ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Υλοποίηση και Αξιολόγηση Αναμεταδότη Πακέτων Δικτύου σε Συνόδους Αναπαραγωγής Μουσικής"

Δημήτριος Παππάς

ΜΜ4130019

ΑΘΗΝΑ, ΙΟΥΝΙΟΣ 2015

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση και Αξιολόγηση Αναμεταδότη Πακέτων Δικτύου σε Συνόδους Αναπαραγωγής Μουσικής

Δημήτριος Παππάς

ΜΜ4130019

Επιβλέπων Καθηγητής: Γ. Ξυλωμένος

ΑΘΗΝΑ, ΙΟΥΝΙΟΣ 2015

1 Πίνακας Περιεχομένων

2 Πίνακας Εικόνων

3 Εισαγωγή

Τις τελευταίες δεκαετίες, με την ανάπτυξη της τεχνολογίας, ιδιαίτερο ενδιαφέρον έχει προσελκύσει η προσπάθεια αναπαραγωγής και δημιουργίας μουσικής από δημιουργούς σε απομακρυσμένες, μεταξύ τους, τοποθεσίες *(Network Music Performance - NMP)*. Τα πρώτα βήματα σε αυτόν τον τομέα έγιναν στα τέλη της δεκαετίας του '70 όταν ένα συγκρότημα, με την ονομασία *The League of Automatic Music Composers*, άρχισαν να χρησιμοποιούν διασυνδεδεμένους υπολογιστές, οι οποίοι αντάλλασαν μεταξύ τους μηνύματα, για να παράγουν μελωδία.

Με την πάροδο των ετών έγινε προσπάθεια η ανταλλασσόμενη πληροφορία να μετατραπεί από απλά μηνύματα σε μουσικά σήματα. Κάτι τέτοιο ήταν πολύ δύσκολο καθώς η ταχύτητα μεταφοράς δεδομένων ήταν πολύ μικρή ενώ ο όγκος της πληροφορίας ήχου πολύ μεγάλος. Έτσι μόνο μετά την ανάπτυξη της κύριας υποδομής του Διαδικτύου έγινε κάτι τέτοιο δυνατό, με το *SoundWire project* [1]. Το project αυτό χρησιμοποίησε το διαθέσιμο εύρος ζώνης για να πραγματοποιήσει αμφίδρομη μετάδοση ασυμπίεστου ήχου.

Αν και επιτεύχθηκε η μετάδοση ήχου μέσω δικτύου, η χρήση ασυμπίεστης μορφής δεδομένων απαιτεί πολύ μεγάλο εύρος ζώνης κάτι το οποίο δεν το κάνει ιδιαίτερα προσιτό. Ειδικά αν αναλογιστούμε τις οικιακές συνδέσεις, κάτι τέτοιο γίνεται απαγορευτικό για τον μέσο χρήστη καθώς η δημιουργία συνόδων, ακόμα και με 3 ή 4 άτομα, θα απαιτούσε τεράστιο εύρος ζώνης αφού για ελάχιστα δευτερόλεπτα απαιτούνται εκατοντάδες MB, έως GB, δεδομένων. Για τον λόγο αυτό ένα μεγάλο ερευνητικό κομμάτι αποτελεί η εξεύρεση τρόπων βελτίωσης των τεχνολογιών του NMP ώστε να γίνει ευρέως διαθέσιμο.

Η δημιουργία μιας τέτοιας συνόδου είναι ιδιαίτερα απαιτητική καθώς για την πραγματοποίησή της απαιτείται η ελαχιστοποίηση όλων των καθυστερήσεων, κάτω από τα 30-50 msec που αποτελούν την ελάχιστη διαφορά χρόνου στην οποία μπορεί ο άνθρωπος να αντιληφθεί διαφορετικά ηχητικά σήματα [2].

Σε ένα τέτοιο σύστημα συνόδων NMP, η καθυστέρηση εισάγεται από διάφορες διαδικασίες που είναι απαραίτητες για την πραγματοποίηση της, όπως:

- Λήψη δεδομένων (εικόνα και/ή ήχος)
- Συμπίεση δεδομένων
- Αποστολή σε κεντρικό εξυπηρετητή
- Αναμετάδοση στους αποδέκτες
- Αποσυμπίεση δεδομένων
- Αναπαραγωγή εικόνας και ήχου

Κάθε μία από τις παραπάνω διαδικασίες αποτελεί και ένα ξεχωριστό τομέα έρευνας.

Στην παρούσα εργασία θα ασχοληθούμε με την καθυστέρηση αναμετάδοσης των δεδομένων από τους κεντρικούς εξυπηρετητές. Θα γίνει η υλοποίηση και αξιολόγηση ενός τέτοιου συστήματος και θα παρουσιαστούν εναλλακτικές μέθοδοι για την βελτίωση της απόδοσής του.

4 Αναμετάδοση μέσων

Συστήματα συνόδων ήχου και εικόνας αποτελούνται συνήθως από έναν εξυπηρετητή, γνωστός ως *Selective Forwarding Unit* (*SFU*) [3] ο οποίος λειτουργεί ως αναμεταδότης πακέτων δικτύου. Κάθε συμμετέχον στην σύνοδο δημιουργεί ροές δεδομένων προς τον κεντρικό εξυπηρετητή - SFU - ο οποίος αναλαμβάνει τον χειρισμό και την ανακατεύθυνσή τους σε όλους τους αποδέκτες με την ελάχιστη δυνατή επεξεργασία και καθυστέρηση.

Στην κλασική τους μορφή, τα συστήματα αυτά προσπαθούν πριν από την αναμετάδοση να συγχρονίσουν και να συνενώσουν όλες τις ροές σε μία. Κάτι τέτοιο προκαλεί μεγάλες καθυστερήσεις, καθώς η διαδικασία συγχρονισμού των ροών μπορεί να είναι ιδιαίτερα αργή, ειδικά σε συνθήκες φόρτου του δικτύου. Επίσης η συνένωση τους απαιτεί επεξεργασία των δεδομένων των ροών, κάτι το οποίο κάνει τα συστήματα αυτά ιδιαίτερα απαιτητικά στην χρήση πόρων συστήματος.

