ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Τμήμα Μηχανικών Πληροφορικής

Broadcast P2P over DVB-T networks

Γιώργος Αλεξίου

ΑΜ: 2087

Εισηγητής: Ευάγγελος Πάλλης

Ηράκλειο

18 Σεπτεμβρίου 2014

"Silence is a source of great strength."

Lao Tzu

Abstract

BitTorrent is one of the most popular P2P file sharing applications. Making use of BitTorrent on a broadcast network such as DVB-T a large amount of data is passing through the channel multiple times. So we developed a mechanism that takes advantage of the broadcast nature of DVB-T network by using data and information which are in the "air", thus reducing the utilization of DVB-T channel and optimizing the use of the BitTorrent on broadcast networks. BitTorrent is suitable for the development of such a mechanism because of the protocol's architecture that allows us to recognize the content exchange between users.

Σύνοψη

Το BitTorrent είναι μία από τις πιο δημοφιλείς P2P εφαρμογές διαμοιρασμού αρχείων. Κάνοντας χρήση του BitTorrent σε ένα ευρείας κάλυψης δίκτυο όπως το DVB-T μεγάλος όγκος δεδομένων επαναλαμβάνεται στο κανάλι. Έτσι προχωρήσαμε στην ανάπτυξη ενός μηχανισμού όπου εκμεταλλεύεται την ευρείας κάλυψη φύση του DVB-T δικτύου αξιοποιώντας δεδομένα και πληροφορίες όπου υπάρχουν στον "αέρα", ελαττώνοντας έτσι την χρησιμοποίηση του DVB-T καναλιού και βελτιστοποιώντας την χρήση του BitTorrent σε δίκτυα ευρείας κάλυψης. Το BitTorrent είναι κατάλληλο για την ανάπτυξη τέτοιου είδους μηχανισμού καθώς η αρχιτεκτονική του πρωτοκόλλου μας επιτρέπει να αναγνωρίζουμε το περιεχόμενο που διακινείται μεταξύ των χρηστών.

Ευχαριστίες

Η πτυχιακή μου εργασία υλοποιήθηκε στο Εργαστήριο Τηλεπικοινωνιών και Δικτύων PASIPHAE του Τ.Ε.Ι Κρήτης. Θα ήθελα να ευχαριστήσω τους ανθρώπους οι οποίοι βοήθησαν για τη διεκπεραίωση της. Πρωτίστως θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Πάλλη Ευάγγελο για την εμπιστοσύνη του αλλά και την πολύτιμη βοήθεια του. Επίσης ένα μεγάλο ευχαριστώ στους καθηγητές κ. Μαρκάκη Ευάγγελο και κ. Ανάργυρο Σιδέρη για την καίρια σημασίας βοήθεια τους. Θα ήθελα να εκφράσω την εκτίμηση μου προς τον κ. Μαστοράκη Γεώργιο για την υποστήριξή του καθ' όλη τη διάρκεια της πτυχιακής εργασίας. Επιπλέον θα ήθελα να ευχαριστήσω τα μέλη του εργαστηρίου για την συνεργασία και την βοήθεια τους. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου για την ανεκτίμητη βοήθεια και υποστήριξη που μου πρόσφεραν.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Συντομογραφίες

ATSC	**A**dvanced **T**elevision **S**ystems **C**ommittee
BAP2P	**B**roadcast **A**ware **P**eer **To P**eer
CD	**C**ontent **D**istribution (Module)
CMN	**C**ell **M**ade **N**ode
CS	**C**ontent **S**torage (Module)
DHT	**D**istributed **H**ash **T**able
DVB	**D**igital **V**ideo **B**roadcasting
DVB-H	**D**igital **V**ideo **B**roadcasting - **H**andheld
DVB-RCS	**D**igital **V**ideo **B**roadcasting - **R**eturn **C**hannel via **S**atellite
DVB-S	**D**igital **V**ideo **B**roadcasting - **S**atellite
DVB-T	**D**igital **V**ideo **B**roadcasting - **T**errestrial
ISDB	**I**ntegrated **S**ervices **D**igital **B**roadcasting
P2P	**P**eer **To P**eer
PEX	**P**eer **EX**change
TCP	**T**ransmission **C**ontrol **P**rotocol
TMI	**T**Traffic **M**onitoring & **I**dentification (Module)
UDP	**U**ser **D**atagram **P**rotocol
uTP	**u**Torrent **T**ransmission **P**rotocol

Στην οικογένεια μου και σε όλους όσους στάθηκαν δίπλα μου, με ιδιαίτερη εκτίμηση και αγάπη…

Κεφάλαιο 1

Εισαγωγή

Η παρούσα πτυχιακή εργασία συμβάλλει στην αποδοτικότερη αξιοποίηση των δικτυακών πόρων σε διαδραστικά περιβάλλοντα ευρυεκπομπής, μελετώντας, σχεδιάζοντας και εφαρμόζοντας σε αυτά, ένα νέο μηχανισμό διανομής διομότιμων υπηρεσιών.

Αξιοποιώντας τις πρόσφατες εξελίξεις στην επίγεια ψηφιακή μετάδοση, και αναλύοντας την δικτυακή απόδοση ενός συστήματος επίγειας διαδραστικής Τηλεόρασης (DVB-T) [1] το οποίο παρέχει IP υπηρεσίες σύμφωνα με το μοντέλο πελάτη/διακομιστή [2], εντοπίζονται προβλήματα δικτυακής συμφόρησης στο κανάλι καθόδου τα οποία οδηγούσαν στη μειωμένη δικτυακή απόδοση του συστήματος. Πιο συγκεκριμένα, παρατηρήθηκε ότι ενώ το κανάλι καθόδου του DVB-T μπορεί πολύ εύκολα να κορεστεί κατά την μεταφορά περιεχομένου μεταξύ των CMN, άλλα τμήματα του δικτύου υπό-χρησιμοποιούνται ή παραμένουν εντελώς ανεκμετάλλευτα.

Προς την κατεύθυνση μιας πιο ισορροπημένης εκμετάλλευσης των δικτυακών πόρων σε όλα τα τμήματα του δικτύου, έγινε εφαρμογή διομότιμων τεχνολογιών, στο σύστημα ψηφιακής τηλεόρασης [2][3], οι οποίες όχι μόνο αυξήσαν τη χρήση των ανεκμετάλλευτων δικτυακών πόρων σε όλα τα τμήματα του DVB-T δικτύου αλλά και οδήγησαν στην 70% αύξηση του λόγου των μεταδιδόμενων υπηρεσιών προς το διαθέσιμο εύρος ζώνης του DVB-T καναλιού (σε σχέση πάντα με τους ταυτόχρονα διασυνδεδεμένους κόμβους). Μολονότι, την εμφανή βελτίωση που επέφερε η χρησιμοποίηση διομότιμων μηχανισμών παρατηρήθηκε το εξής φαινόμενο: πανομοιότυπα κομμάτια περιεχομένου περνούσαν παραπάνω από μια φορά από το DVB-T κανάλι καθόδου ενώ παίρνοντας υπόψη ότι σε ένα περιβάλλον ευρυεκπομπής όλοι οι χρήστες λαμβάνουν τα ιδιά δεδομένα μεσώ του καναλιού καθόδου θα ήταν αποδοτικότερο να περνούσαν μόνο μία φορά και να μην επαναδεσμεύουν το διαθέσιμο εύρος ζώνης.

Έτσι σχεδιάστηκε, υλοποιήθηκε και αξιολογήθηκε ένας πρότυπος μηχανισμός ανίχνευσης P2P περιεχομένου σε δίκτυα ευρείας κάλυψης, που επιτρέπει την δραστική μείωση του αριθμού επανεκπομπών πανομοιότυπων κομματιών περιεχομένου από το DVB-T κανάλι. Τα πειραματικά αποτελέσματα έδειξαν μεγάλη μείωση της κίνησης στο κανάλι DVB-T, αποδεικνύοντας την αποδοτικότητα του μηχανισμού.

1.1 Στόχοι

Στόχος αυτής της πτυχιακής εργασίας είναι η ανάπτυξη ενός μηχανισμού (BAP2P) ο οποίος θα κάνει πλήρη εκμετάλλευση των δικτυακών πόρων σε διαδραστικά περιβάλλοντα ευρυεκπομπής και θα αξιοποιεί τα δεδομένα και τις πληροφορίες που υπάρχουν στον "αέρα" βελτιστοποιώντας έτσι την χρήση διομότιμων υπηρεσιών σε τέτοιου είδους δίκτυα. Σε δεύτερη φάση στόχος είναι η αξιολόγηση του μηχανισμού πραγματοποιώντας πειράματα σε πραγματικό διαδραστικό περιβάλλον ευρυεκπομπής.

1.2 Δομή

Αρχικά το κεφάλαιο 2 αναφέρεται στις τεχνολογίες και τα πρωτόκολλα που χρησιμοποιήθηκαν ούτως ώστε ο αναγνώστης να δημιουργήσει ένα κατάλληλο θεωρητικό υπόβαθρό και να είναι σε θέση να κατανόηση το περιεχόμενο της εργασίας. Στο κεφάλαιο 3 γίνεται αναλυτική περιγραφή και παρουσίαση του μηχανισμού που αναπτύχθηκε. Στην συνέχεια στο κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα των πειραματικών σεναρίων που πραγματοποιήθηκαν αλλά και τα συμπεράσματα που προκύπτουν από τα αποτελέσματα, καθώς επίσης και κάποιες προτάσεις για μελλοντική εργασία.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Σε αυτό κεφάλαιο παρέχονται βασικές πληροφορίες για τους αναγνώστες που δεν είναι εξοικειωμένοι με τις τεχνολογίες και τα πρωτόκολλα που χρησιμοποιήθηκαν.

2.1 Ψηφιακή Τηλεόραση

Η ψηφιακή τηλεόραση (DTV) είναι ένας τρόπος μετάδοσης εικόνας και ήχου ψηφιακά κάνοντας επεξεργασία και πολυπλεξία σήματος, σε αντίθεση με τα αναλογικά πρότυπα τηλεόρασης όπου κάθε κανάλι χρησιμοποιεί το δικό του εύρος ζώνης για την μετάδοση. Η ψηφιακή τηλεόραση υποστηρίζει περισσότερα από ένα προγράμματα στο ίδιο εύρος ζώνης. Πρόκειται για μια πρωτοποριακή υπηρεσία που αποτελεί την πρώτη σημαντική εξέλιξη στην τεχνολογία τηλεόρασης από την έγχρωμη τηλεόραση που εμφανίστηκε στην δεκαετία του 1950. Πολλές χώρες αντικατέστησαν την αναλογική εκπομπή τηλεοπτικού σήματος με ψηφιακό αξιοποιώντας έτσι για διαφορετικούς σκοπούς το διαθέσιμο εύρος ζώνης. Αρκετές περιοχές του κόσμου βρίσκονται ακόμη σε διαφορετικά στάδια προσαρμογής και εφαρμόζουν διαφορετικά πρότυπα ψηφιακών μεταδόσεων. Η ψηφιακής τηλεόρασης υλοποιείται παγκοσμίως από τρία βασικά πρότυπα:

Πίνακας 2.1: Βασικά πρότυπα ψηφιακής τηλεόρασης

Χρησιμοποιείται	**Πρότυπο**
Αμερική	ATSC (Advanced Television Systems Committee)
Ευρώπη	DVB (Digital Video Broadcasting)
Ιαπωνία	ISDB(Integrated Services Digital Broadcasting)

Το σύνολο των προτύπων DVB που χρησιμοποιούνται στην Ευρώπη αποτελείται από τα εξής πρότυπα:

Πίνακας 2.2: Σύνολο των προτύπων DVB

Πρότυπο	**Κανάλι Επιστροφής**
DVB-H	Handheld
DVB-RCS	Return Channel via Satellite
DVB-S	Satellite
DVB-T	Terrestrial

