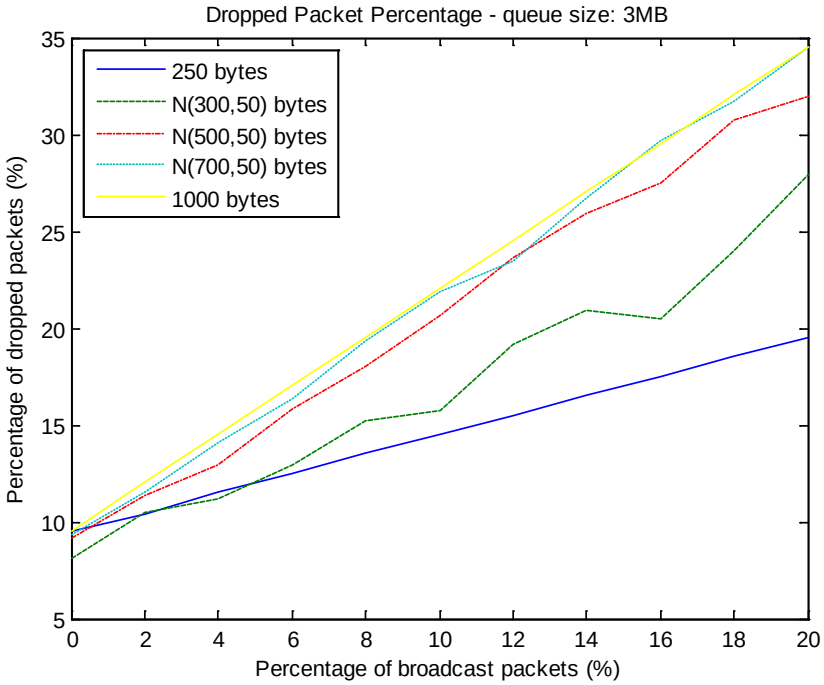


Εικόνα 18: Στιγμιότυπο προσομοίωσης λειτουργίας switch σε υψηλό φόρτο εργασίας

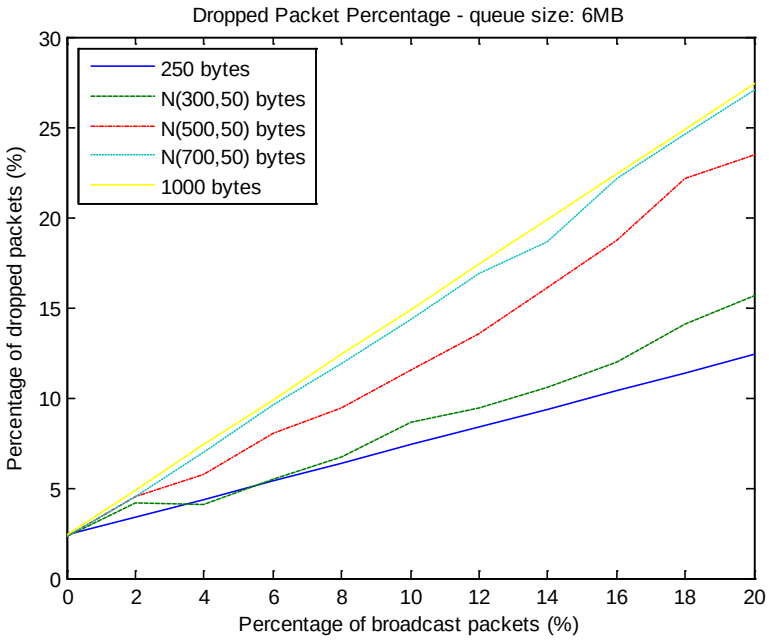
Οι μετρήσεις έγιναν για διάφορες περιπτώσεις μεγέθους πακέτων. Συγκεκριμένα έγιναν για ουρές που δέχονται πακέτα το μέγεθος των οποίων ακολουθεί κανονική κατανομή με μέση τιμή 300, 500 και 700 bytes και τυπική απόκλιση 50 bytes. Επίσης μελετήθηκε η απόδοση των ουρών για την περίπτωση που δέχονται μόνο πακέτα μικρού μεγέθους (250 bytes) ή μόνο μεγάλα πακέτα (1000 bytes). Οι κόμβοι 1 και 5 της εικόνας 18 αποστέλλουν μικρά πακέτα, οι κόμβοι 2 και 6 μεγάλα ενώ οι 3 και 4 λειτουργούν με βάση την κατανομή που περιγράφηκε.

Η λειτουργία του μεταγωγέα προσομοιώθηκε για μεγέθη ουράς 3MB και 6MB, ενώ οι μετρήσεις έγιναν για διαφορετικά ποσοστά broadcast αποστολών στο δίκτυο. Συγκεκριμένα, για ποσοστά broadcast πακέτων από 0% έως 20% με βήμα 2%.