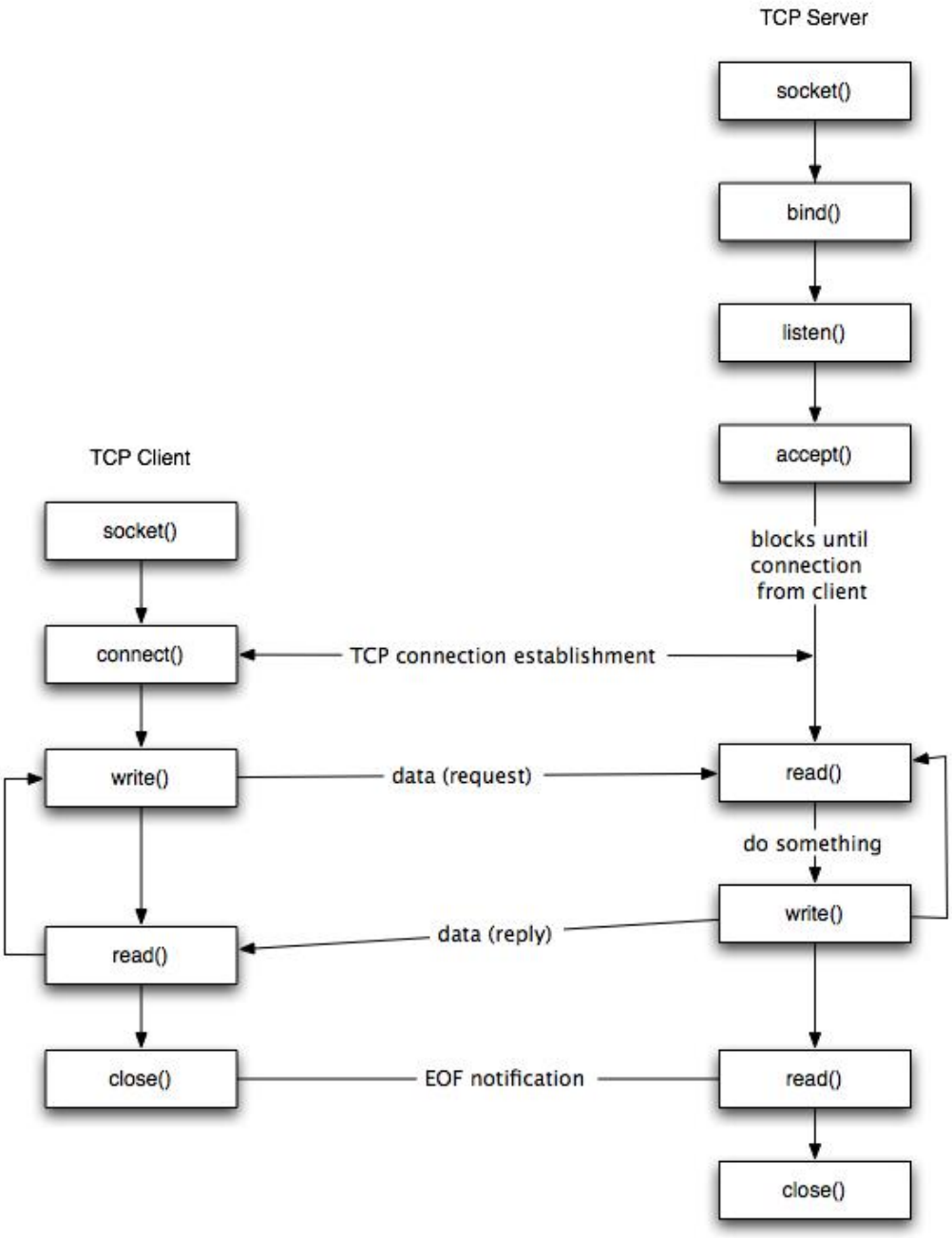
Τα σύγχρονα συστήματα SFU αναπαράγουν και αποστέλλουν τις ροές χωρίς να τις συγχρονίσουν και χωρίς να κάνουν κάποια άλλη επεξεργασία. Με τον τρόπο αυτό μειώνουν την καθυστέρηση, λόγω επεξεργασίας, αλλά απαιτούν μεγαλύτερο ρυθμό εξερχόμενων πακέτων, καθώς στέλνουν πολλά μικρά πακέτα έναντι λίγων μεγάλων του κλασσικού SFU. Έτσι χρησιμοποιούν έντονα το I/O του δικτύου κάτι το οποίο τα καθιστά υποψήφιο πεδίο δοκιμών για την εξεύρεση τρόπων μείωσής του απαιτούμενου χρόνου I/O και αύξησης της απόδοσής τους.

4.1 POSIX Sockets

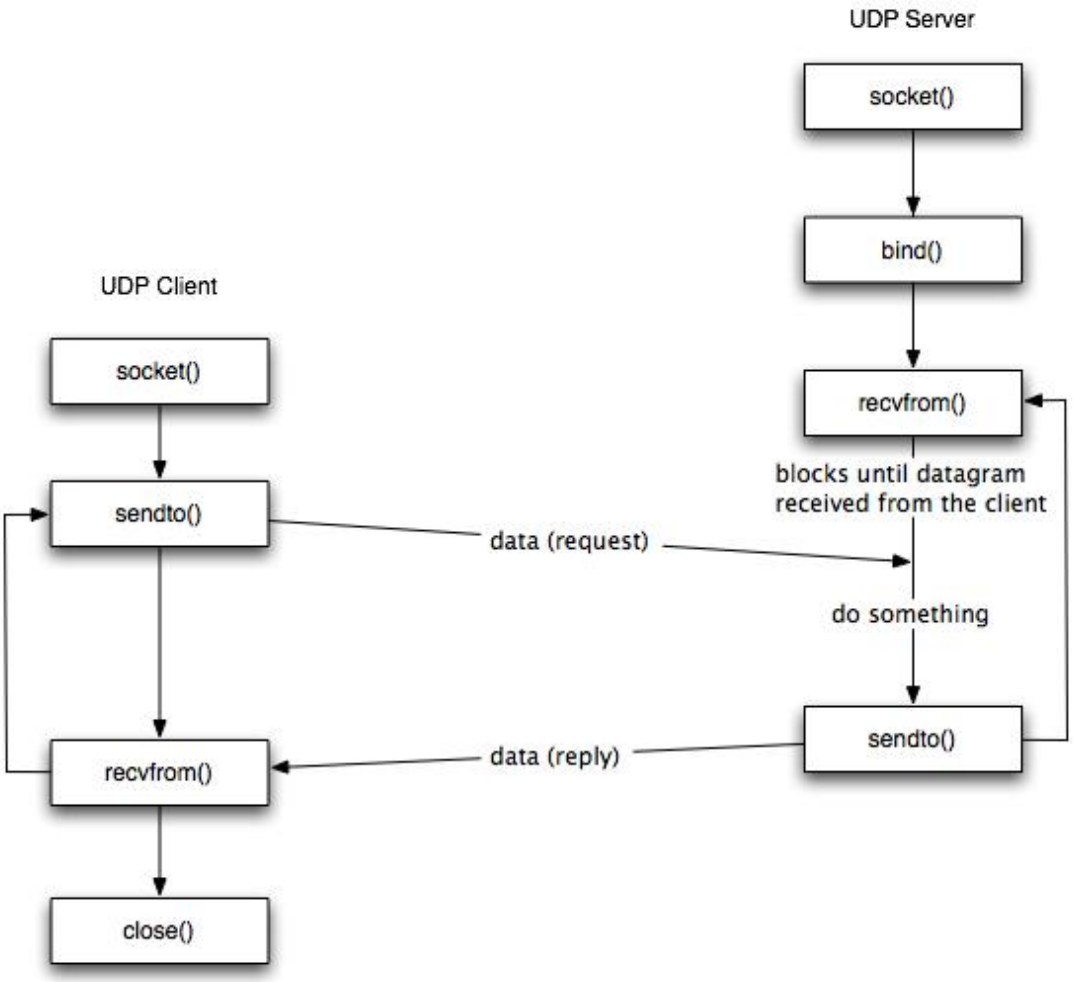
Στα σύγχρονα λειτουργικά συστήματα, που ακολουθούν την POSIX [4] αρχιτεκτονική, η διεπαφή socket είναι ο κύριος μηχανισμός μεταφοράς πακέτων μεταξύ των εφαρμογών και του πυρήνα του λειτουργικού για μεταφορά από και προς το δίκτυο. Η βασική αρχή λειτουργίας είναι η δημιουργία ενός socket(), από την εφαρμογή, μέσω του οποίου ενημερώνεται ο πυρήνας για τα στοιχεία των πακέτων

που θέλει να λάβει ή στείλει (πρωτόκολλο δικτύου, τύπος επικοινωνίας, πρωτόκολλο μεταφοράς).

Στην Εικόνα 4.1 φαίνεται ένα διάγραμμα επικοινωνίας του πρωτοκόλλου TCP με χρήση sockets και οι αντίστοιχες μέθοδοι που πρέπει να καλέσει η εφαρμογή ώστε να πραγματοποιηθεί η επικοινωνία, ενώ στην Εικόνα 4.2 το αντίστοιχο διάγραμμα του UDP.



Εικόνα 4.1 Διάγραμμα ροής επικοινωνίας TCP socket



Εικόνα 4.2 Διάγραμμα ροής επικοινωνίας UDP Socket

Κάθε συνάρτηση των ανωτέρω διαγραμμάτων αποτελεί και μία κλήση συστήματος (*system call*) της εφαρμογής προς τον πυρήνα για την διαδικασία την οποία θέλει να επιτελέσει. Οι περισσότερες από τις λειτουργίες αυτές πραγματοποιούνται πολύ λίγες φορές στην διάρκεια λειτουργίας της εφαρμογή. Εξαίρεση αποτελούν οι λειτουργίες αποστολής και λήψης δεδομένων οι οποίες καταλαμβάνουν το μεγαλύτερο μέρος του απαιτούμενου I/O.