2.1.1 DVB-T

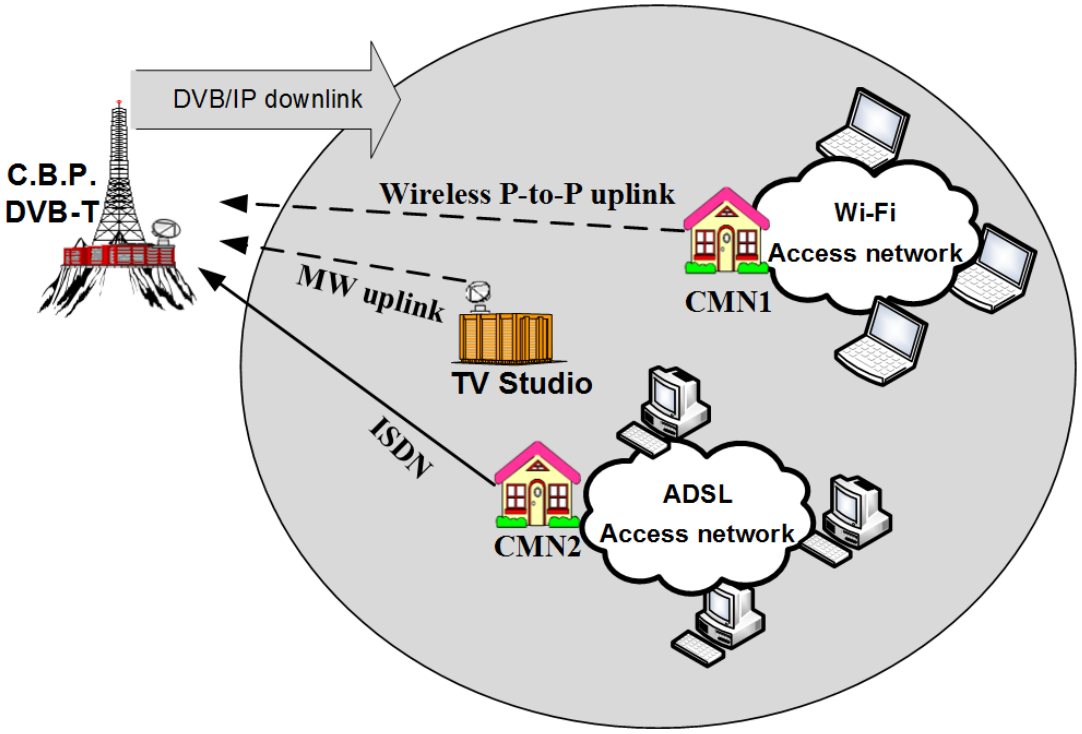
Το DVB-T [4] είναι μια τεχνολογία επίγειας μετάδοσης ψηφιακού σήματος και χρησιμοποιείται κυρίως για στη μετάδοση ψηφιακών τηλεοπτικών προγραμμάτων. Έχει σχεδιαστεί για την επίγεια μετάδοση σήματος κυρίως από σταθερά σημεία. Το DVB-T μεταδίδει εικόνα, ήχο αλλά και δεδομένα [5] σε ψηφιακή συμπιεσμένη μορφή, χρησιμοποιώντας διαμόρφωση COFDM (Coded Orthogonal Frequency Division Multiplexing)[6][7]. Ο μηχανισμός μετάδοσης χρησιμοποιεί πολλαπλές φέρουσες συχνότητες και βασίζεται στην εκπομπή ψηφιακών δεδομένων. Το DVB-T κάνει χρήση των καναλιών 21-69 της UHF μπάντας έχοντας διαθέσιμο εύρος ζώνης 8 Mhz.

Το DVB-T είναι ένα μονόδρομο πρότυπο επικοινωνίας. Σε περιπτώσεις όπου θα πρέπει να επιτευχθεί η αμφίδρομη επικοινωνία με σκοπό την αλληλεπίδραση με τον τελικό χρήστη είναι απαραίτητη η ύπαρξη κάποιου άλλου καναλιού το οποίο θα χρησιμοποιηθεί σαν κανάλι επιστροφής ώστε να γίνονται οι αιτήσεις για την υπηρεσία που ζητά ο τελικός χρήστης. Αυτό το κανάλι επιστροφής μπορεί να είναι ένα PSTN, ISDN, DSL δίκτυο η οποιαδήποτε άλλη τεχνολογία εξυπηρετεί [8][9].

2.1.2 Διαδραστική Ψηφιακή Τηλεόραση (Interactive DVB-T)

Το γενικό μοντέλο της αρχιτεκτονικής του δικτύου διαδραστικής ψηφιακής τηλεόρασης σύμφωνα με την προσέγγιση του ATHENA [1] με το οποίο ασχολήθηκε το εργαστήριο PASIPHAE, αποτελείται από 3 βασικά μέρη:

- Κεντρικό σημείο ευρυεκπομπής (Πλατφόρμα DVB-T)
- Ενδιάμεσους κόμβους διανομής (CMN)
- Τελικούς χρήστες (Clients)



Σχήμα 2.1: Μοντέλο διαδραστικής ψηφιακής τηλεόρασης

Όλη η περιοχή καλύπτεται από το ψηφιακό σήμα του DVB-T, η περιοχή χωρίζεται σε κυψέλες όπου βρίσκεται ο κάθε CMN. Κάθε CMN λαμβάνει την κίνηση από το DVB-T κανάλι μέσω της κάρτας τηλεόρασης και είναι υπεύθυνος για την δρομολόγηση της κίνησης προς τους τελικού χρήστες. Παράλληλα δρομολογεί την κίνηση των τελικών χρηστών προς την πλατφόρμα DVB-T μέσω του καναλιού επιστροφής που έχει ο κάθε CMN. Η αμφίδρομη επικοινωνία μεταξύ του κεντρικού σημείου ευρυεκπομπής και των CMN επιτυγχάνεται κάνοντας χρήση δύο διαφορετικών μονόδρομων καναλιών επικοινωνίας. Η κίνηση ενθυλακώνεται στο ρεύμα μεταφοράς DVB-T χρησιμοποιώντας την τεχνική της ενθυλάκωση πολλαπλών πρωτοκόλλων (Multi Protocol Encapsulation – MPE) δημιουργώντας έτσι ένα εικονικό δίκτυο κορμού για τις υπηρεσίες.

2.2 Πρωτόκολλα μεταφοράς

2.2.1 TCP

Το TCP (Transmission Control Protocol)[10] είναι ένα πρωτόκολλο μεταφοράς και ένα από τα βασικότερα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται σήμερα. Το βασικό χαρακτηριστικό του TCP είναι η αξιοπιστία. Κάνοντας χρήση μηχανισμών ελέγχου λαθών εξασφαλίζει την αξιόπιστη μεταφορά των δεδομένων μεταξύ 2 κόμβων. Και οι 2 πλευρές συνεργάζονται με σκοπό την σωστή μεταφορά των δεδομένων αλλά και την σωστή σειρά. Ο ένας κόμβος είναι αυτός που δημιουργεί την σύνδεση ενώ ο άλλος είναι αυτός που ακούει από την άλλη πλευρά και περιμένει για μια εισερχόμενη σύνδεση.

2.2.2 UDP

Το UDP (User Datagram Protocol)[11] είναι και αυτό από τα βασικότερα πρωτόκολλα επικοινωνίας. Είναι ένα απλούστερης μορφής πρωτόκολλο με λιγότερη επιβάρυνση για το δίκτυο σε σχέση με TCP. Στο UDP πρωτόκολλο η μεταφορά δεδομένων γίνεται χωρίς την εδραίωση της σύνδεσης και δεν παρέχει αξιοπιστία στην μετάδοση των δεδομένων, τα δεδομένα μπορεί να χαθούν η να μην φτάσουν με την σωστή σειρά.

2.2.2.1 uTP

Το πρωτόκολλο uTP (uTorrent Transmission Protocol)[12] είναι ένα πρωτόκολλο μεταφοράς δεδομένων το οποίο τρέχει πάνω στο UDP πρωτόκολλο. Το πρωτόκολλο αυτό δημιουργήθηκε με σκοπό την αντικατάσταση του TCP πρωτοκόλλου που χρησιμοποιεί το BitTorrent. Το uTP σχεδιάστηκε για να λύσει τα προβλήματα που δημιουργεί το TCP πρωτόκολλο κατά την μεταφορά δεδομένων. Το TCP είναι ένα πρωτόκολλο "αδηφάγο", καθώς δεσμεύει όσο εύρος ζώνης είναι διαθέσιμο με αποτέλεσμα να έχει ένα αθέμιτο πλεονέκτημα όταν ανταγωνίζεται με άλλες υπηρεσίες. Το uTP προσφέρει αξιοπιστία στην μεταφορά δεδομένων και παράλληλα λύνει αυτό το πρόβλημα χρησιμοποιώντας το μέγεθος της ουράς αναμονής του μόντεμ ως ελεγκτή δίνοντας του την δυνατότητα του δυναμικού ελέγχουν του ρυθμού αποστολής. Αυτό δίνει την δυνατότητα στο πρωτόκολλο να "αφήνει χώρο" για άλλες υπηρεσίες αν αυτές τον χρειάζονται ή να κάνει πλήρη αξιοποίησης του εύρους ζώνης όταν δεν υπάρχει ανταγωνισμός από άλλες υπηρεσίες.

2.3 Peer-to-Peer

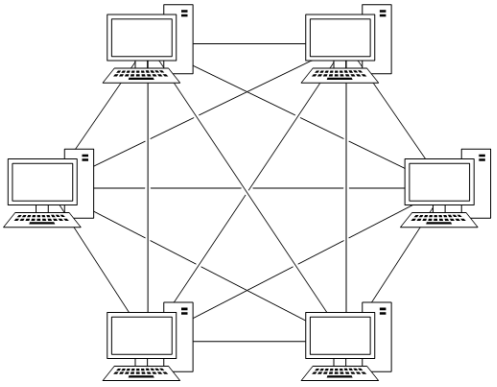
Το P2P [13] είναι ένα μοντέλο επικοινωνίας όπου ο ρόλος του κάθε χρήστη είναι διπλός. Ο κάθε χρήστης μπορεί να ζητά (client) αλλά και να προσφέρει (server) ταυτόχρονα μια υπηρεσία. Κάθε χρήστης που συνδέεται σε ένα P2P δίκτυο μέσω κάποιας P2P εφαρμογής ονομάζεται *peer*. Ένα P2P δίκτυο ορίζεται ως ένα κατανεμημένο σύστημα που αποτελείται από διασυνδεδεμένους κόμβους (peers) χωρίς να είναι απαραίτητη η διαμεσολάβηση ή η υποστήριξη από κάποιον κεντρικό server [14]. Η αποτελεσματικότητα ενός P2P συστήματος είναι ανάλογη του αριθμού των διασυνδεδεμένων peer, αυξάνεται καθώς αυξάνεται και ο αριθμός τους. Αυτό σημαίνει πως έχοντας περισσότερους διασυνδεδεμένους peers έχουμε και καλύτερη ποιότητα υπηρεσίας.

Σε ένα P2P δίκτυο όλοι οι peers συνήθως είναι ισότιμοι και έχουν τις ίδιες δυνατότητες. Τα P2P συστήματα έχουν την ικανότητα να αξιοποιούν τους διαθέσιμους πόρους κάθε peer και είναι σε θέση να διανείμουν τον φόρτο εργασίας δυναμικά. Αυτό τα κάνει να προσαρμόζονται πολύ εύκολα σε αλλαγές στην πληθικότητα των peer [14]. Το γεγονός αυτό καθιστά τα P2P συστήματα κατάλληλα για πολλούς τύπους εφαρμογών.