Και στις δύο περιπτώσεις μεγέθους ουράς που προσομοιώθηκαν, παρατηρήθηκε ότι η αύξηση του ποσοστού των broadcast πακέτων που καταφθάνουν σε κάθε ουρά έχει σαν αποτέλεσμα μια γραμμική αύξηση του ποσοστού των πακέτων που απορρίπτονται από την ουρά λόγω υπερχείλισης, όπως διαπιστώνεται από τις εικόνες 19 και 20.



Εικόνα 19: Ποσοστό απορριφθέντων πακέτων ανά πόρτα (μέγεθος ουράς 3mb)



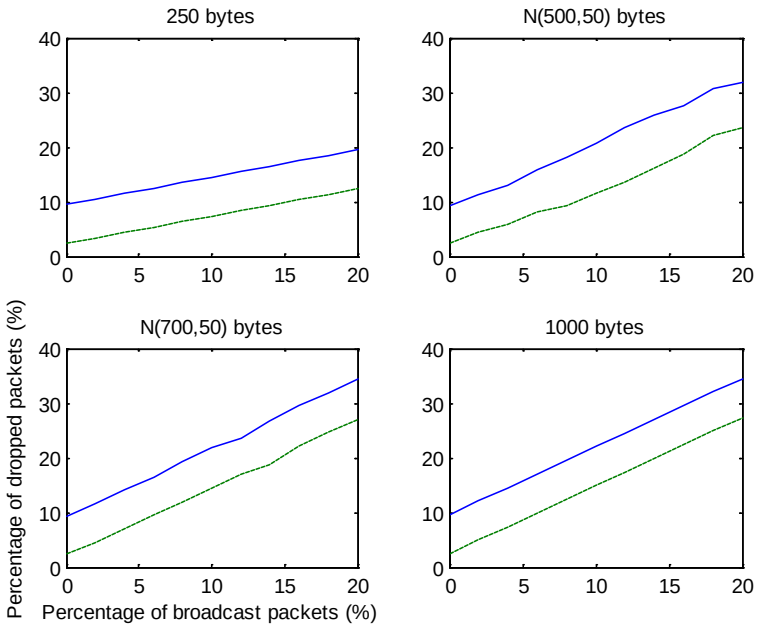
Εικόνα 20: Ποσοστό απορριφθέντων πακέτων ανά πόρτα (μέγεθος ουράς 6mb)

Η κλίση της καμπύλης επηρεάζεται από το μέγεθος των unicast πακέτων που λαμβάνει η ουρά. Όπως φαίνεται, όταν η ουρές λαμβάνουν μεγάλα πακέτα, η αύξηση του ποσοστού broadcast πακέτων, αυξάνει το ποσοστό απόρριψης με μεγαλύτερο ρυθμό.

Στις παραπάνω εικόνες παρατηρούμε ότι όταν το ποσοστό broadcast είναι μηδενικό, περίπου το 9% των εισερχόμενων πακέτων για ουρές

μεγέθους 3MB και το 2.5% για τις ουρές μεγέθους 6MB αντίστοιχα, απορρίπτεται. Αυτό συμβαίνει γιατί ο μεταγωγέας καλείται να στείλει unicast δεδομένα με ρυθμό μεγαλύτερο από τον ονομαστικό ρυθμό λειτουργίας του, επομένως, σε μικρό χρονικό διάστημα από την έναρξη της προσομοίωσης, οι ουρές υπερχειλίζουν, ακόμα και όταν δεν υπάρχουν αναστολές της εξυπηρέτησης τους για broadcasting.

Η ακόλουθη γραφική παράσταση απεικονίζει το ποσοστό απορριφθέντων πακέτων συγκρίνοντας ανά δύο τις καμπύλες που έχουν προκύψει από τις προσομοιώσεις με ουρές μεγέθους 3MB και 6MB.