Σε περίπτωση αποστολής δεδομένων ο πυρήνας (*kernel*) του λειτουργικού αναλαμβάνει μία σειρά από διαδικασίες:

- Λήψη του system call από την εφαρμογή
- Δέσμευση χώρου στην αποκλειστική μνήμη του πυρήνα

- Μεταφορά του payload από την εφαρμογή στον δεσμευμένο χώρο
- Μεταφορά του payload στη στοίβα δικτύου (*network stack*)
- Επεξεργασία και συμπλήρωση του πακέτου με headers-trailers
- Αποστολή του πακέτου μέσω της κάρτας δικτύου

Σε περίπτωση λήψης γίνεται η αντίστροφη διαδικασία για την μεταφορά του payload στην εφαρμογή.

Στην διαδικασία αυτή παρατηρούμε ότι τα πακέτα διαχειρίζονται από την εφαρμογή σε επίπεδο payload, κάτι που δημιουργεί συνεχή προσθήκη και αφαίρεση κεφαλίδων (Ethernet, IP, TCP/UDP). Ακόμα απαιτείται η συνεχής δημιουργία system calls και η διαρκής δέσμευση-αποδέσμευση μνήμης για την μεταφορά των πακέτων ανάμεσα στον πυρήνα και την εφαρμογή. Τέλος σε περίπτωση αποστολής του ίδιου πακέτου σε πολλούς αποδέκτες απαραίτητη είναι η επανάληψη όλης της διαδικασίας για κάθε έναν καθώς δεν υπάρχει δυνατότητα μαζικής αποστολής του ίδιου μηνύματος σε πολλούς αποδέκτες.

Όλες αυτές οι λειτουργίες εκτός από την καθυστέρηση που εισάγουν δημιουργούν επιπλέον φόρτο στο σύστημα με αποτέλεσμα σε συνθήκες αυξημένης κίνησης να φτάνουν τον υπολογιστή πιο εύκολα στα όρια λειτουργίας του, κάτι το οποίο δημιουργεί πολύ μεγάλες καθυστερήσεις.

Για τις περισσότερες εφαρμογές οι καθυστερήσεις από τις διαδικασίες αυτές δεν έχουν ιδιαίτερη επίπτωση και η διεπαφή socket παρέχει έναν εύκολο τρόπο προγραμματισμού με δυνατότητα φορητότητας σε διαφορετικά συστήματα (Linux, Unix, BSD, κτλ). Σε κάποιες περιπτώσεις όμως, όπου τα εισερχόμενα πακέτα φτάνουν με πολύ υψηλό ρυθμό και απαιτείται ελάχιστη ή καθόλου επεξεργασία του payload, τα socket μπορούν να γίνουν σημείο συμφόρησης και να μειώσουν αρκετά την απόδοση του συστήματος.

Για την επίλυση των προβλημάτων αυτών είναι δυνατή η δημιουργία αρθρωμάτων (*modules*) στον πυρήνα που επιτελούν τις ίδιες λειτουργίες με τις εφαρμογές σε επίπεδο χρήστη. Ο προγραμματισμός εντός πυρήνα, παρόλο που δεν

διαφέρει πολύ από τον προγραμματισμό μιας απλής εφαρμογής σε επίπεδο χρήστη, παρουσιάζει αρκετές δυσκολίες (πχ debugging) αλλά και κινδύνους για το ίδιο το λειτουργικό καθώς πολύ μικρά προβλήματα μπορούν να προκαλέσουν εύκολα την κατάρρευσή του.

Έχουν προταθεί διάφορες μέθοδοι για την αντικατάσταση των *POSIX Sockets* στα συστήματα αυτά με εναλλακτικά υψηλότερης απόδοσης [5] όπως το *Click modular router* [6], το *NetFPGA* [7], το N*etMap* [8]. Στην παρούσα εργασία γίνεται η υλοποίηση ενός συστήματος SFU με χρήση του Click modular router και η αξιολόγησή του σε σύγκριση με ένα σύγχρονο σύστημα SFU που χρησιμοποιεί POSIX Sockets.

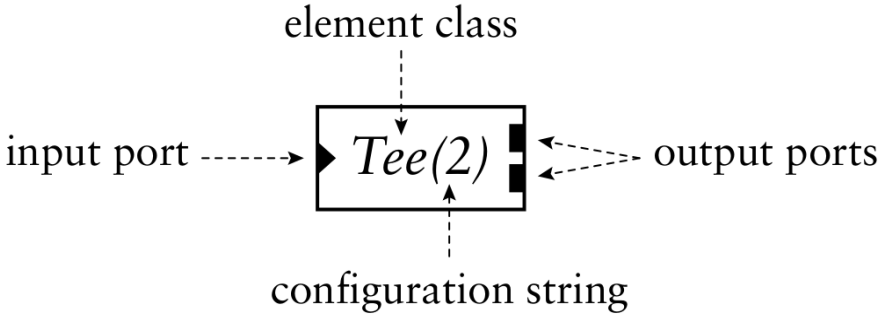
4.2 Click Modular Router

Για την αντιμετώπιση των δυσκολιών δημιουργίας kernel modules είναι δυνατή η χρήση του Click Modular Router (ή Click), το οποίο δίνει την δυνατότητα δημιουργίας δικτυακών εφαρμογών με χρήση ενός συνόλου modules (Click Elements) τα οποία μπορούν να εκτελεστούν τόσο σε επίπεδο χρήστη όσο και σε επίπεδο πυρήνα μέσα από το αντίστοιχο Kernel Module του Click (Click driver).

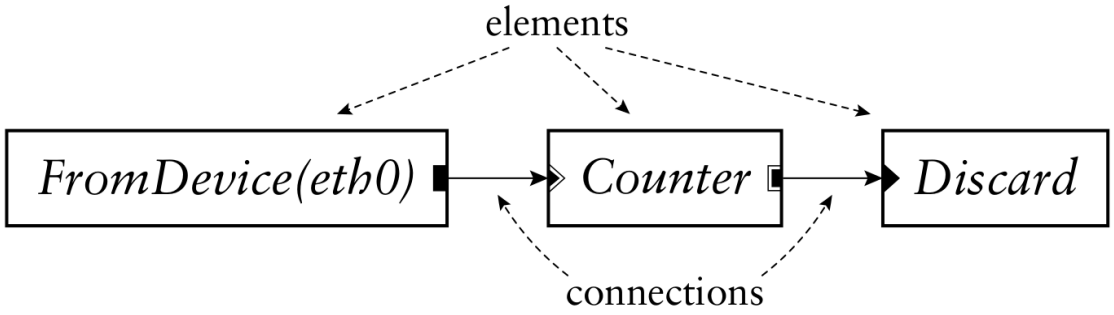
Τα elements αποτελούν το βασικό συστατικό του Click και κάθε ένα αναλαμβάνει μία συγκεκριμένη επεξεργασία. Ο χρήστης έχει την δυνατότητα να δημιουργήσει δικά του (με χρήση C++) ή να εκμεταλλευτεί τα υπάρχοντα που παρέχονται από το click. Κάθε element μπορεί να έχει input και/ή output ports τα οποία αποτελούν το σημείο διασύνδεσής τους με άλλα elements ενώ μπορεί να δέχεται και κάποιο configuration string για την αρχικοποίησή του.

Μέσα από ένα αρχείο διαμόρφωσης (configuration) δημιουργείται ένας κατευθυνόμενος γράφος ο οποίος έχει σαν κόμβους τα elements, που κάνουν την επεξεργασία, ενώ οι ακμές, που ονομάζονται connections, ορίζουν τα πιθανά μονοπάτια που μπορεί να ακολουθήσει ένα πακέτο.

Στην Εικόνα 4.3 φαίνεται πώς μπορούμε να απεικονίσουμε γραφικά ένα element ενώ η Εικόνα 4.4 παρουσιάζει ένα απλό configuration το οποίο ορίζει τρία (3) elements και τις μεταξύ τους συνδέσεις για την δημιουργία ενός router για την μέτρηση των εισερχόμενων πακέτων.