2.3.1 Αποκεντροποιημένο Μοντέλο

Το αποκεντροποιημένο μοντέλο μπορεί να χαρακτηριστεί σαν ένα "αμιγώς P2P μοντέλο". Όλοι οι peers είναι ισότιμοι και έχουν τις ίδιες δυνατότητες. Σε αυτό το μοντέλο δεν υπάρχει κάποιος κεντρικός διακομιστής ή κάποιος κόμβος με διαχειριστικό ρόλο που να ελέγχει και να

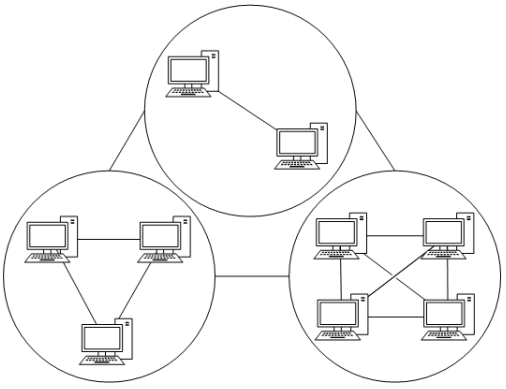
συντονίζει τις διασυνδέσεις των peer. Ως εκ τούτου οι κόμβοι αλληλεπιδρώντας μεταξύ τους αυτό-οργανώνονται με βάση τις τοπικές πληροφορίες που είναι διαθέσιμες. Ένα αποκεντροποιημένο P2P δίκτυο μπορεί να είναι *δομημένο* ή *μη δομημένο*, αυτό εξαρτάται από τον τρόπο με τον οποίο διασυνδέονται οι peers μεταξύ τους.



Σχήμα 2.2: Αποκεντροποιημένο P2P Μοντέλο

2.3.1.1 Δομημένο Μοντέλο

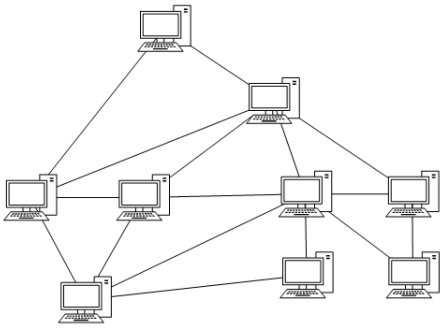
Στα *δομημένα* συστήματα οι peers διασυνδέονται με κάποια λογική ή κάποιους συγκεκριμένους κανόνες.



Σχήμα 2.3: Δομημένο P2P Μοντέλο

2.3.1.2 Μη Δομημένο Μοντέλο

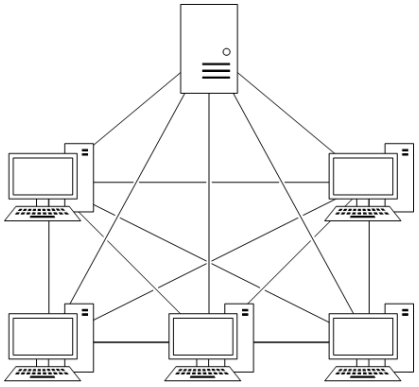
Σε αντίθεση με τα *δομημένα* συστήματα στα *μη δομημένα* η διασυνδέση των peer γίνεται με τυχαίο και αυθαίρετο τρόπο.



Σχήμα 2.4: Μή Δομημένο P2P Μοντέλο

2.3.2 Κεντροποιημένο Μοντέλο

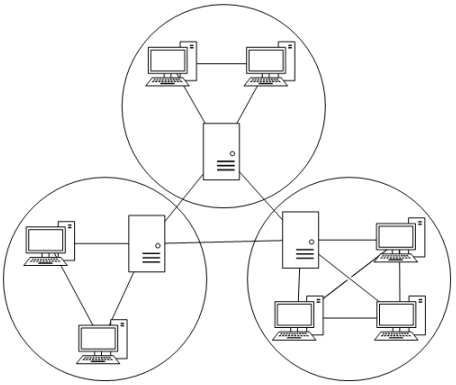
Στο κεντροποιημένο μοντέλο γίνεται χρήση ενός κεντρικού διακομιστή (server) ο οποίος είναι υπεύθυνος για λειτουργίες όπως η ανεύρεση peer και περιεχομένου. Σε αυτό το μοντέλο ο κεντρικός server περιέχει μια βάση δεδομένων με όλους τους peers που είναι συνδεδεμένοι στο P2P δίκτυο. Επίσης στην βάση του μπορεί να περιέχει πληροφορίες για τα αρχεία που είναι διαθέσιμα στο δίκτυο όπως και πληροφορίες για τις πηγές των αρχείων αυτών. Ο κάθε peer για να πραγματοποιήσει κάποια ενέργεια όπως η σύνδεση ή η αποσύνδεση από το δίκτυο αλλά και η αναζήτηση κάποιου αρχείου είναι απαραίτητη η επικοινωνία με τον server. Ωστόσο η ανταλλαγή αρχείων πραγματοποιείται απευθείας μεταξύ των peer.



Σχήμα 2.5: Κεντροποιημένο P2P Μοντέλο

2.3.3 Υβριδικό Μοντέλο

Το υβριδικό P2P μοντέλο υιοθετεί και ενσωματώνει λειτουργίες και από τα 2 προηγούμενα μοντέλα. Στο μοντέλο αυτό υπάρχουν κάποιοι κεντρικοί υπέρ-κόμβοι (superpeers) όπου παίζουν τον ρόλο του μεσάζοντα στο δίκτυο. Οι superpeers εκτελούν κυρίως δύο βασικές λειτουργίες. Ενεργούν σαν κεντρικοί κατάλογοι όπου χαρτογραφούν τους συνδεδεμένους peers και το περιεχόμενο (διαθέσιμα αρχεία) των peer.



Σχήμα 2.6: Υβριδικό P2P Μοντέλο

2.4 Πρωτόκολλο BitTorrent

Το πρωτόκολλο BitTorrent [15] είναι ένα P2P πρωτόκολλο σχεδιασμένο για διαμοιρασμό αρχείων. Αρχικά το Bittorrent ξεκίνησε σαν ένα κεντροποιημένο P2P σύστημα όπου ο κάθε peer θα έπρεπε να επικοινωνήσει με έναν κεντρικό κόμβο (Tracker) ο οποίος περιέχει την λίστα των peer που διαμοιράζονται το ίδιο αρχείο. Ωστόσο χάρη του DHT πρωτοκόλλου [16] το οποίο λειτουργεί σαν επέκταση του BitTorrent, οι peers είναι σε θέσει να κάνουν ανεύρεση και άλλων peer που διαμοιράζονται το ίδιο αρχείο χωρίς να είναι αναγκαία η επικοινωνία με τον Tracker. Με αυτό τον τρόπο είναι δυνατόν να έχουμε ένα αμιγώς αποκεντροποιημένο σύστημα. Πλέον γίνεται συνδυασμός και τον δύο τεχνικών και έτσι αν χαρακτηρίζαμε το BitTorrent πρωτόκολλο όπως το ξέρουμε σήμερα θα λέγαμε πως είναι ένα υβριδικό P2P μοντέλο. Σήμερα πλέον στις εφαρμογές BitTorrent γίνεται συνδυασμός και τον δύο τεχνικών.

Πιο συγκεκριμένα, όταν ένας χρήστης θέλει να διαμοιράσει ένα αρχείο ή μια ομάδα αρχείων, οργανώνει όλα τα αρχεία σε μια ακολουθία από bytes και τα διαιρεί σε κομμάτια (Pieces) με μια λογική σειρά, επίσης υπολογίζει το SHA1 Hash για όλα τα κομμάτια. Στην συνέχεια προσδιορίζει μια διεύθυνση διακομιστή ο οποίος είναι υπεύθυνος για την ανταλλαγή των αρχείων. Αυτός ο διακομιστής ονομάζεται Tracker. Ο Tracker είναι υπεύθυνος για την μεταξύ επικοινωνία των peer και να μπορεί να ανακαλύψει ο ένας τον άλλο με σκοπό την δημιουργία μίας γειτονιάς. Όλα αυτά μαζί με άλλες πληροφορίες (metadata) για το αρχείο ή τα αρχεία που πρόκειται να μοιραστούν καταγράφονται σε ένα μικρό αρχείο το οποίο ονομάζεται torrent. Ένας peer για να κατεβάσει τα αρχεία προϋποθέτει να έχει στην διάθεσή του αυτό το torrent αρχείο. Σε δεύτερη φάση και αφού οι peers είναι πλέων διασυνδεδεμένοι μεταξύ τους, είναι σε θέση να ανταλλάξουν τα δεδομένα.

2.4.1 BitTorrent Handshake

Το Handshake μήνυμα είναι το 1ο μήνυμα που ανταλλάσσουν οι peers μεταξύ τους και χρησιμοποιείτε για να εδραιώσουν μια σύνδεση.

- **protocol name length**: ακέραιος μεγέθους ενός byte ο οποίος προσδιορίζει το μέγεθος (σε byte) του ονόματος του πρωτοκόλλου.

- **protocol name**: μεταβλητού μήκους πεδίο το οποίο περιέχει το όνομα του πρωτοκόλλου.
- **reserved ext bytes**: μια ακολουθία από 8 bytes όπου χρησιμοποιείται για την επέκταση του πρωτοκόλλου και την υποστήριξη επιπλέον λειτουργιών
- **hash info**: ένα 20 byte SHA1 Hash των metadata των πληροφοριών που περιέχει το αρχείο Torrent
- **peer id**: μια συμβολοσειρά μεγέθους 20 bytes όπου περιέχει πληροφορίες για τον BitTorrent client, όπως όνομα και έκδοση

2.4.2 BitTorrent Μηνύματα

Στο BitTorrent πρωτόκολλο οι peers για την μεταξύ τους επικοινωνία χρησιμοποιούν συγκεκριμένης μορφής μηνυμάτα. Όλα τα μηνύματα αρχίζουν με έναν ακέραιο (4 bytes) ο οποίος προσδιορίζει το μήκος του υπόλοιπου μηνύματος, σε bytes. Στην συνέχεια ακολουθεί το id του μηνύματος που αποτελείται από 1 byte, με εξαίρεση το μήνυμα Keep-Alive το οποίο είναι ένα "κενό" μήνυμα.

2.4.2.1 Keep-Alive

Το Keep-Alive μήνυμα αποτελείτε από 4 μηδενικά bytes τα οποία προσδιορίζουν το μέγεθος του μηνύματος το οποίο είναι μηδενικό. Το συγκεκριμένο μήνυμα αποστέλλεται σε τακτά χρονικά διαστήματα για να παραμείνει ενεργή μια σύνδεση. Αν αυτό το μήνυμα δεν αποστολή μέσα στο χρονικό περιθώριο των 2 λεπτών η σύνδεση θεωρείται ανενεργή.

Πίνακας 2.3: Δομή του BitTorrent Keep-Alive μηνύματος

len
0000

2.4.2.2 Choke

Τα Choke μηνύματα έχουν μέγεθος 5 bytes και αποστέλλονται από έναν peer όταν δεν είναι σε θέση να να εξυπηρετήσει τα αιτήματα που δέχεται για την αποστολή blocks.

Πίνακας 2.4: Δομή του BitTorrent Choke μηνύματος

len	**id**
0001	0

2.4.2.3 Unchoke

Τα Unchoke μηνύματα έχουν και αυτά μέγεθος 5 bytes και αποστέλλονται από έναν peer για να δείξει πως έχει τους διαθέσιμους πόρους και είναι σε θέση να εξυπηρετήσει αιτήματα για την αποστολή blocks.

Πίνακας 2.5: Δομή του BitTorrent Unchoke μηνύματος

len	**id**
0001	1

2.4.2.4 Interested

Τα Interested μηνύματα αποτελούνται από 5 bytes και αποστέλλονται από έναν peer για να δείξει το ενδιαφέρον του για να το Torrent αρχείο και την επιθυμία του να ζητήσει διαθέσιμα Pieces.