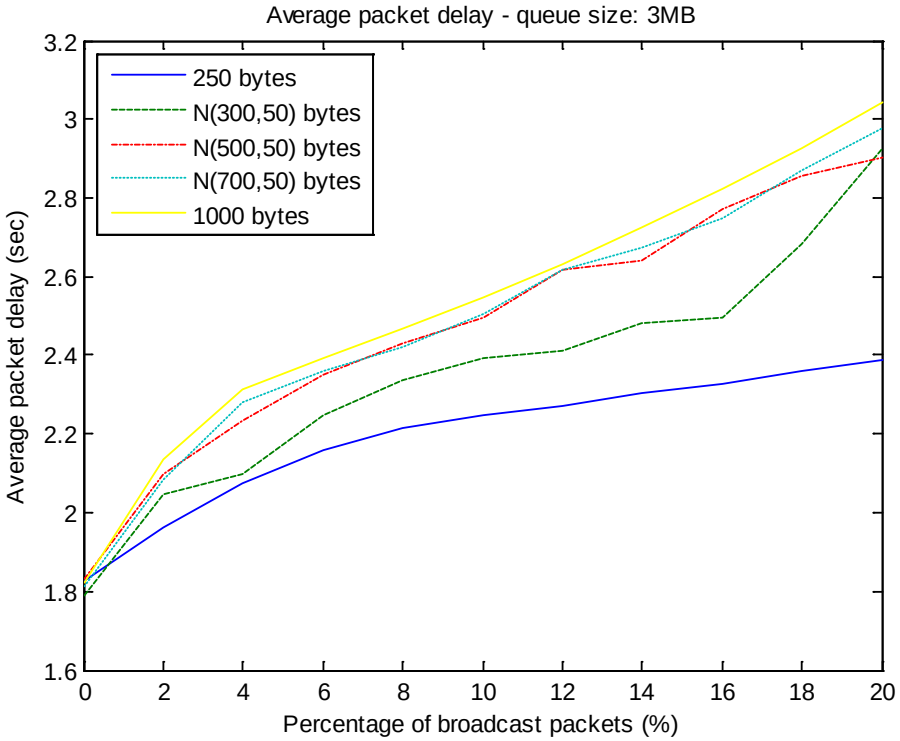


Εικόνα 21: Σύγκριση ποσοστού απορριφθέντων πακέτων για τα 2 μεγέθη ουρών που προσομοιώθηκαν

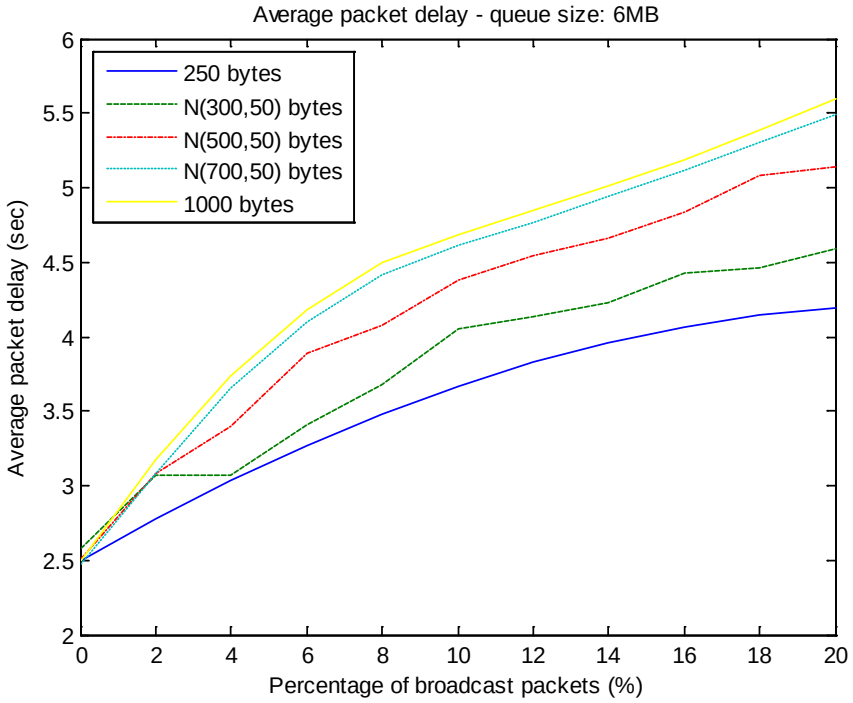
Συγκρίνοντας το ποσοστό απόρριψης πακέτων ανάμεσα στις 2 υλοποιήσεις, παρατηρούμε ότι οι καμπύλες για πακέτα ίδιου μεγέθους είναι παράλληλες. Για τον συγκεκριμένο ρυθμό με τον οποίο οι ουρές δέχονται δεδομένα από τους κόμβους-παραγωγούς, ο διπλασιασμός της διαθέσιμης μνήμης σε κάθε ουρά μειώνει το ποσοστό απόρριψης κατά περίπου 7.2 ποσοστιαίες μονάδες. Όμως η αύξηση του μεγέθους της ουράς έχει και αρνητικά αποτελέσματα όπως θα δούμε στην συνέχεια.

Οι εικόνες 22 και 23 απεικονίζουν την μέση καθυστέρηση εξυπηρέτησης πακέτων ανά έξοδο, σε σχέση με το ποσοστό broadcast δεδομένων που δέχεται ο μεταγωγέας. Παρατηρούμε όπως ήταν αναμενόμενο, ότι η καθυστέρηση αυξάνεται με την αύξηση του ποσοστού

broadcast. Όμως φαίνεται ότι τείνει να συγκλίνει σε κάποια τιμή, όταν το ποσοστό broadcast πακέτων έχει υπερβεί το 20% του συνόλου των unicast πακέτων.