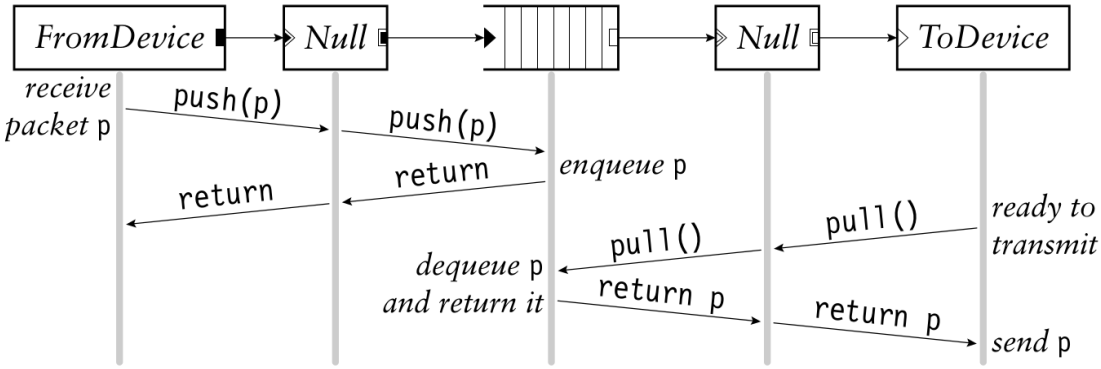
Εικόνα 4.3 Click Sample Element



Εικόνα 4.4 Click Sample Configuration

Κάθε connection και τα αντίστοιχα port των elements μπορούν να οριστούν είτε σαν pull είτε σαν push. Με αυτόν τον τρόπο δίνεται η δυνατότητα στα elements είτε να δέχονται πακέτα (push) και να κάνουν τις αντίστοιχες λειτουργίες είτε να ζητάνε πακέτα (pull) όταν τελειώσουν την λειτουργία που έκαναν. Τέτοιο παράδειγμα είναι τα element FromDevice και ToDevice. Η έξοδος του FromDevice λειτουργεί ως push, ώστε να προωθεί άμεσα τα πακέτα στα επόμενα elements, ενώ η είσοδος του ToDevice λειτουργεί ως pull ώστε να δέχεται πακέτα μόνο όταν μπορεί να τα προωθήσει στην κάρτα δικτύου.

Καθώς επιτρέπονται συνδέσεις μόνο μεταξύ όμοιων ports, υπάρχουν και τα βοηθητικά elements Queue και Dequeue, τα οποία αναλαμβάνουν την μετατροπή από push σε pull και αντίστροφα. Ένα τέτοιο παράδειγμα διασύνδεσης φαίνεται στην Εικόνα 4.5 όπου φαίνονται τα αντίστοιχα elements και οι διαδικασίες που πραγματοποιούνται αυτόματα από το click.



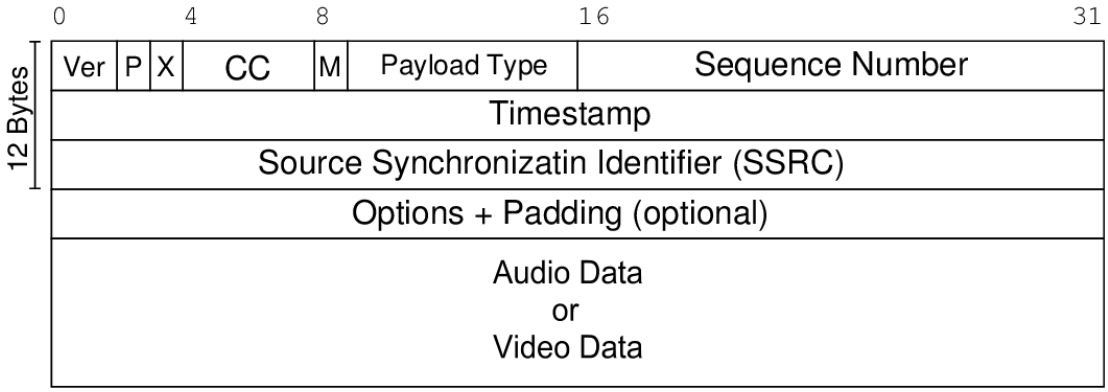
Εικόνα 4.5 Click Push-Pull Flow

Ορισμένα elements, που παρέχει το Click, λειτουργούν μόνο σε επίπεδο χρήστη ή μόνο σε επίπεδο πυρήνα. Για τον λόγο αυτό μπορεί να είναι απαραίτητες ορισμένες τροποποιήσεις του configuration, ώστε να λειτουργήσει ανάλογα το επίπεδο που στοχεύει κάθε φορά.

5 Υλοποίηση SFU

Η βασική αρχή λειτουργίας του SFU είναι η αναπαραγωγή και αποστολή των ροών προς όλους τους αποδέκτες που αφορά. Η μετάδοση των ροών γίνεται με το πρωτόκολλο *Real-time Transport Protocol* (*RTP)* [9] το οποίο δημιουργήθηκε για την μεταφορά ήχου και εικόνας πάνω από δίκτυα IP.

Κάθε πακέτο RTP αποτελείται από ένα SSRC identifier το οποίο χρησιμοποιείται για την αναγνώριση της ροής στην οποία ανήκει. Κατά την υλοποίηση του SFU, είναι το μοναδικό πεδίο του RTP που χρησιμοποιείται καθώς το σύστημα δεν πραγματοποιεί κανέναν άλλο έλεγχο ή επεξεργασία στα πακέτα.



Εικόνα 5.1 RTP Header

Για τον έλεγχο του SFU και την δυνατότητα προσθήκης ροών αλλά και συμμετεχόντων σε αυτές χρησιμοποιείται ένα πρωτόκολλο βασισμένο στις λειτουργίες του *Session Initiation Protocol* (*SIP*) [*10*] με την ακόλουθη δομή:

88 bit – 11 Bytes			
8 bit	32 bit	32 bit	16 bit
Action	FlowID (SSRC)	Address	Port

Το πεδίο Action λαμβάνει τις ακόλουθες τιμές με τις αντίστοιχες λειτουργίες:

- 0x01 Προσθήκη συμμετέχοντα σε σύνοδο
- 0x02 Αφαίρεση συμμετέχοντα από σύνοδο
- 0x03 Προσθήκη ροής συνόδου
- 0x04 Αφαίρεση ροής συνόδου

Το πεδίο FlowID είναι ίδιο με το SSRC του RTP πρωτοκόλλου ενώ τα Address και Port δηλώνουν τον αντίστοιχο δημιουργό ή αποδέκτη της ροής.

Η υλοποίηση του SFU έγινε σε Ubuntu Linux 10.04 με χρήση της γλώσσας C++ και πραγματοποιήθηκε σε τρία (3) στάδια, για την μέτρηση της απόδοσης:

- Εφαρμογή σε επίπεδο χρήστη
- Click configuration
  - σε επίπεδο χρήστη
  - σε επίπεδο πυρήνα.

Οι δύο υλοποιήσεις του click χρησιμοποιούν παρόμοιο configuration με διαφορές που έγκειται κυρίως στα ενσωματωμένα elements και την δυνατότητα λειτουργίας τους εντός και εκτός πυρήνα.

5.1 Εφαρμογή σε επίπεδο χρήστη

Η εφαρμογή στο επίπεδο χρήστη βασίζεται σε δύο νήματα (*threads*) που εκτελούν τις επικοινωνίες σε SIP και RTP επίπεδο. Για την συνολική λειτουργία είναι απαραίτητη η δημιουργία ενός πίνακα με όλες τις διαθέσιμες ροές και τους αποδέκτες τους. Οι λειτουργίες αυτές παρέχονται από την κλάση *Communicator* η οποία κατά την δημιουργία της δέχεται τις θύρες των RTP και SIP πρωτοκόλλων που θα χρησιμοποιηθούν.

Λόγω της χρήσης πολλαπλών νημάτων είναι αναγκαία η εξασφάλιση της ακεραιότητας του πίνακα ροών. Για τον λόγο αυτό είναι απαραίτητη η χρήση ενός

σημαφόρου (*mutex*) ο οποίος πρέπει να κλειδωθεί πριν από κάθε τροποποίηση. Κατά την διάρκεια ανάγνωσης του πίνακα δεν χρησιμοποιείται κλείδωμα για λόγους αύξησης της ταχύτητας.

Κάθε ροή αποθηκεύεται στον πίνακα ροών, μέσω την κλάσης *Stream*, ενώ σαν κλειδί του πίνακα χρησιμοποιείται το SSRC της. Η χρήση του κλειδιού είναι απαραίτητη για την επιτάχυνση της διαδικασίας αναζήτησης των ροών. Κάθε Stream περιέχει μία λίστα με όλους τους αποδέκτες της ροής (κλάση *Client*).