Πίνακας 2.6: Δομή του BitTorrent Interested μηνύματος

len	**id**
0001	2

2.4.2.5 Not Interested

Τα Not Interested μηνύματα έχουν μέγεθος 5 bytes και αποστέλλονται από έναν peer για να δείξει πως δεν ενδιαφέρετε για το Torrent αρχείο.

Πίνακας 2.7: Δομή του BitTorrent Not Interested μηνύματος

len	**id**
0001	3

2.4.2.6 Have

Τα Have μηνύματα αποτελούνται από 9 bytes και αποστέλλονται από έναν peer για να πληροφορήσει πως έχει κάποιο συγκεκριμένο Piece του Torrent αρχείου. Τα 4 τελευταία bytes του μηνύματος δηλώνουν τον αριθμό του Piece στο οποίο αναφέρετε.

Πίνακας 2.8: Δομή του BitTorrent Have μηνύματος

len	**id**	**piece index**
0005	4	<piece index>

2.4.2.7 Bitfield

Τα μηνύματα Bitfield είναι μηνύματα που αποστέλλονται αμέσως μετά την ολοκλήρωσης της διαδικασίας Handshake και πριν την αποστολή οποιουδήποτε άλλου μηνύματος. Είναι ένα προαιρετικό μήνυμα και δεν είναι απαραίτητη η αποστολή του αν ο peer δεν έχει κανένα piece. Το μέγεθος του Bitfield μηνύματος είναι μεταβλητό και X είναι το μέγεθος του bitfield. Το bitfield είναι η αναπαράσταση των piece που έχει διαθέσιμα ο peer σε δυαδική μορφή (σε bit). Το bit υψηλής τάξεως (το 1ο αριστερά) αντιστοιχεί στο 1ο piece του torrent αρχείου. Τα bit τα οποία είναι 0 υποδεικνύουν τα pieces που λείπουν και αυτά που είναι 1 τα pieces που είναι διαθέσιμα. Τα bits που περισσεύουν για να ολοκληρωθεί το τελευταίο byte γίνονται 0.

Πίνακας 2.9: Δομή του BitTorrent Bitfield μηνύματος

len	**id**	**bitfield**
0001+X	5	<bitfield>

2.4.2.8 Request

Τα Request μηνύματα αποτελούνται από 17 bytes και αποστέλλονται από έναν peer για να ζητήσει ένα block. Το μήνυμα περιέχει τις παρακάτω πληροφορίες.

- **index**: ακέραιος ο οποίος προσδιορίζει τον αύξων αριθμό του piece
- **begin**: ακέραιος ο οποίος δείχνει το σημείο του piece απ όπου θα ξεκινάει το block
- **length**: ακέραιος ο οποίος προσδιορίζει το μέγεθος του block

Πίνακας 2.10: Δομή του BitTorrent Request μηνύματος

len	**id**	**index**	**begin**	**length**
0013	6	<index>	<begin>	<length>

2.4.2.9 Piece

Το Piece μήνυμα χρησιμοποιείται για να αποστείλει ένας Peer δεδομένα (blocks). Το μέγεθος του μηνύματος είναι μεταβλητό και X είναι το μέγεθος του block. Το μήνυμα περιέχει τις παρακάτω πληροφορίες.

- **index**: ακέραιος ο οποίος προσδιορίζει τον αύξων αριθμό του piece
- **begin**: ακέραιος ο οποίος δείχνει το σημείο του piece απ όπου θα ξεκινάει το block
- **block**: μια ακολουθία από bytes που περιέχουν τα δεδομένα του block

Πίνακας 2.11: Δομή του BitTorrent Piece μηνύματος

len	**id**	**index**	**begin**	**block**
0009+X	7	<index>	<begin>	<length>

2.4.2.10 Cancel

Το Cancel μήνυμα έχει μέγεθος 17 bytes και αποστέλλεται για να ακυρώσει ένα Request μήνυμα. Το περιεχόμενο του μηνύματος είναι ίδιο με του Request μηνύματος.

Πίνακας 2.12: Δομή του BitTorrent Cancel μηνύματος

len	**id**	**index**	**begin**	**length**
0013	8	<index>	<begin>	<length>

2.4.3 Διαδραστική Ψηφιακή Τηλεόραση και Peer-to-Peer

2.5 Λογισμικό και Βιβλιοθήκες

2.5.1 pcapy

Η pcapy[17] είναι μια βιβλιοθήκη γραμμένη σε Python η οποία μας επιτρέπει την σύλληψη πακέτων από την κάρτα δικτύου του υπολογιστή και την χρησιμοποιούμε για την ανίχνευση της BitTorrent κίνησης.

2.5.2 dpkt

Η dpkt είναι μια βιβλιοθήκη γραμμένη σε Python. Διαθέτει κλάσεις που αναπαριστούν δικτυακά πακέτα και υποστηρίζει τα περισσότερα γνωστά πρωτόκολλα. Ακόμα μπορούμε να περάσουμε τα πακέτα που έχουμε συλλάβει με την pcapy και να διαχειριστούμε τα διάφορα πεδία τους πολύ εύκολα.

2.5.3 AutonomoTorrent

Η εφαρμογή AutonomoTorrent[18] είναι ένας BitTorrent Client γραμμένος εξολοκλήρου σε python και τον χρησιμοποιούμε για την επικοινωνία των Client με τον μηχανισμό και την παράδοση του περιεχομένου.

2.5.4 Transmission

Ο Transmission[19] είναι ένας διαδεδομένος BitTorrent Client γραμμένος σε C++. Ο Transmission χρησιμοποιήθηκε στους Clients για να κάνουν την λήψη του περιεχομένου κατά τα πειραματικά σενάρια.

2.5.5 Vuze

Ο Vuze[20] είναι ένας από τους πιο δημοφιλείς BitTorrent Clients και είναι γραμμένος σε JAVA. Ο Vuze χρησιμοποιήθηκε στον Server ο οποίος διαμοιράζει το περιεχόμενο. Ο λόγος που στον Server έγινε χρήση διαφορετικού Client είναι ότι ο Vuze έχει ένα πολύ καλό γραφικό περιβάλλον που μας επιτρέπει να βλέπουμε την εξέλιξη του πειράματος.

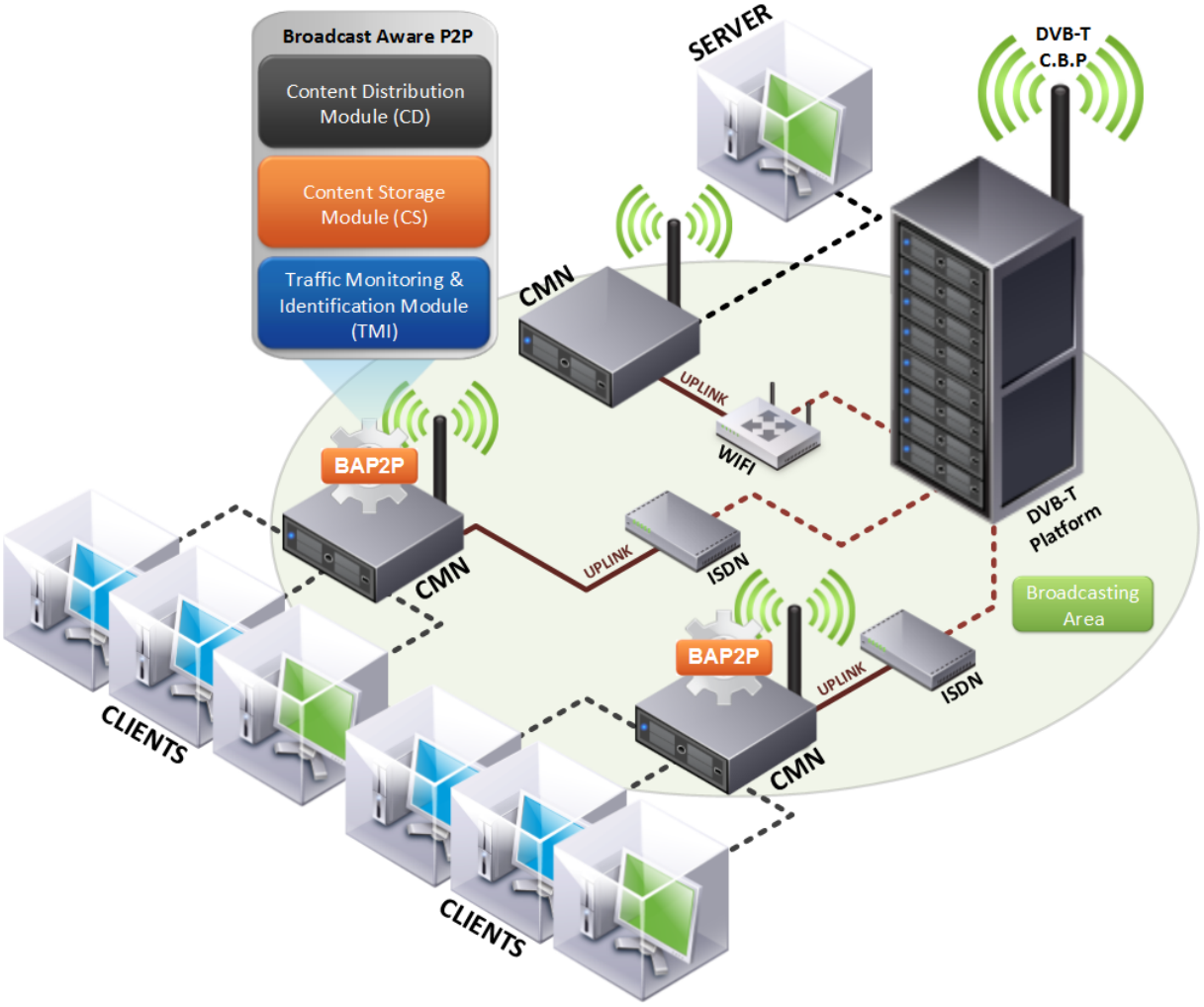
Κεφάλαιο 3

Μηχανισμός Broadcast Aware P2P

Σε αυτό το κεφάλαιο θα γίνει εκτενή αναφορά της αρχιτεκτονικής και του τρόπου λειτουργίας του μηχανισμού Broadcast Aware P2P. Σκοπός του μηχανισμού είναι η καλύτερη διαχείριση των πόρων ενός ευρείας κάλυψης δικτύου, στην προκυμμένη περίπτωση σε ένα διαδραστικό δίκτυο ψηφιακής τηλεόρασης και την βελτιστοποίηση της χρήσης του BitTorrent σε τέτοιου είδους δίκτυα.

3.1 Αρχιτεκτονική

Το σύστημα μας βασίζεται στη γενική αρχιτεκτονική που περιγράψαμε στο κεφάλαιο 2.1.2 χρησιμοποιώντας κανάλια επιστροφής WLAN για τον CMN όπου είναι συνδεδεμένος ο SERVER και ISDN κανάλια για τους υπόλοιπους CMN όπου σε αυτούς συνδέονται οι Clients. Το δίκτυο μας αποτελείται από το κεντρικό σημείο εκπομπής (πλατφόρμα DVB-T) τους ενδιάμεσους κόμβους (CMN) και τους τελικούς χρήστες (Clients). Η αμφίδρομη επικοινωνία μεταξύ της πλατφόρμας DVB-T και των CMN επιτυγχάνεται χρησιμοποιώντας σαν κανάλι καθόδου το DVB-T όπου αποστέλλονται τα δεδομένα και ως κανάλι επιστροφής ένα WLAN και ISDN κανάλια. Τέλος οι τελικοί χρήστες συνδέονται στους CMN μέσω DSL καναλιών και όλη η κίνηση δρομολογείται μέσα από τους CMN.