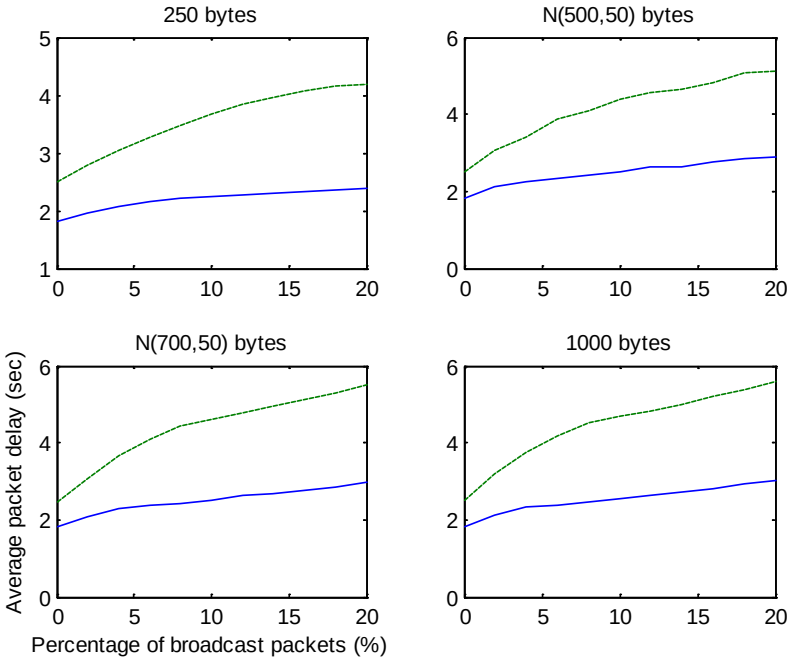


Εικόνα 22: Μέση καθυστέρηση πακέτου για ουρές μεγέθους 3Mb



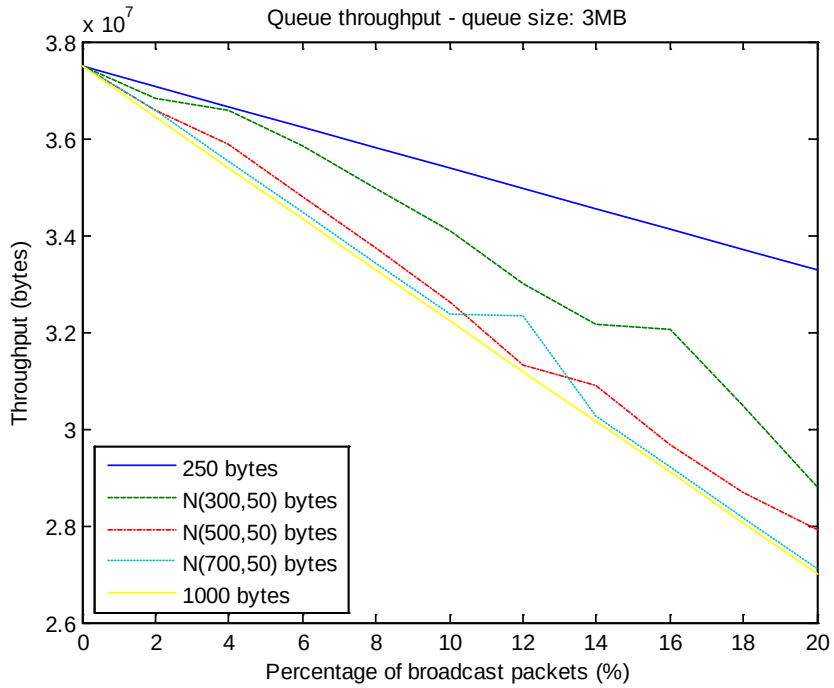
Εικόνα 23: Μέση καθυστέρηση πακέτου για ουρές μεγέθους 6Mb

Σε αυτήν την περίπτωση παρατηρούμε ότι ο διπλασιασμός του μεγέθους της ουράς, έχει αρνητική επίπτωση στο σύστημα, καθώς οδηγεί σε μια αντίστοιχη αύξηση του μέσου χρόνου που απαιτείται από την παραλαβή έως την αποστολή ενός πακέτου από τον μεταγωγέα, όπως γίνεται αντιληπτό από την εικόνα 24 .