Όλη η επικοινωνία σε SIP επίπεδο γίνεται μέσα από την κλάση *SIPCommunicator* και την εκτέλεση της *SIPCommunicator::startListening()* σε ξεχωριστό thread. Η startListening() δημιουργεί ένα socket, για επικοινωνία με πρωτόκολλο TCP και περιμένει εισερχόμενες συνδέσεις. Σε κάθε σύνδεση αποκωδικοποιεί το μήνυμα μέσω του *ComProtocol* και αναλαμβάνει την προσθήκη και την αφαίρεση ροών και συμμετεχόντων.

Για την διαχείρηση των πακέτων των ροών η Communicator δημιουργεί την *RTPCommunicator* η οποία εκκινεί σε ξεχωριστό thread την *RTPCommunicator::startListening()*. Η startListening() δημιουργεί ένα socket, για επικοινωνία με πρωτόκολλο UDP και περιμένει εισερχόμενα πακέτα. Σε κάθε εισερχόμενο πακέτο διαβάζει το SSRC του, από τον πίνακα των ροών βρίσκει όλους τους παραλήπτες, το αναπαράγει και το αποστέλλει στο socket προσθέτοντας την διεύθυνση και τη πόρτα του αντίστοιχου παραλήπτη.

5.2 Click Configuration

Για την χρήση του Click Modular Router, ως SFU για το NMP, απαραίτητη είναι η δημιουργία ενός καινούργιου element, το *NMPHandler*, το οποίο πραγματοποιεί τις απαραίτητες λειτουργίες. Το NMPHandler διαθέτει δύο πόρτες εισόδου και δύο εξόδου, όλες τύπου PUSH. Η πόρτα εισόδου/εξόδου 0 αντιστοιχεί

στο πρωτόκολλο SIP ενώ η πόρτα εισόδου/εξόδου 1 αντιστοιχεί στο πρωτόκολλο RTP.

Η λειτουργία του NMPHandler είναι παρόμοια με αυτή του Communicator της εφαρμογής σε επίπεδο χρήστη. Όπως και το Communicator διαθέτει ένα πίνακα όπου κάθε ροή ταξινομείται με βάση το SSRC του ενώ κάθε port αναλαμβάνει και την αντίστοιχη επικοινωνία (SIP, RTP). Το NMPHandler δεν χρησιμοποιεί threads καθώς κάτι τέτοιο δεν υποστηρίζεται από το Click.

5.2.1 Επίπεδο Χρήστη

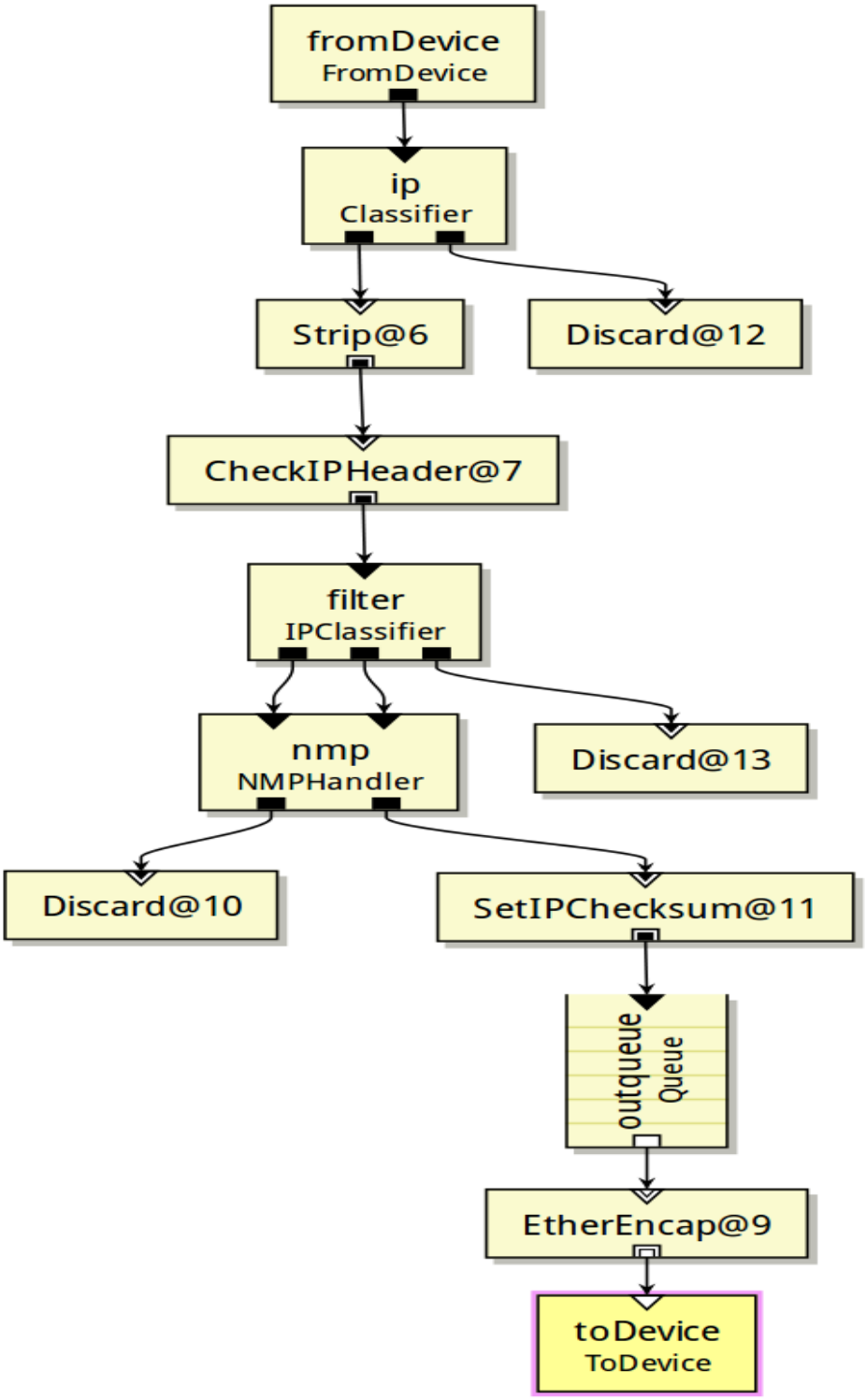
Το configuration του click σε επίπεδο χρήστη είναι αρκετά απλό. Μόνη ιδιαιτερότητα αποτελούν τα socket και η λειτουργία του NMPHandler. Το NMPHandler έχει δημιουργήθηκε ώστε να κάνει την επεξεργασία των πακέτων σε επίπεδο IP και για τον λόγο αυτό δεν μπορούν να χρησιμοποιηθούν άμεσα τα Socket Elements, καθώς λειτουργούν με βάση το payload. Έτσι το configuration χρησιμοποιεί Socket Elements αλλά μόνο για να διαχειριστούν τις συνδέσεις (TCP).

Το configuration βασίζεται στα ακόλουθα element με τις αντίστοιχες λειτουργίες:

- *Socket*: Δημιουργεί POSIX sockets ώστε να επιτύχει την TCP/RTP επικοινωνία
- *FromDevice*: Τα πακέτα λαμβάνονται στο configuration από την κάρτα δικτύου
- *Classifier*: Γίνεται φιλτράρισμα στα πακέτα ώστε να συνεχίσουν μόνο αυτά που είναι IPv4
    - 12/0800: Ethernet Type IPv4
    - -: Όλα τα υπόλοιπα πάνε στο *Discard*