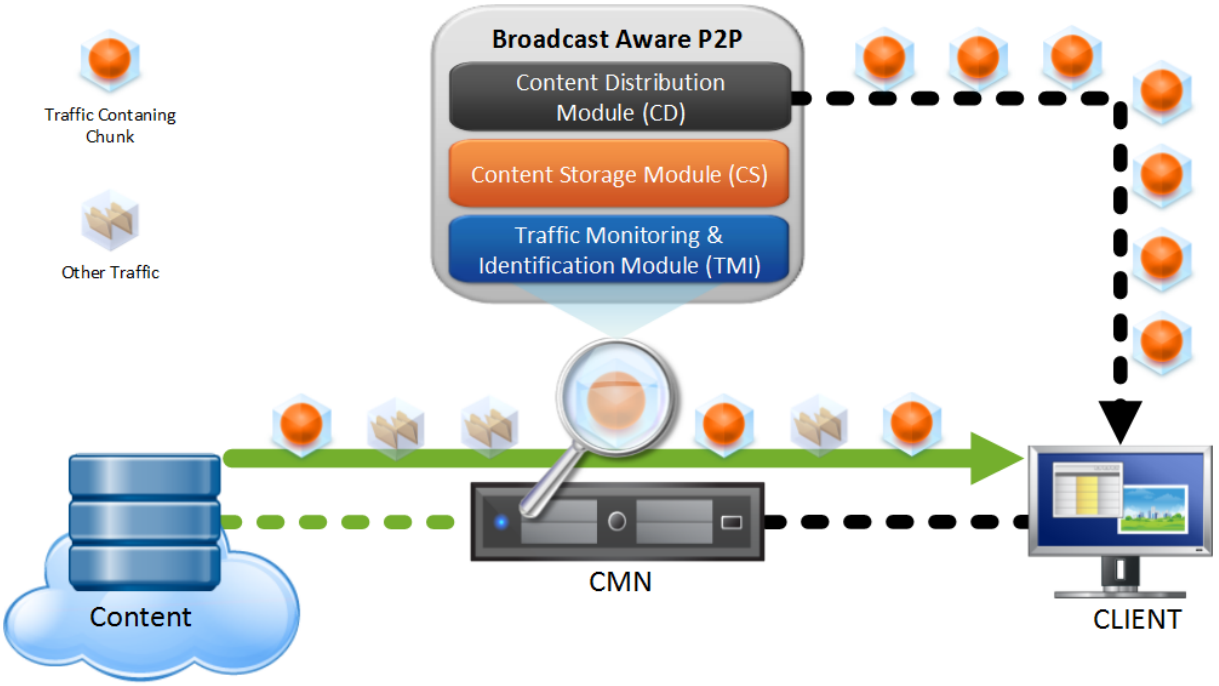


Σχήμα 3.1: Αρχιτεκτονική του μηχανισμού BAP2P

Ο μηχανισμός BAP2P αποτελείτε από 3 βασικά modules:

- **Traffic Monitoring & Identification Module (TMI)**: Ανιχνεύει την BitTorrent κίνηση
- **Content Storage Module (CS)**: Αποθηκεύει και διαχειρίζεται το περιεχόμενο
- **Content Distribution Module (CD)**: Προσφέρει το περιεχόμενο στους χρήστες

Η εγκατάσταση του μηχανισμού γίνεται στους ενδιάμεσους κόμβους (CMN) όπου έχουν άμεση πρόσβαση στην ευρυεκπομπή. Αρχικά ο μηχανισμός ανιχνεύει την BitTorrent κίνηση που κυκλοφορεί στον "αέρα" και αποθηκεύει το περιεχόμενο, στην συνέχεια ενημερώνει όλους του χρήστες της γειτονιάς του για το συγκεκριμένο περιεχόμενο που έχει αποθηκευτεί και είναι πλέον διαθέσιμο από τον μηχανισμό. Τέλος οι χρήστες είναι σε θέση να ζητήσουν το περιεχόμενο αυτό από το μηχανισμό εσωτερικά, από την γειτονία τους και χωρίς να κάνουμε χρήση του DVB-T καναλιού.



Σχήμα 3.2: Ανίχνευση και παράδοση περιεχομένου από τον μηχανισμό BAP2P

3.1.1 Traffic Monitoring & Identification Module (TMI)

Το *Traffic Monitoring & Identification Module* είναι υπεύθυνο για την ανίχνευση της BitTorrent κίνησης που κυκλοφορεί στον "αέρα" μέσω του DVB-T δικτύου. Ο μηχανισμός ελέγχει τα πακέτα, εντοπίζει τα HANDSHAKE και τα PIECE μηνύματα και τα προωθεί στο *Content Storage Module (CS)*. Ο μηχανισμός επίσης είναι σε θέση να ελέγχει ταυτόχρονα τις TCP συνδέσεις αλλά και τις uTP. Άλλη μία λειτουργία που πραγματοποιείται στον μηχανισμό είναι η ανακατασκευή των μηνυμάτων που μεταφέρονται μέσω TCP/UDP πακέτων.

3.1.2 Content Storage Module (CS)

Το *Content Storage Module* είναι αυτό που διαχειρίζεται τα δεδομένα που ανίχνευονται απο τον *TMI* μηχανισμό και είναι υπεύθυνο για τη σωστή αποθήκευση τους. Παίρνει σαν είσοδο HANDSHAKE και PIECE μηνύματα και εξάγει τις απαραίτητες πληροφορίες από τα πακέτα για καθένα από τα συγκεκριμένα μηνύματα.

Εξάγοντας αυτές τις πληροφορίες από τα HANDSHAKE μηνύματα ο μηχανισμός είναι σε θέση να γνωρίζει σε ποιο BitTorrent αρχείο αναφέρονται τα δεδομένα της κάθε σύνδεσης καταγράφοντας το *Hash Info*, την διεύθυνση IP αλλά και την θύρα του αποστολέα και του παραλήπτη. Τα PIECE μηνύματα είναι αυτά που περιέχουν τα δεδομένα (blocks) του BitTorrent αρχείου, αυτό που δεν μπορούμε όμως να γνωρίσουμε από τις πληροφορίες που υπάρχουν στο PIECE μήνυμα είναι σε πιο BitTorrent αρχείο αναφέρονται τα συγκεκριμένα δεδομένα, γι' αυτό

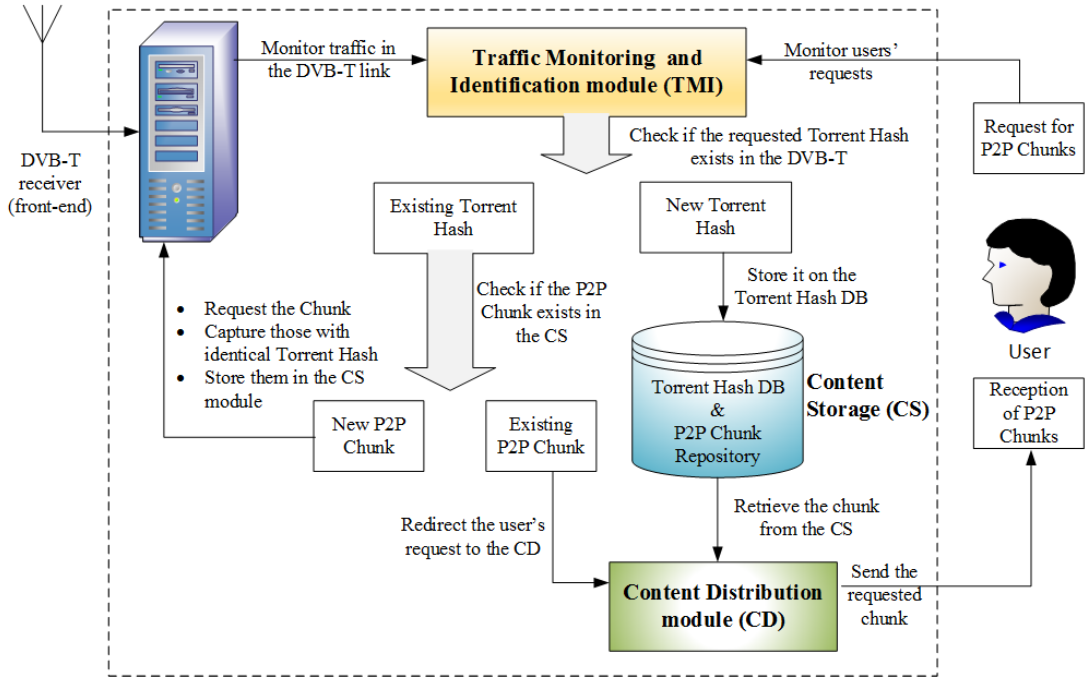
Πίνακας 3.1: Πληροφορίες που εξάγονται απο τα πακέτα

HANDSHAKE	PIECE
Διεύθυνση IP και θύρα αποστολέα.	Διεύθυνση IP και θύρα αποστολέα.
Διεύθυνση IP και θύρα παραλήπτη.	Διεύθυνση IP και θύρα παραλήπτη.
Hash Info του αρχείου Torrent.	Piece Index.
	Piece Length.
	Δεδομένα του Block.

είναι απαραίτητη η καταγραφή των πληροφοριών των HANDSHAKE μηνυμάτων. Έτσι μπορούμε πολύ εύκολα να ξέρουμε σε πιο BitTorrent αρχείο ανήκουν τα δεδομένα που υπάρχουν στο PIECE μήνυμα. Επίσης στο PIECE μήνυμα υπάρχει το *Piece Index* και το *Piece Length*, όπου μας βοηθάνε στη ταξινομημένη αποθήκευση των δεδομένων που πραγματοποιεί ο μηχανισμός, αλλά και στην εύκολη ανάκτηση τους από το *Content Distribution Module (CD)*.

3.1.3 Content Distribution Module (CD)

Το *Content Distribution Module* περιέχει τις βασικές λειτουργίες ενός BitTorrent Client και μπορεί να διαθέσει στους χρήστες κομμάτια ή και ολόκληρο το BitTorrent αρχείο που έχει αποθηκευτεί από τον μηχανισμό. Άλλη μια επιπρόσθετη λειτουργεία του μηχανισμού είναι να κάνει broadcast ένα HAVE μήνυμα σε όλου τους Peers της γειτονιάς του κάθε φορά που ένα block αποθηκεύεται και είναι διαθέσιμο από τον μηχανισμό, έτσι οι Peers τοπικά είναι σε θέση να γνωρίζουνε πως το συγκεκριμένο block είναι διαθέσιμο από τον BAP2P μηχανισμό.