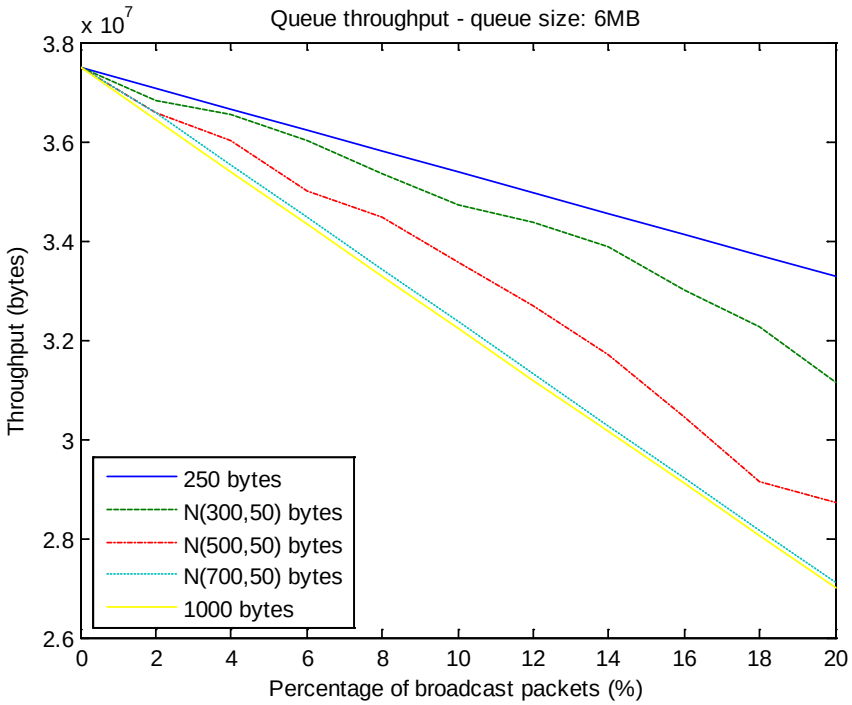


Εικόνα 24: Σύγκριση μέσων καθυστερήσεων ανά μέγεθος ουράς

Όσον αφορά το throughput των ουρών, δηλαδή την ποσότητα unicast πληροφορίας που πέρασε από κάθε ουρά, παρατηρούμε βάσει των εικόνων 25 και 26, ότι μειώνεται όσο αυξάνεται το μέγεθος των πακέτων που δέχεται η κάθε ουρά, για ίδια ποσοστά broadcast πακέτων. Επίσης δεν επηρεάζεται από το μέγεθος των ουρών αφού λειτουργούν σε κατάσταση υπερχείλισης καθόλη την διάρκεια της προσομοίωσης.