- *Discard*: Ειδικό element που απορρίπτει πακέτα για τα οποία δεν μας ενδιαφέρει η συνέχιση της επεξεργασίας τους.
- *Strip*: Αφαιρεί συγκεκριμένα bytes από την αρχή του πακέτου. Χρησιμοποιείται για την αφαίρεση του ethernet header ώστε να είναι πιο εύκολη η επεξεργασία του.
- *CheckIPHeader*: Ελέγχει το πακέτο αν έχουν σωστό IP Header και το χαρακτηρίζει αναλόγως ώστε να μπορεί να επεξεργαστεί ως IP.
- *IPClassifier*: Φιλτράρισμα των πακέτων ως προς το περιεχόμενό τους και σε ποιό service αντιστοιχούν (πρωτόκολλο, θύρα, flags)
    - *dst tcp port 10000 and psh*: Όλα τα TCP πακέτα προς την θύρα 10000 που έχουν το flag "PUSH". Στην θύρα 10000 λειτουργεί το SIP πρωτόκολλο ενώ ξεχωρίζουμε το push ώστε να λάβουμε τα δεδομένα, δεν μας ενδιαφέρουν τα SYN, ACK, FIN κτλ καθώς τα χειρίζεται το socket από το shell
    - *dst udp port 10001*: Όλα τα πακέτα προς την UDP θύρα 10001 στην οποία λειτουργεί το RTP πρωτόκολλο
    - -: Όλα τα υπόλοιπα πακέτα πάνε στο Discard
- *NMPHandler*: Χειρισμός των πακέτων SIP και RTP στα ports 0 και 1 αντίστοιχα.
    - Output Port 0: τα πακέτα πάνε στο Discard καθώς την σύνδεση την έχει αναλάβει το socket στο shell.
    - Output Port 1: Από αυτό το port τα RTP πακέτα γίνονται push σε ένα QUEUE για να προωθηθούν πίσω στην κάρτα δικτύου.
- *EtherEncap*: Συμπλήρωση του πακέτου με το απαραίτητο Ethernet Header
- *ToDevice*: Αποστολή των πακέτων πίσω στο δίκτυο

Το διάγραμμα λειτουργίας του configuration φαίνεται στην Εικόνα 5.2.



Εικόνα 5.2 Click user configuration

5.2.2 Επίπεδο Πυρήνα

Σε επίπεδο πυρήνα το Click λειτουργεί με παρόμοιο τρόπο με αυτό σε επίπεδο χρήστη. Βασική διαφορά είναι ο τρόπος χειρισμού των πακέτων καθώς το click λαμβάνει προτεραιότητα έναντι του πυρήνα και τα πακέτα που απορρίπτουμε αγνοούνται εντελώς από αυτόν. Επίσης, καθώς έχουμε τον πλήρη έλεγχο των πακέτων, πρέπει να φροντίσουμε για όλες τις λειτουργίες σε επίπεδο TCP (Handshake, ACKs κτλ).

Το configuration βασίζεται στα ακόλουθα element με τις αντίστοιχες λειτουργίες:

- *FromDevice*: Τα πακέτα λαμβάνονται στο configuration από την κάρτα δικτύου
- *ToHost*: Επιστροφή των πακέτων στο Kenrel για συνέχιση της επεξεργασίας τους από αυτό
- *Classifier*: Γίνεται φιλτράρισμα στα πακέτα ώστε να συνεχίσουν μόνο αυτά που είναι IPv4
    - *12/0800*: Ethernet Type IPv4
    - -: Όλα τα υπόλοιπα πάνε στο ToHost
- *Strip*: Αφαιρεί συγκεκριμένα bytes από την αρχή του πακέτου. Χρησιμοποιείται για την αφαίρεση του ethernet header ώστε να είναι πιο εύκολη η επεξεργασία του.
- *CheckIPHeader*: Ελέγχει το πακέτο αν έχουν σωστό IP Header και το χαρακτηρίζει αναλόγως ώστε να μπορεί να επεξεργαστεί ως IP.
- *IPClassifier*: Φιλτράρισμα των πακέτων ως προς το περιεχόμενό τους και σε ποιό service αντιστοιχούν (πρωτόκολλο, θύρα, flags)
    - *dst tcp port 10000 and psh*: Όλα τα TCP πακέτα προς την θύρα 10000 που έχουν το flag “PUSH”. Στην θύρα 10000 λειτουργεί

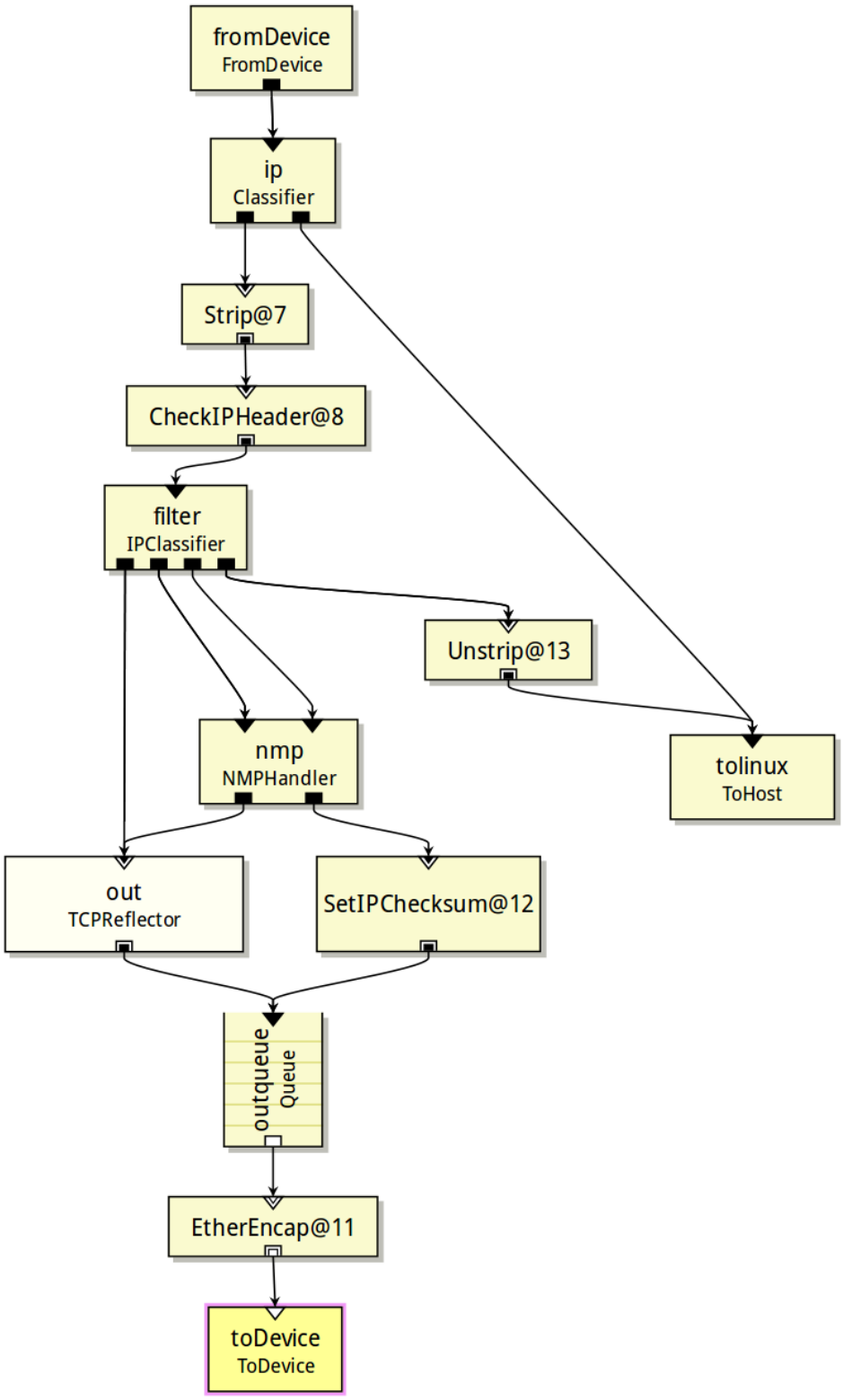
το SIP πρωτόκολλο ενώ ξεχωρίζουμε το push ώστε να λάβουμε τα δεδομένα

- *dst tcp port 10000*: Τα υπόλοιπα SIP πακέτα πρέπει να χειριστούν αναλόγως
- *dst udp port 10001*: Όλα τα πακέτα προς την UDP θύρα 10001 στην οποία λειτουργεί το RTP πρωτόκολλο
- -: Όλα τα υπόλοιπα πακέτα πάνε στο ToHost

- *TCPReflector*: Element που λειτουργεί σαν Socket με απλουστευμένη λειτουργία. Δεν κάνει έλεγχο κατάστασης αλλά απαντάει στατικά στις εισερχόμενες συνδέσεις.
- *NMPHandler*: Χειρισμός των πακέτων SIP και RTP στα ports 0 και 1 αντίστοιχα.
  - Output Port 0: τα πακέτα πάνε στο TCPReflector για να σταλεί το απαραίτητο Acknowledgement στον αποστολέα
  - Output Port 1: Από αυτό το port τα RTP πακέτα γίνονται push σε ένα QUEUE για να προωθηθούν πίσω στην κάρτα δικτύου.
- *EtherEncap*: Συμπλήρωση του πακέτου με το απαραίτητο Ethernet Header
- *Unstrip*: Επανατοποθέτηση bytes στην αρχή του πακέτου, που είχαν αφαιρεθεί με το Strip.
- *ToDevice*: Αποστολή των πακέτων πίσω στο δίκτυο

Πρέπει να σημειωθεί ότι το configuration αυτό έχει δημιουργηθεί έτσι ώστε να παρέχει την απολύτως αναγκαία λειτουργικότητα χωρίς να λαμβάνονται υπόψη παράμετροι ασφάλειας του συστήματος.