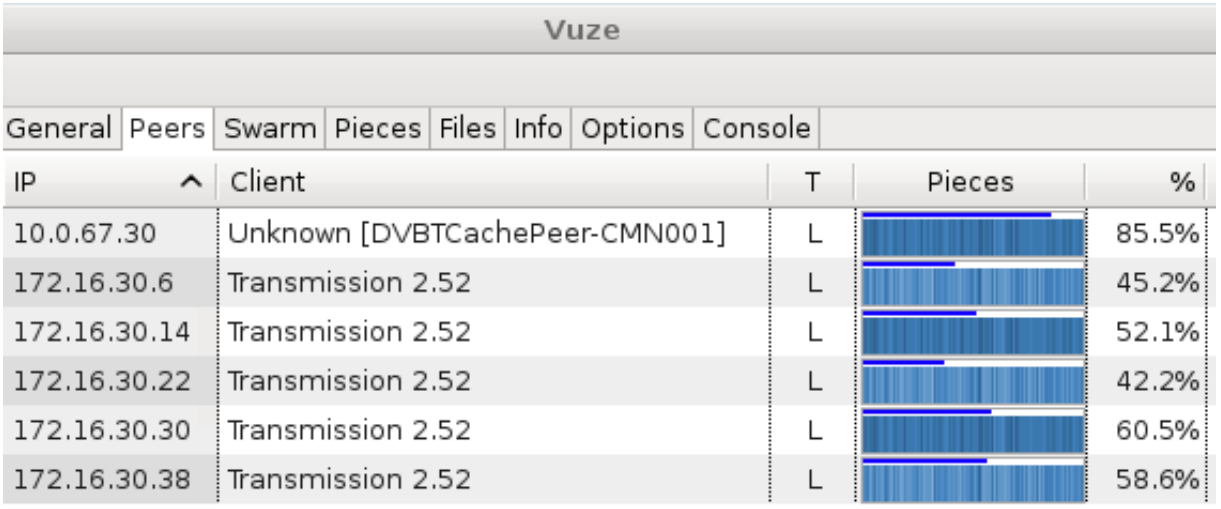


Σχήμα 3.3: Επεξεργασία των BitTorrent ροών από τον BAP2P

Κεφάλαιο 4

Αξιολόγηση του μηχανισμού BAP2P και συμπεράσματα

Στο κεφάλαιο αυτό θα γίνει η παρουσίαση των πειραματικών σεναρίων που πραγματοποιήθηκαν αλλά και των αποτελεσμάτων που προέκυψαν κατά την εφαρμογή του μηχανισμού BAP2P στο DVB-T δίκτυο καθώς και τα συμπεράσματα που προκύπτουν. Το σύνολο των πειραμάτων σχεδιάστηκε με σκοπό την επαλήθευση της εγκυρότητας της προτεινόμενης λύσης κάτω από πραγματικές συνθήκες δικτυακής κίνησης, καθώς και για την αξιολόγηση και την σύγκριση των επιδόσεων σε σχέση με το τυπικό μοντέλο BitTorrent. Γι' αυτό τον σκοπό όλοι οι Clients ζητούν ταυτόχρονα το ίδιο περιεχόμενο (το οποίο φιλοξενείται στον Server) το οποίο έχει μέγεθος 50 MB όπου αντιπροσωπεύει ένα τυπικό YouTube Video 2,5 λεπτών με ποιότητα 480p και μορφής MPEG4. Το αρχείο αυτό σύμφωνα με το BitTorrent πρωτόκολλο είναι χωρισμένο σε 1526 κομμάτια (Pieces) και κάθε κομμάτι αποτελείτε από 2 Block των 16384 bytes. Όλες αυτές οι πληροφορίες περιέχονται στο torrent αρχείο που παρέχεται στους χρήστες προκειμένου να κατεβάσουν το περιεχόμενο. Όλοι οι Clients για την παράδοση περιεχομένου χρησιμοποιούν τον BitTorrent Client Transmission και ο Server τον Vuze. Το κανάλι καθόδου DVB-T έχει εύρος 8Mbps. Το κανάλι ανόδου των Clients (ISDN) έχει εύρος 128Kbps και το κανάλι ανόδου (WLAN) του Server έχει εύρος 22Mbps. Σε όλα τα πειραματικά σενάρια σε πρώτη φάση γίνεται η παράδοση περιεχομένου με την χρήση του τυπικού μοντέλου BitTorrent, χωρίς την χρήση του BAP2P μηχανισμού και σε δεύτερη φάση γίνεται η παράδοση περιεχομένου με τον μηχανισμό BAP2P ενεργοποιημένο.

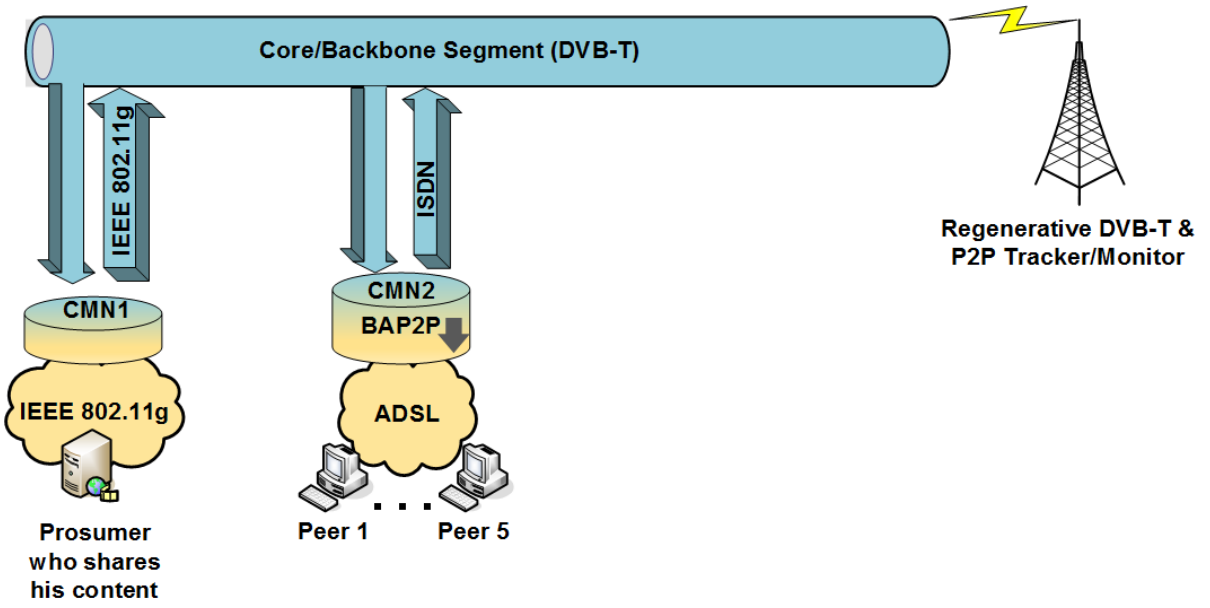


Σχήμα 4.1: Στιγμιότυπο του Vuze κατά την παράδοση περιεχομένου.

4.1 Αξιολόγηση

4.1.1 Σενάριο 1 (5 χρήστες)

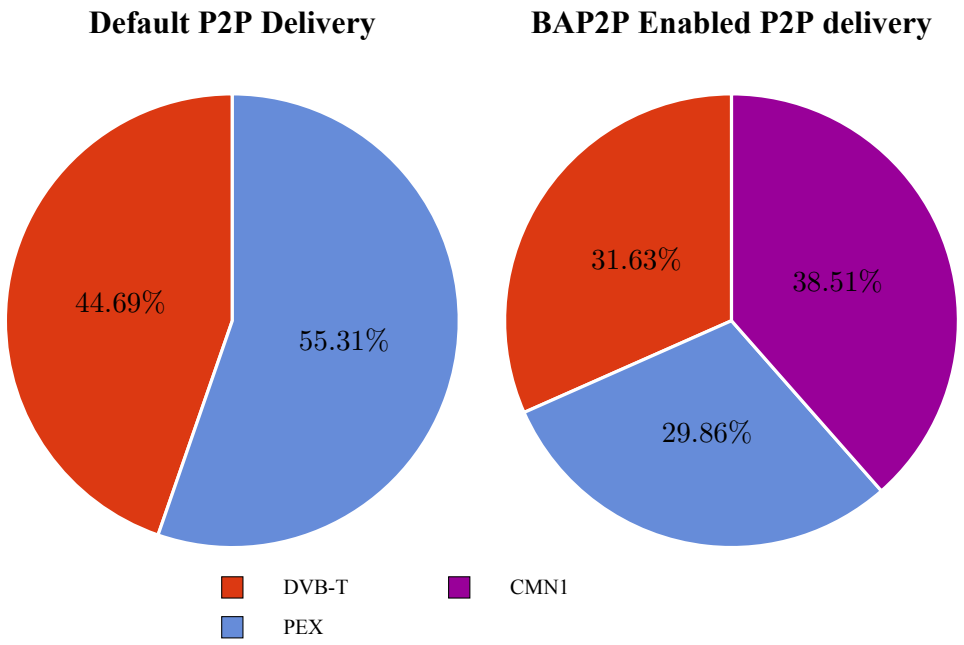
Στο πρώτο πειραματικό σενάριο υπάρχουν δύο CMN. Στον CMN1 υπάρχει ένας χρήστης όπου έχει το περιεχόμενο. Στον CMN2 υπάρχουν 5 χρήστες όπου ζητάνε το περιεχόμενο.



Σχήμα 4.2: 5 χρήστες - Σύνθεση δικτύου

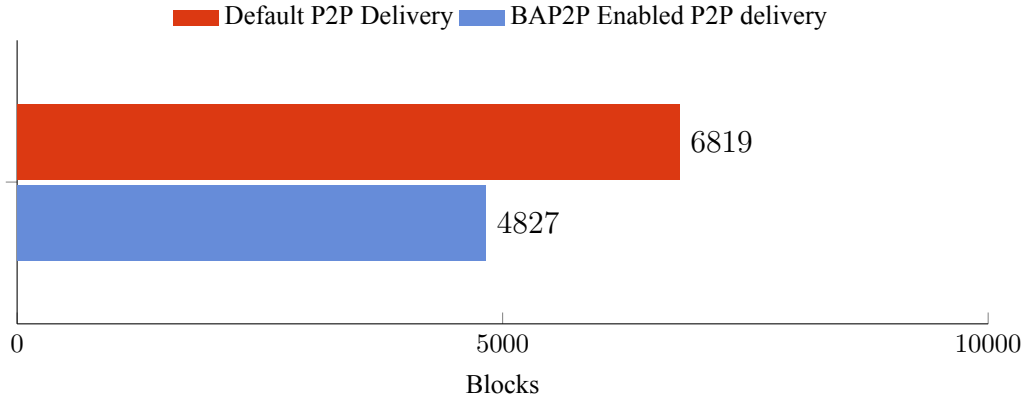
4.1.1.1 Αποτελέσματα

Στο παρακάτω σχήμα φαίνεται το ποσοστό χρησιμοποίησης όλων των καναλιών κατά την παράδοση του P2P περιεχομένου σε 5 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Είναι εμφανές πως υπάρχει σημαντική μείωση στην χρησιμοποίηση του DVB-T καναλιού.



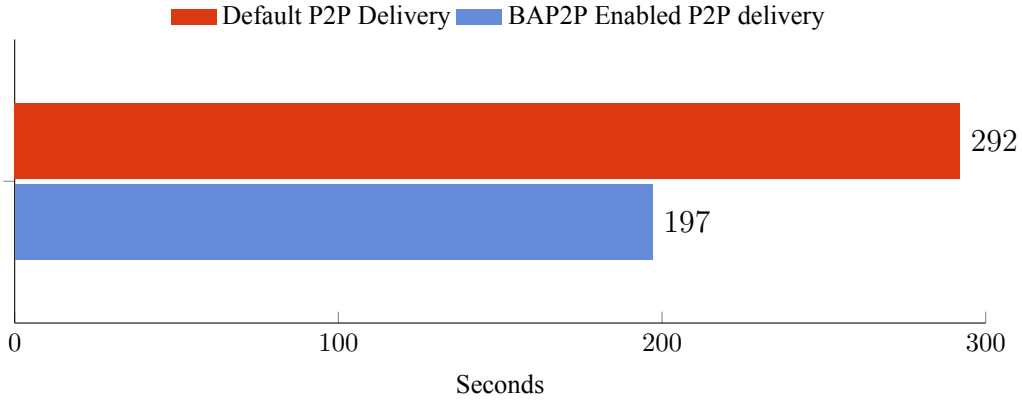
Σχήμα 4.3: 5 χρήστες - Ποσοστό χρησιμοποίησης κατά την παράδοση P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των Blocks που πέρασαν από το DVB-T κανάλι κατά την παράδοση του P2P περιεχομένου σε 5 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον αριθμό των Blocks που περνάνε από το DVB-T καθώς το μεγαλύτερο μέρος τους εξυπηρετείτε τοπικά από τον μηχανισμό.