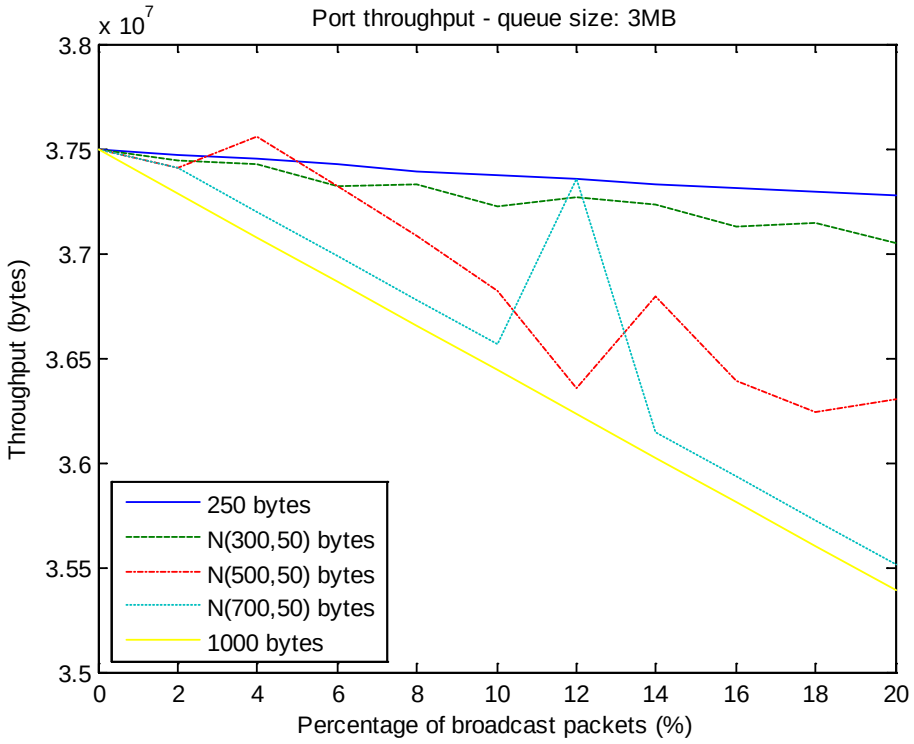


Εικόνα 25: Throughput ουρών μεγέθους 3Mb

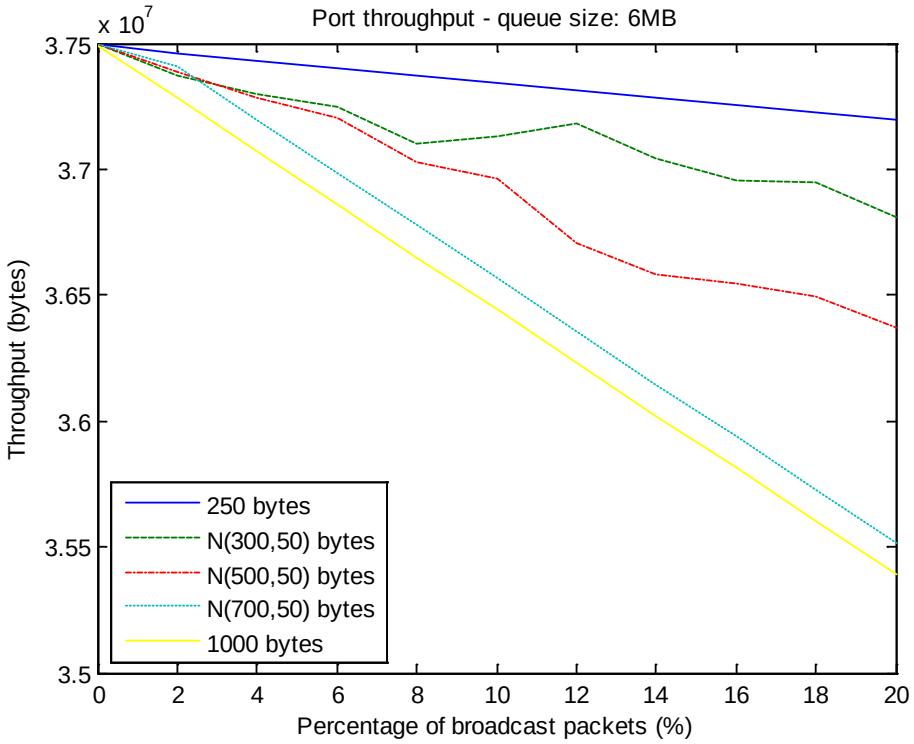


Εικόνα 26: Throughput ουρών μεγέθους 6Mb

Τέλος, παρατηρούμε ότι όταν η έξοδος καλείται να στέλνει δεδομένα με ρυθμό μεγαλύτερο του ονομαστικού ρυθμού λειτουργίας της, και το συνολικό throughput της εξόδου δεν διαφέρει για διαφορετικά μεγέθη ουράς, όπως φαίνεται στις εικόνες 27 και 28.



Εικόνα 27: Throughput εξόδων για ουρές μεγέθους 3Mb



Εικόνα 28: Throughput εξόδων για ουρές μεγέθους 6Mb

Η απόδοση (throughput) όμως της κάθε εξόδου του μεταγωγέα επηρεάζεται έντονα από το ποσοστό broadcast πακέτων που λαμβάνει.

Όταν ο μεταγωγέας δεν δέχεται καθόλου broadcast πακέτα, η απόδοση είναι η ίδια ανεξαρτήτως του μεγέθους των unicast πακέτων που λαμβάνει.

Για μικρά unicast πακέτα (250 bytes), η απόδοση των εξόδων δεν επηρεάζεται σημαντικά από την αύξηση του ποσοστού των broadcast. Κάθε φορά που ο μεταγωγέας λαμβάνει ένα broadcast πακέτο (με μέγεθος 500 bytes), ακυρώνει την τρέχουσα αποστολή των (μικρότερων) unicast πακέτων έτσι ώστε να εξυπηρετηθεί η broadcast ουρά. Ο χρόνος που έχει χαθεί από την έναρξη της αποστολής του unicast μέχρι την ακύρωσή της είναι μικρός, επομένως η πόρτα παραμένει ανενεργή για λίγο. Έτσι, για broadcast πακέτα μεγαλύτερα από τα unicast, η απόδοση παραμένει σχεδόν σταθερή, παρά τις ακυρώσεις αποστολών unicast πακέτων.

Αντίθετα, όταν το σύστημα λειτουργεί για μεγάλα unicast πακέτα (1000 bytes), η αύξηση του ποσοστού broadcast αποστολών μειώνει την απόδοση του συστήματος. Αυτό οφείλεται στον μεγάλο χρόνο αδράνειας κάθε εξόδου, λόγω των ακυρώσεων αποστολών μεγάλων πακέτων, με σκοπό την εξυπηρέτηση μικρότερων. Για ενδιάμεσα μεγέθη unicast πακέτων, παρατηρείται μια μικρή πτώση της απόδοσης, η οποία εξαρτάται από το μέγεθος των πακέτων.

Τέλος, παρατηρούμε ότι όταν το σύστημα δουλεύει σε υψηλούς ρυθμούς, το throughput των εξόδων δεν επηρεάζεται από το μέγεθος των ουρών, για συγκεκριμένο χρονικό διάστημα προσομοιώσεων.