Το διάγραμμα λειτουργίας του configuration φαίνεται στην Εικόνα 5.3.



Εικόνα 5.3 Click Kernel configuration

6 Αξιολόγηση Υλοποίησης

6.1 Διαδικασία Ελέγχου

Για την μέτρηση της επίδοσης των υλοποιήσεων δημιουργήθηκε μία πειραματική διάταξη με δύο υπολογιστές διασυνδεδεμένους με Gigabit Ethernet και λειτουργικό σύστημα Ubuntu Server 10.04 και έκδοση πυρήνα 2.6.36, η οποία είναι και η προτεινόμενη για το Click Modular Router. Ο πρώτος υπολογιστής χρησιμοποιήθηκε για την λειτουργία του SFU ενώ ο δεύτερος για την δημιουργία πακέτων ροών προς το SFU.

Σε όλες τις περιπτώσεις το SFU ρυθμίστηκε ώστε να έχει 1 ροή με 10 αποδέκτες ενώ όλα τα πακέτα ροών που χρησιμοποιήθηκαν είχαν μέγεθος 80bytes. Για την αξιολόγησή του έγιναν μετρήσεις με βάση την απόδοση του συστήματος σε διάφορους ρυθμούς εισερχόμενων πακέτων (CPU Utilization) καθώς και με τον αριθμό εισερχομένων και εξερχομένων πακέτων.

Για την δημιουργία των πακέτων δημιουργήθηκαν δύο ανεξάρτητα προγράμματα, ένα για τον έλεγχο του SFU (πρωτόκολλο SIP) και ένα για την παραγωγή πακέτων ροών προς το SFU. Για τον καλύτερο έλεγχο των διαδικασιών μέτρησης δημιουργήθηκε ένα script μέσω του οποίου αυτοματοποιήθηκε η διαδικασία παραγωγής των πακέτων, ώστε να είναι ακριβώς ίδιο σε κάθε περίπτωση.

Οι μετρήσεις έγιναν με το πρόγραμμα *sar* [11], το οποίο εμφανίζει την χρήση της CPU ανά τακτά χρονικά, καθώς και με τα στατιστικά του συστήματος, από τον ψευδοφάκελο /sys/class/net, για τα εισερχόμενα-εξερχόμενα πακέτα.

Για την αυτοματοποίηση λήψης των στατιστικών δημιουργήθηκε το ακόλουθο shell script το οποίο λαμβάνει τα στατιστικά ανά δέκα (10) δευτερόλεπτα:

```
#!/bin/bash
while true
do
        R1=`cat /sys/class/net/$1/statistics/rx_packets`
        T1=`cat /sys/class/net/$1/statistics/tx_packets`
        sleep 10
        R2=`cat /sys/class/net/$1/statistics/rx_packets`
        T2=`cat /sys/class/net/$1/statistics/tx_packets`
        TX=`expr $T2 - $T1`
        RX=`expr $R2 - $R1`
echo TX: $TX   |   RX: $RX
done
```

6.2 Αποτελέσματα

Τα αποτελέσματα των μετρήσεων φαίνονται στους ακόλουθους πίνακες ομαδοποιημένα με βάση την υλοποίηση ενώ στις εικόνες Εικόνα 6.1 και Εικόνα 6.2 παρουσιάζονται συγκεντρωτικά γραφήματα των μετρήσεων:

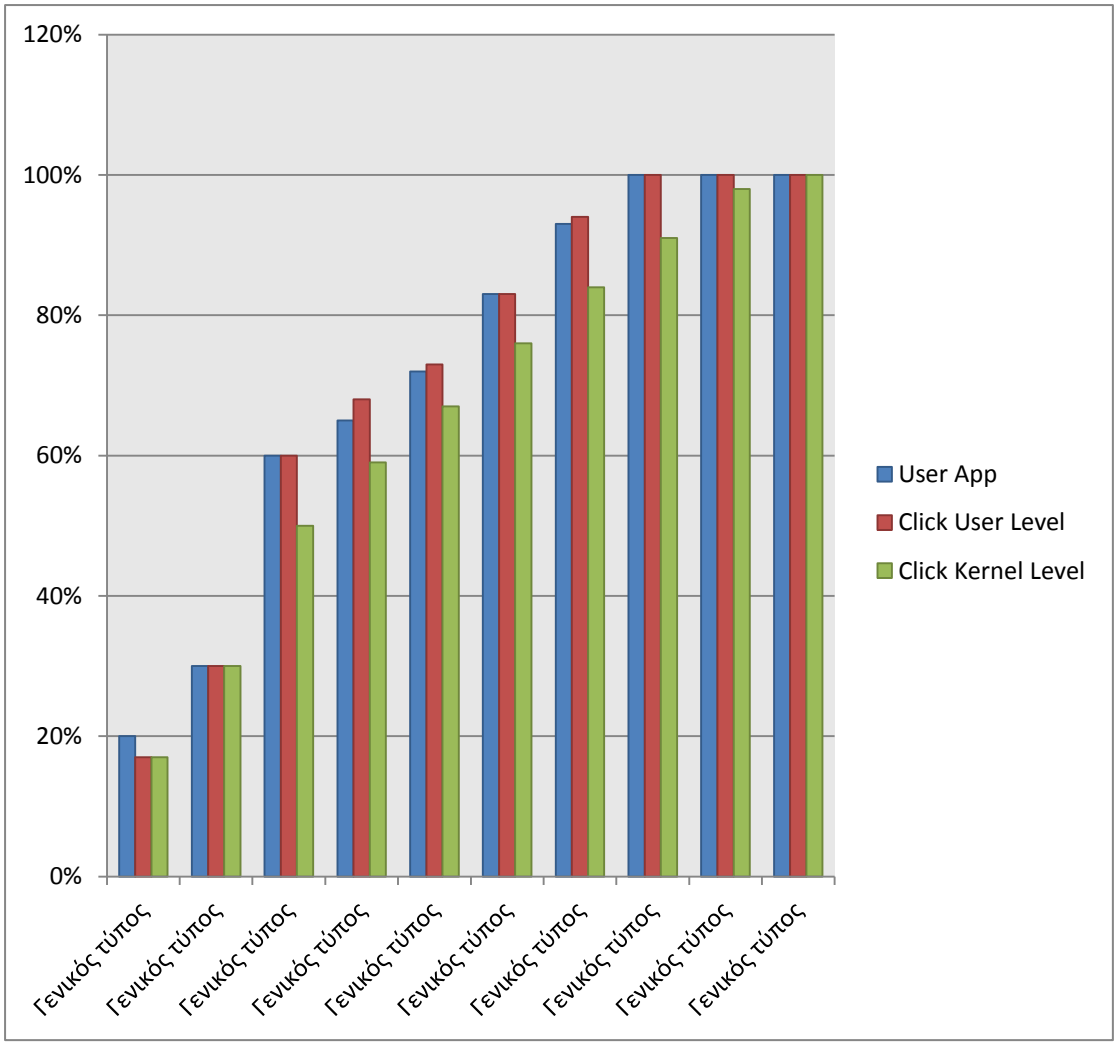
Εφαρμογή σε επίπεδο χρήστη					
	Received PPS	**Received Packets**	**Sent Packets**	**Lost Packets**	**CPU Utilization**
Run1	700	20001	199642	0	20%
Run2	1000	39807	397543	0	30%
Run3	2200	59673	596352	0	60%
Run4	2400	79557	795473	0	65%
Run5	2600	99668	996339	0	72%
Run6	2900	119969	1200235	0	83%