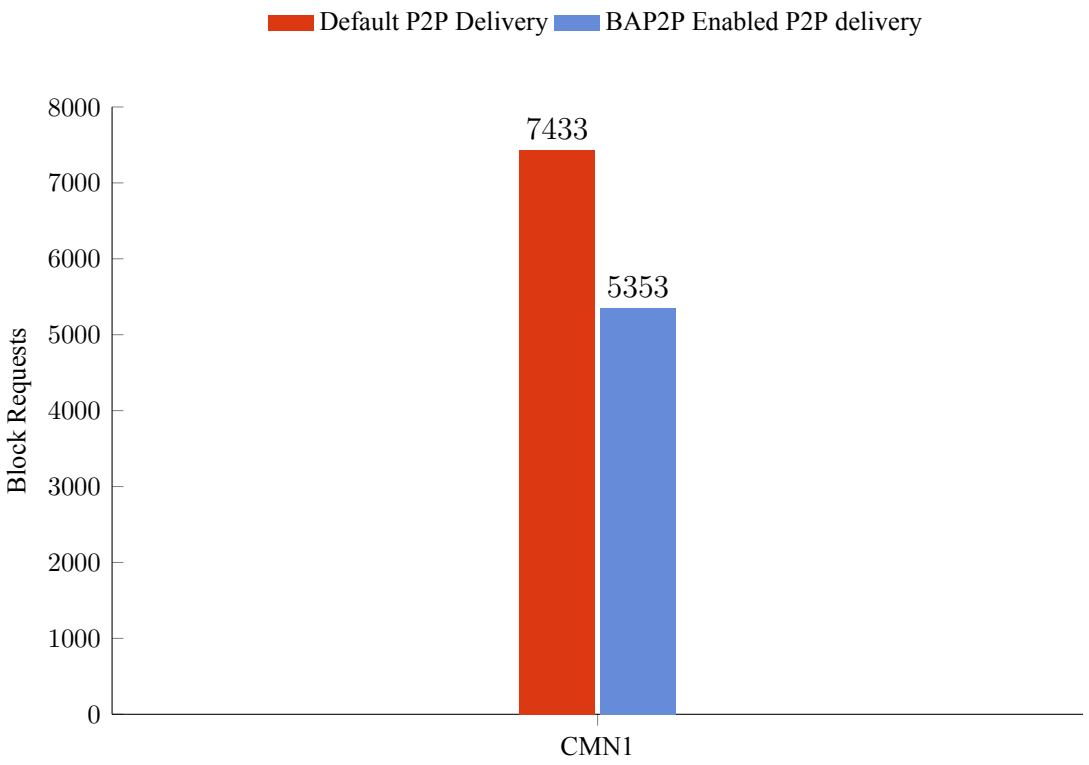
Σχήμα 4.4: 5 χρήστες - Blocks που πέρασαν απο το DVB-T

Στο παρακάτω σχήμα φαίνεται ο χρόνος ολοκλήρωσης της παράδοσης του P2P περιεχομένου σε 5 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον χρόνο παράδοσης του περιεχομένου καθώς κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T.



Σχήμα 4.5: 5 χρήστες - Διάρκεια ολοκλήρωσης της παράδοσης P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των αιτήματα για Blocks που πέρασαν από το DVB-T κανάλι ανά γειτονιά, κατά την παράδοση του P2P περιεχομένου σε 5 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο αριθμός των αιτημάτων έχει μειωθεί σημαντικά καθώς ο μηχανισμός κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T και έτσι τα Blocks που ζητάει έξω από την γειτονία του είναι πολύ λιγότερα.



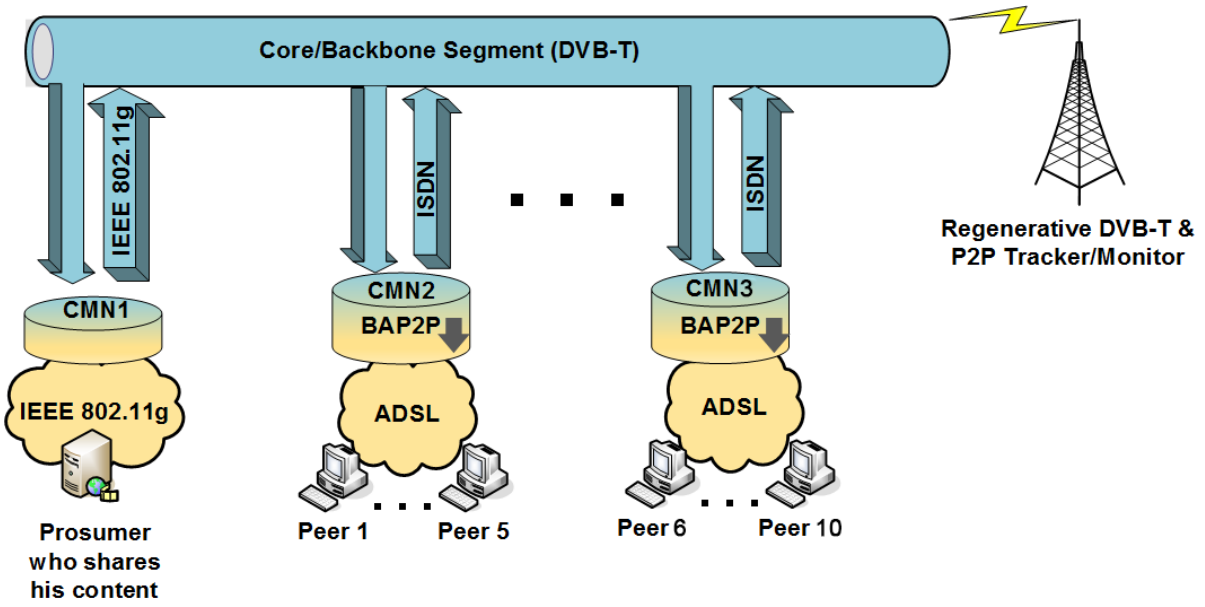
Σχήμα 4.6: 5 χρήστες - Αιτήματα για Blocks που περνάνε από το DVB-T

4.1.1.2 Αξιολόγηση

Όπως προκύπτει από τα αποτελέσματα ο μηχανισμός κατάφερε να ελαττώσει την χρησιμοποίηση του DVB-T δικτύου κατά 13,06%. Πιο συγκεκριμένα το 38,51% του περιεχομένου εξυπηρετήθηκε από τον μηχανισμό μέσω του CMN2 μειώνοντας τον αριθμό των Blocks που πέρασαν από το DVB-T κατά 1992 blocks. Τέλος ο χρόνος ολοκλήρωσης της παράδοσης βελτιώθηκε κατά 95 δευτερόλεπτα.

4.1.2 Σενάριο 2 (10 χρήστες)

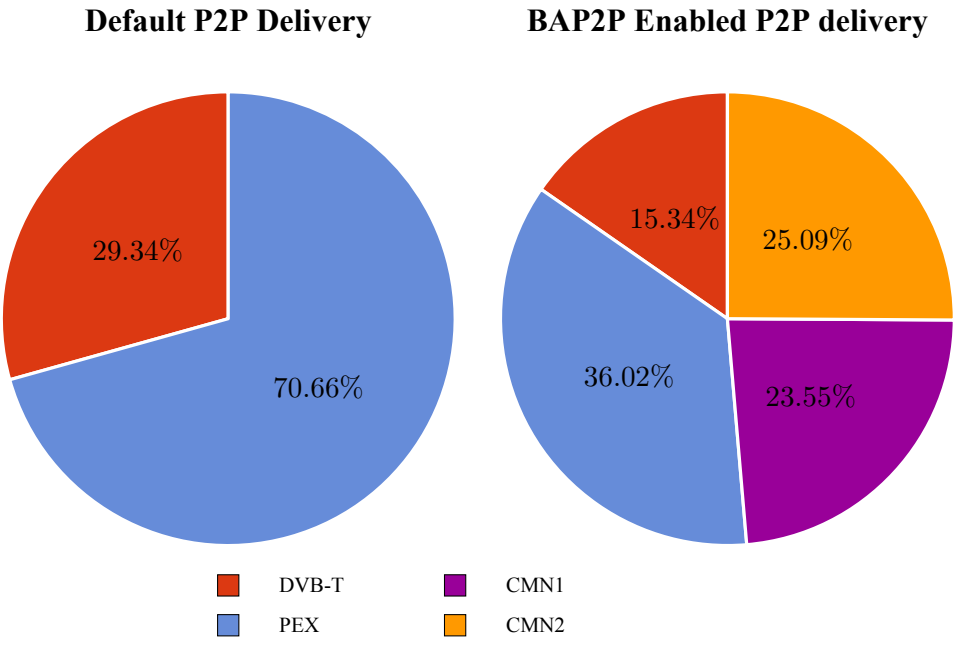
Στο δεύτερο πειραματικό σενάριο υπάρχουν τρεις CMN. Στον CMN1 υπάρχει ένας χρήστης όπου έχει το περιεχόμενο. Στους CMN2 και CMN3 υπάρχουν από 5 χρήστες όπου ζητάνε το περιεχόμενο.



Σχήμα 4.7: 10 χρήστες - Σύνθεση δικτύου

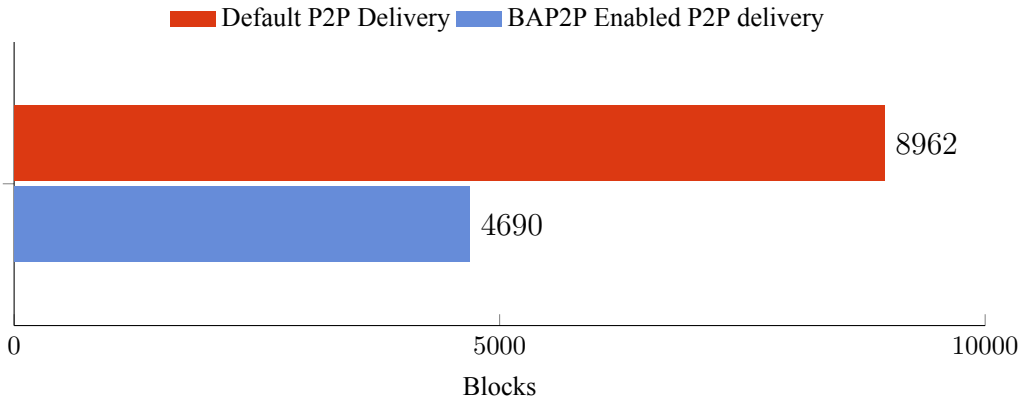
4.1.2.1 Αποτελέσματα

Στο παρακάτω σχήμα φαίνεται το ποσοστό χρησιμοποίησης όλων των καναλιών κατά την παράδοση του P2P περιεχομένου σε 10 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Είναι εμφανές πως υπάρχει σημαντική μείωση στην χρησιμοποίηση του DVB-T καναλιού.