Οι παραπάνω παρατηρήσεις έρχονται σε αντίθεση με την παρατήρηση ότι σε συνθήκες κανονικού φόρτου, για ίδια μεγέθη πακέτων, το throughput των εξόδων δεν επηρεάζεται σημαντικά από την μεταβολή του ποσοστού broadcast αποστολών (εικόνα 17).

5 Επίλογος

5.1 Μελλοντικές επεκτάσεις

Όπως κάθε σύστημα που σχεδιάζεται για πρώτη φορά, έτσι και ο μεταγωγέας που αναπτύχθηκε, μπορεί μελλοντικά να επιδεχθεί παρεμβάσεις με σκοπό την βελτιστοποίηση της λειτουργίας του ή την διεξαγωγή πιο εξειδικευμένων προσομοιώσεων.

Όσον αφορά την λειτουργία του μεταγωγέα, μια σημαντική βελτίωση είναι η μετατροπή των στατικών ουρών πακέτων σε δυναμικές. Θα μπορούσε δηλαδή να υπάρχει μια κεντρική μνήμη, η οποία θα κατανέμεται στις διαφορετικές εξόδους ανάλογα με τον φόρτο εργασίας κάθε εξόδου. Για να γίνει αυτό πρέπει να αναπτυχθεί μια λογική ελέγχου του καταμερισμού της μνήμης καθώς και να οριστούν οι κανόνες που θα διέπουν αυτό τον καταμερισμό, με σκοπό την διατήρηση της ισονομίας για κάθε έξοδο.

Επιπλέον, θα μπορούσε να τροποποιηθεί η μέθοδος διαχείρισης των πακέτων, ίδιας IP προτεραιότητας, που πρέπει να αποσταλούν από μια πόρτα. Η εξυπηρέτηση με βάση την χρονική προτεραιότητα των παραληφθέντων πακέτων που έχει υλοποιηθεί ως τώρα, θα μπορούσε να συνδυαστεί με μια μέθοδο ελέγχου του φόρτου εργασίας ανά πόρτα εισόδου. Έτσι, ο μεταγωγέας θα μπορούσε να δίνει προτεραιότητα σε εισερχόμενα πακέτα από μια είσοδο συνδεδεμένη με έναν κόμβο που αποστέλλει δεδομένα με υψηλότερο ρυθμό σε σχέση με άλλους κόμβους συνδεδεμένους στις υπόλοιπες εισόδους του συστήματος.

Μια δεύτερη πιθανή βελτίωση του συστήματος θα ήταν η αλλαγή του τρόπου διαχείρισης πακέτων διαφορετικών IP προτεραιοτήτων. Στην παρούσα έκδοση, όσο υπάρχουν πακέτα υψηλότερης προτεραιότητας σε μια ουρά, εξυπηρετούνται, ενώ οι υπόλοιπες ουρές παραμένουν ανενεργές, αποδεχόμενες εισερχόμενα πακέτα μέχρι να υπερχειλίσουν. Μελλοντικά θα

μπορούσε να αναπτυχθεί ένας αλγόριθμος που να κατανέμει ένα μικρό χρονικό ποσοστό της λειτουργίας του μεταγωγέα σε εξυπηρέτηση ουρών χαμηλότερης προτεραιότητας, ενώ υπάρχουν πακέτα υψηλότερης προτεραιότητας προς αποστολή.

Όσον αφορά τις προσομοιώσεις, μπορούν να αναπτυχθούν νέα σενάρια χρήσης του μεταγωγέα. Αρχικά μπορεί να ελεγχθεί η λειτουργία του, όταν κάθε πόρτα εξόδου καλείται να αποστείλει ταυτόχρονα πακέτα διαφορετικών IP προτεραιοτήτων. Επιπλέον μπορούν να γίνουν προσομοιώσεις που να περιλαμβάνουν περισσότερους από έναν μεταγωγείς στο ίδιο δίκτυο, κάτι το οποίο υποστηρίζεται στην παρούσα έκδοση, αλλά δεν προσομοιώθηκε διεξοδικά.