Run7	3100	139917	1401094	0	93%
Run8	3300	158911	1555115	3399,5	100%
Run9	3400	179609	1685671	11041,9	100%
Run10	3600	199283	1778619	21421,1	100%

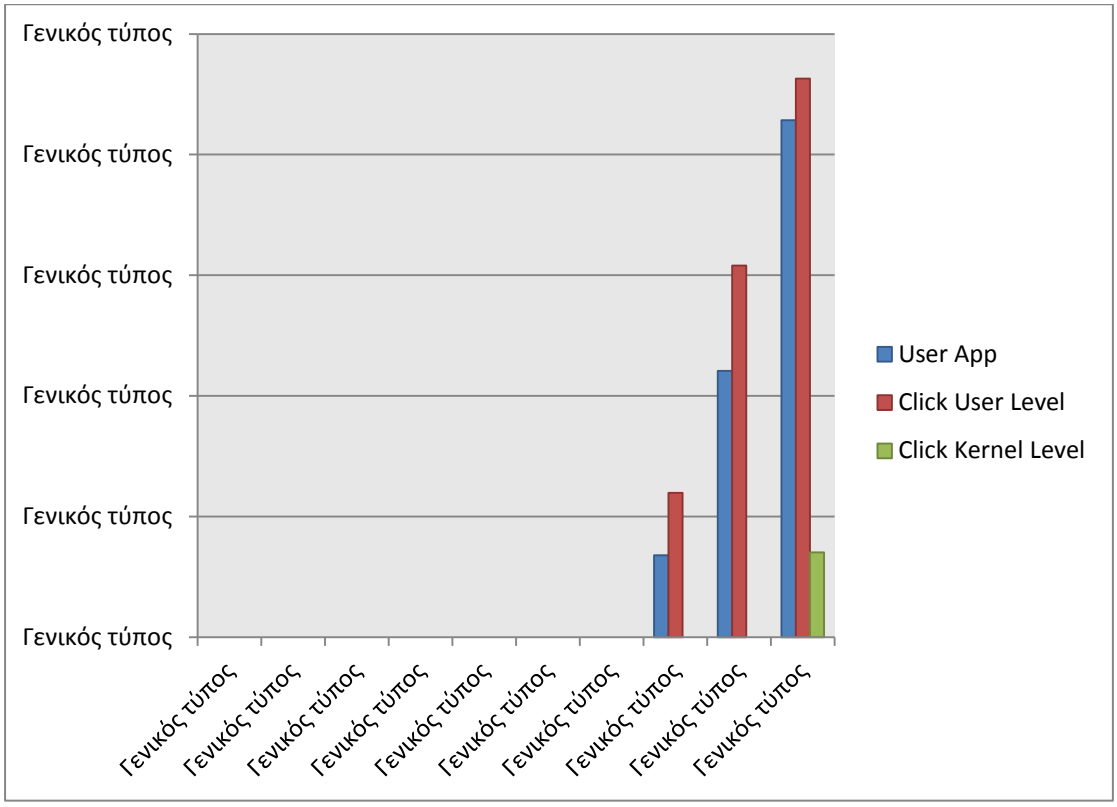
Click User level					
	Received PPS	**Received Packets**	**Sent Packets**	**Lost Packets**	**CPU Utilization**
Run1	700	19860	198561	0	17%
Run2	1000	39747	397331	0	30%
Run3	2200	59661	596581	0	60%
Run4	2400	79383	793881	0	68%
Run5	2600	99282	993582	0	73%
Run6	2900	119630	1197392	0	83%
Run7	3100	139930	1400729	0	94%
Run8	3300	158949	1529603	5988,7	100%
Run9	3400	179133	1637274	15405,6	100%
Run10	3600	198602	1754475	23154,5	100%

Click Kernel level					
	Received PPS	**Received Packets**	**Sent Packets**	**Lost Packets**	**CPU Utilization**
Run1	700	19870	198672	0	17%
Run2	1000	39784	397772	0	30%
Run3	2200	59701	597031	0	50%

Run4	2400	79354	793662	0	59%
Run5	2600	99289	992682	0	67%
Run6	2900	119363	1194662	0	76%
Run7	3100	138972	1391406	0	84%
Run8	3300	158130	1583356	0	91%
Run9	3400	178393	1789072	0	98%
Run10	3600	198690	1951698	3520,2	100%



Εικόνα 6.1 CPU Utilization vs PPS



Εικόνα 6.2 Packet loss vs PPS

6.3 Συμπεράσματα

Φαινομενικά το click θα έπρεπε να συμπεριφέρεται καλύτερα σε σχέση με τη εφαρμογή καθώς χρησιμοποιεί socket μόνο στην λήψη πακέτων ενώ ταυτόχρονα τα πακέτα επεξεργάζονται σε επίπεδο IP και αποστέλλονται άμεσα στην κάρτα δικτύου. Από τα αποτελέσματα των μετρήσεων όμως παρατηρούμε ότι το click σε επίπεδο χρήστη συμπεριφέρεται σχεδόν χειρότερα από την εφαρμογή σε επίπεδο χρήστη.

Η συμπεριφορά αυτή είναι αποτέλεσμα της γενικής φύσης και του modularity του click το οποίο προσθέτει επιπλέον φόρτο για την σύνδεση και εκτέλεση των διαφορετικών elements.

Σε επίπεδο πυρήνα τα αποτελέσματα είναι καλύτερα καθώς πλέον εξαλείφονται όλων των ειδών οι επικοινωνίες ανάμεσα στον πυρήνα και το επίπεδο χρήστη. Η αύξηση της επίδοσης είναι της τάξης του 7,5% η οποία δεν μπορεί να

θεωρηθεί ιδιαίτερα καλή, σε σύγκριση με διαφορετικές μεθόδους που πετυχαίνουν βελτίωση της τάξης του 75-80% [12].

Από τις μετρήσεις φαίνεται ότι το Network I/O μπορεί να έχει σημαντικό αντίκτυπο σε εφαρμογές όπως το NMP. Για τον λόγο αυτό απαιτείται η πρόσβαση όσο το δυνατόν πιο κοντά στο υλικό (κάρτα δικτύου) ώστε να μειωθούν όλες οι ενδιάμεσες μεταφορές και επεξεργασίες οι οποίες δεν χρειάζονται για την εκάστοτε εφαρμογή.

7 Βιβλιογραφία

1. Chris Chafe, Scott Wilson, Randal Leistikow, Dave Chisholm, Gary Scavone - *"A simplified approach to high quality music and sound over IP"* - In Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), (Verona, Italy, December 7-9,2000)

2. N Schuett - *"The Effects of Latency on Ensemble Performance"* - Undergraduate Honors Thesis, Stanford CCRMA, 2002 - Citeseer

3. A. Eleftheriadis, R. M. Civanlar, and O. Shapiro - *"Multipoint videoconferencing with scalable video coding"* - Journal of Shejiang University - SCIENCE A, vol. 7, pp. 696–705, 2006.

4. IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1-1988)

5. G. Xylomenos, C. Tsilopoulos, Y. Thomas, and G. C. Polyzos - *"Reduced switching delay for networked music performance"* - in Packet Video Workshop (Poster Session), 2013

6. E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek - *"The Click modular router"* - ACM Transactions on Computer Systems, vol. 18, pp. 263–297, 2000

7. J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo - *"NetFPGA–an open platform for gigabit-rate network switching and routing"* - in Proc. of the IEEE Microelectronic Systems Education Conference (MSE), 2007

8. L. Rizzo - *"Netmap: a novel framework for fast packet I/O"* - in Proc. of the USENIX Annual Technical Conference (ATC), 2012

9. RFC1889 - *RTP: A Transport Protocol for Real-Time Applications*

10. RFC3261 - *SIP: Session Initiation Protocol*

11. http://sebastien.godard.pagesperso-orange.fr/

12. George Baltas and George Xylomenos, *"Evaluating the Impact of Network I/O on Ultra-Low Delay Packet Switching"*, Proceedings of the IEEE Symposium on Computers and Communications (ISCC) 2015