Κατά της διαδικασία της αξιολόγησης του προσομοιούμενου συστήματος, δεν χρησιμοποιήθηκαν δεδομένα αναφοράς για την επιβεβαίωση της λειτουργίας του. Αυτό έγινε γιατί ο τρόπος δρομολόγησης των πακέτων εντός του κόμβου μεταγωγέα (διευθυνσιοδότηση εξόδων με βάση το id του συνδεδεμένου σε αυτές κόμβου – παράγραφος 3.1.1) σε συνδυασμό με την μέθοδο αποστολής πακέτων του συστήματος NS2 (κλήση συναρτήσεων παραλαβής από τον παραλήπτη – παράγραφος 2.7), διασφάλιζαν την ορθή κίνηση των πακέτων. Ωστόσο, θα μπορούσαν μελλοντικά να δημιουργηθούν δεδομένα αναφοράς σε σενάρια χρήσης του συστήματος κατά τα οποία τουλάχιστον δύο αποστολείς θα στέλνουν πακέτα διαφορετικών IP προτεραιοτήτων σε τουλάχιστον 2 παραλήπτες. Προσομοιώνοντας το σύστημα με τις συγκεκριμένες εισόδους και ενεργοποιώντας την λειτουργία καταγραφής trace αρχείων του NS2, μέσω της σύγκρισης θεωρητικών και πειραματικών αποτελεσμάτων, θα μπορούσε να επιβεβαιωθεί η χρονικά ορθή παράδοση των πακέτων.

Τέλος μπορούν να γίνουν προσομοιώσεις λειτουργίας του μεταγωγέα σε ένα δίκτυο στο οποίο θα επιδρούν και εξωγενείς παράγοντες, όπως καθυστερήσεις διάδοσης της πληροφορίας κατά μήκος των συνδέσεων των κόμβων ή απώλειες πακέτων λόγο θορύβου.

6 Βιβλιογραφία

- Ερμής Ιωάννης, Σχεδιασμός και υλοποίηση ενός 8x8 Ethernet switch, μεταπτυχιακή διατριβή, Πολυτεχνείο Κρήτης, τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών, Χανιά, Απρίλιος 2007.
- James F. Kurose, Keith W. Ross, Δικτύωση Υπολογιστών - Προσέγγιση από Πάνω προς τα Κάτω με Έμφαση στο Διαδίκτυο, δεύτερη έκδοση, εκδόσεις Μ. Γκιούρδας, Αθήνα, 2004
- Jean Walrand, Δίκτυα Επικοινωνιών, δεύτερη έκδοση, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Αθήνα, 2003
- Hongyun Zheng, Yongxiang Zhao and Changjia Chen, Design and Implementation of Switches in Network Simulator (ns2), Proceedings of the First International Conference on Innovative Computing, Information and Control, 2006
- Isaac Keslassy and Josef Hai Kanizo, ABC: The All-Buffered Crossbar Switch
- The *ns* Manual (formerly *ns* Notes and Documentation) - A Collaboration between researchers at UC Berkeley, LBL, USC/ISI and Xerox PARC, January 2009.
- http://nsnam.isi.edu/nsnam/
- http://wiki.tcl.tk/
- http://www.cygwin.com
- http://www.ieee.org
- http://www.wikipedia.org
- Αρχεία βοήθειας Matlab

Παράρτημα

Εγκατάσταση

Για την εγκατάσταση και προσομοίωση του συστήματος πρέπει να ακολουθηθούν τα παρακάτω βήματα:

1. Εγκατάσταση του NS2, σύμφωνα με τις οδηγίες που περιέχονται στο αρχείο *Manual for installing NS2 under Window XP.pdf*:
2. Αντιγραφή του φακέλου “Switch” εντός του φακέλου ~/ns-allinone-2.34/ns-2.34
3. Αντικατάσταση των αρχείων classifier.h και classifier.cc που βρίσκονται στον φάκελο ~/ns-allinone-2.34/ns-2.34classifier, με τα αντίστοιχα που περιέχονται στο cd της εργασίας.
4. Αντικατάσταση των αρχείων ping.h και ping.cc που βρίσκονται στον φάκελο ~/ns-allinone-2.34/ns-2.34/apps, με τα αντίστοιχα που περιέχονται στο cd της εργασίας.
5. Αντικατάσταση των αρχείων ns-node.tcl και ns-default.tcl που βρίσκονται στον φάκελο ~/ns-allinone-2.34/ns-2.34/tcl/lib, με τα αντίστοιχα που περιέχονται στο cd της εργασίας.
6. Τροποποίηση του αρχείου “*Makefile*” που βρίσκεται στον φάκελο ~/ns-allinone-2.34/ns-2.34:
    - Προσθήκη της γραμμής
      -I./Switch
      στο πεδίο “INCLUDES =” το οποίο δηλώνει τους φακέλους που περιέχουν τα πηγαία αρχεία για του συστήματος NS2
    - Προσθήκη της γραμμής
      ```
      Switch/sw_node.o Switch/sw_queue.o
      Switch/sw_timer.o \
      ```
      στο πεδίο “OBJ_CC =” το οποίο δηλώνει τους φακέλους που περιέχουν τα αντικείμενα αρχεία (.o) του συστήματος.

7. Ανοίγουμε το Cygwin, και κατευθυνόμαστε στον φάκελο ~/ns-allinone-2.34/s-2.34
8. Δίνουμε στο σύστημα την εντολή “make”. Η διαδικασία πρέπει να ολοκληρωθεί χωρίς λάθη