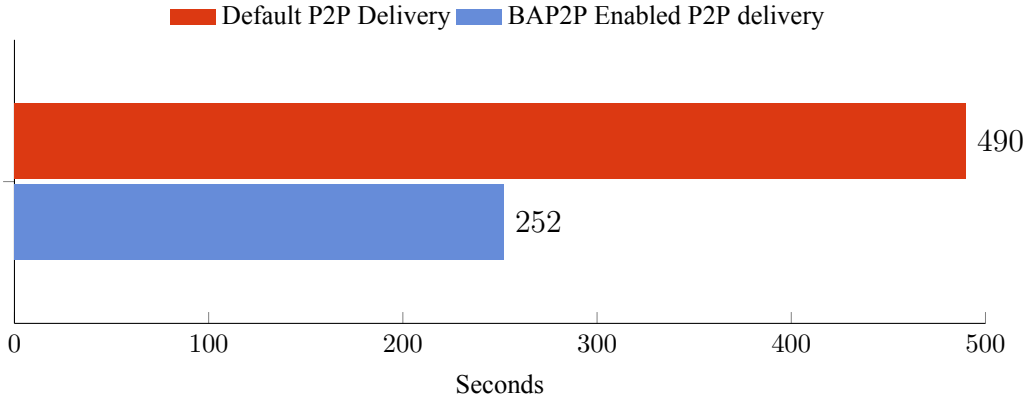
Σχήμα 4.8: 10 χρήστες - Ποσοστό χρησιμοποίησης κατά την παράδοση P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των Blocks που πέρασαν από το DVB-T κανάλι κατά την παράδοση του P2P περιεχομένου σε 10 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον αριθμό των Blocks που περνάνε από το DVB-T καθώς το μεγαλύτερο μέρος τους εξυπηρετείτε τοπικά από τον μηχανισμό.



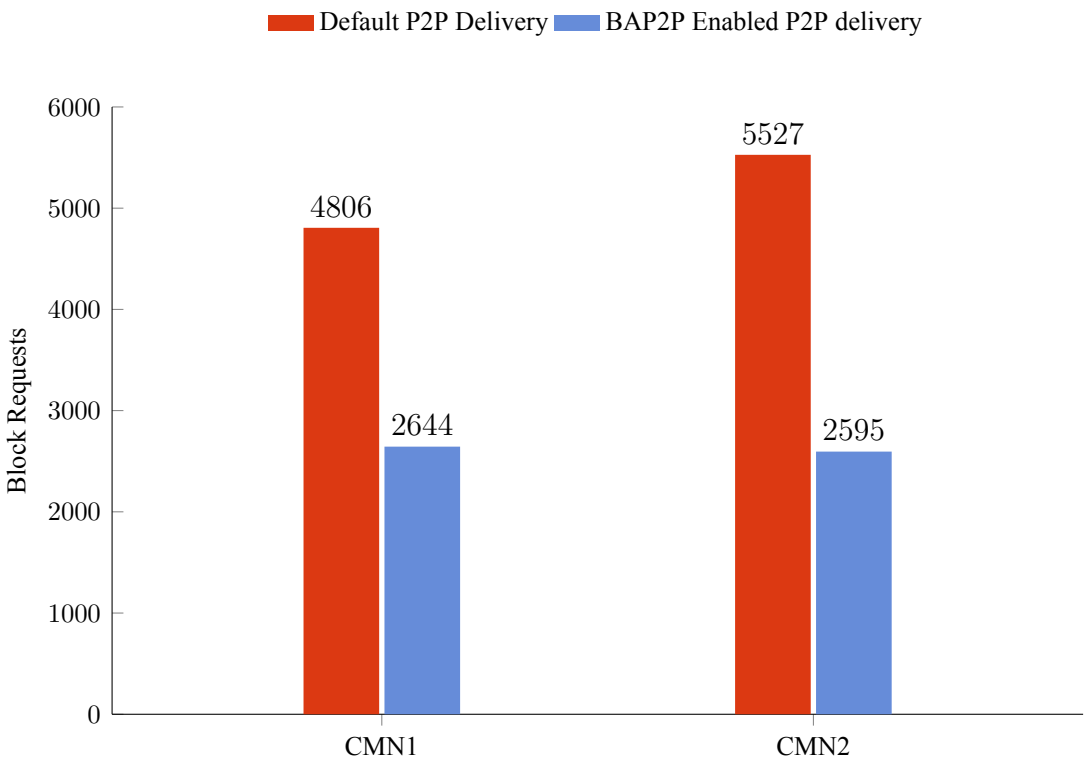
Σχήμα 4.9: 10 χρήστες - Blocks που πέρασαν απο το DVB-T

Στο παρακάτω σχήμα φαίνεται ο χρόνος ολοκλήρωσης της παράδοσης του P2P περιεχομένου σε 10 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον χρόνο παράδοσης του περιεχομένου καθώς κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T.



Σχήμα 4.10: 10 χρήστες - Διάρκεια ολοκλήρωσης της παράδοσης P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των αιτήματα για Blocks που πέρασαν από το DVB-T κανάλι ανά γειτονιά, κατά την παράδοση του P2P περιεχομένου σε 10 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο αριθμός των αιτημάτων έχει μειωθεί σημαντικά καθώς ο μηχανισμός κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T και έτσι τα Blocks που ζητάει έξω από την γειτονία του είναι πολύ λιγότερα.



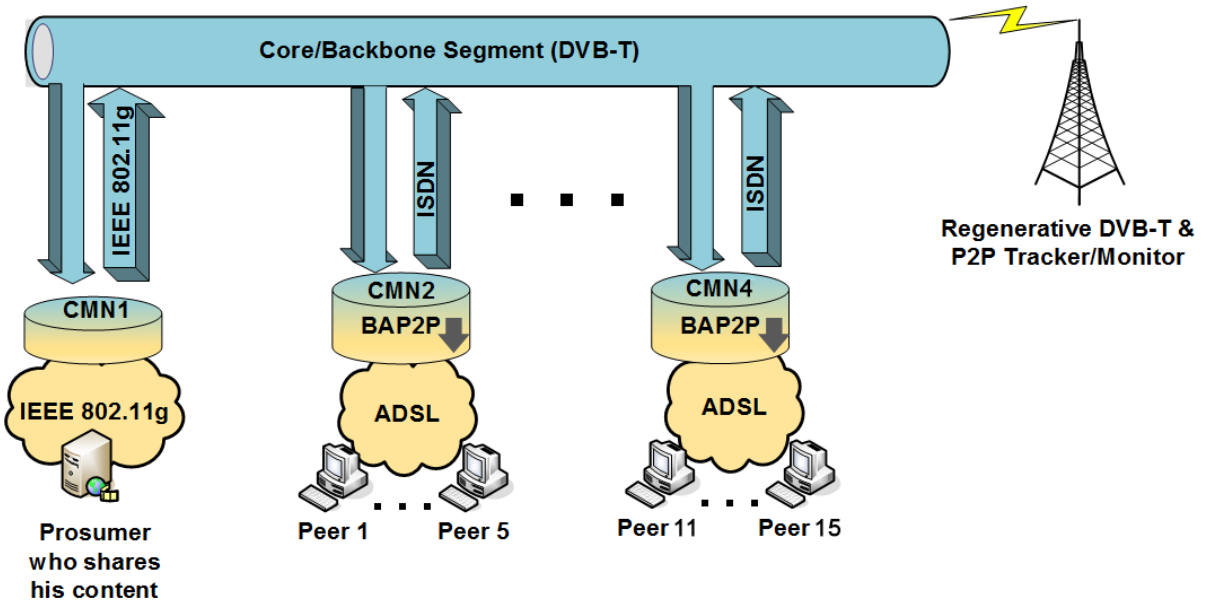
Σχήμα 4.11: 10 χρήστες - Αιτήματα για Blocks που περνάνε από το DVB-T

4.1.2.2 Αξιολόγηση

Όπως προκύπτει από τα αποτελέσματα ο μηχανισμός κατάφερε να ελαττώσει την χρησιμοποίηση του DVB-T δικτύου κατά 14%. Πιο συγκεκριμένα το 48,64% του περιεχομένου εξυπηρετήθηκε από τον μηχανισμό μέσω του CMN2 και CMN3 μειώνοντας τον αριθμό των Blocks που πέρασαν από το DVB-T κατά 4272 blocks. Τέλος ο χρόνος ολοκλήρωσης της παράδοσης βελτιώθηκε κατά 238 δευτερόλεπτα.

4.1.3 Σενάριο 3 (15 χρήστες)

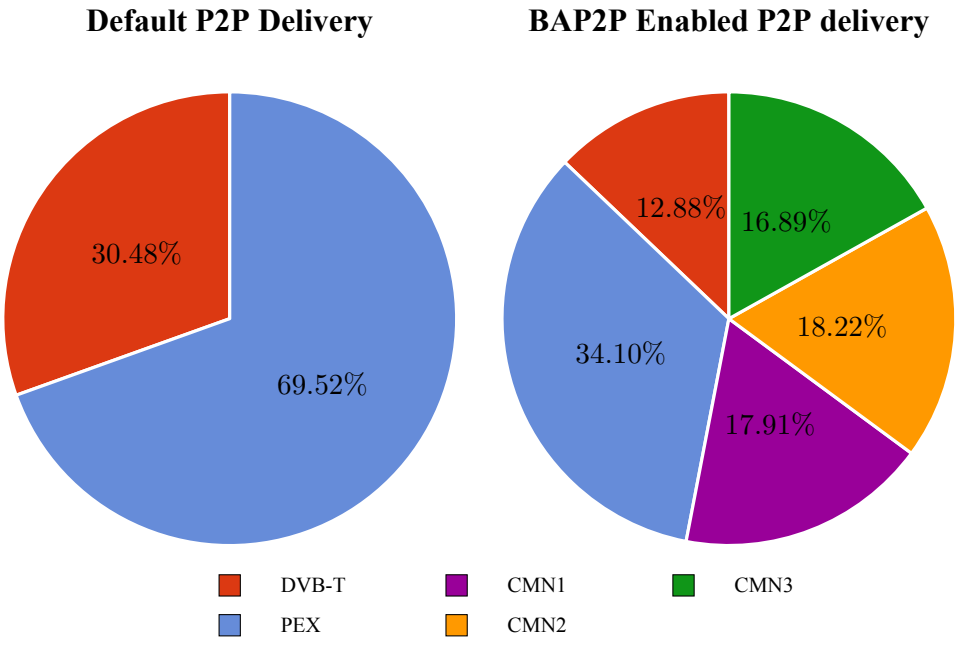
Στο τρίτο πειραματικό σενάριο υπάρχουν τέσσερις CMN. Στον CMN1 υπάρχει ένας χρήστης όπου έχει το περιεχόμενο. Στους CMN2, CMN3 και CMN4 υπάρχουν από 5 χρήστες όπου ζητάνε το περιεχόμενο.



Σχήμα 4.12: 15 χρήστες - Σύνθεση δικτύου

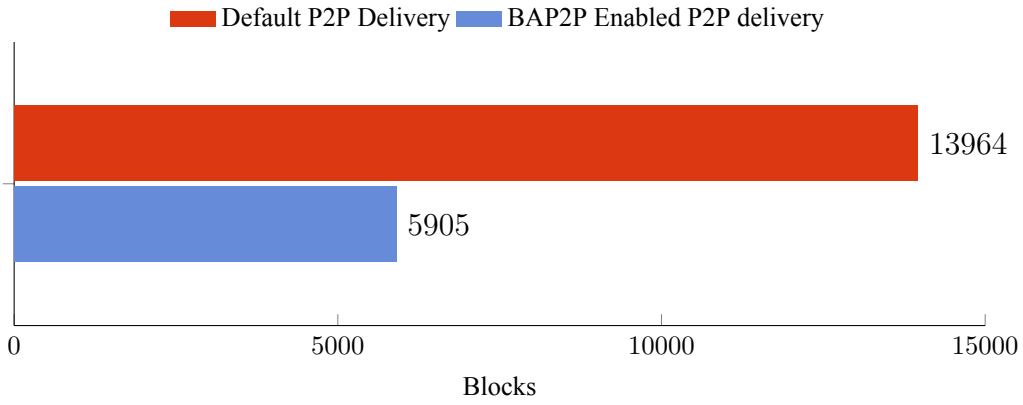
4.1.3.1 Αποτελέσματα

Στο παρακάτω σχήμα φαίνεται το ποσοστό χρησιμοποίησης όλων των καναλιών κατά την παράδοση του P2P περιεχομένου σε 15 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Είναι εμφανές πως υπάρχει σημαντική μείωση στην χρησιμοποίηση του DVB-T καναλιού.



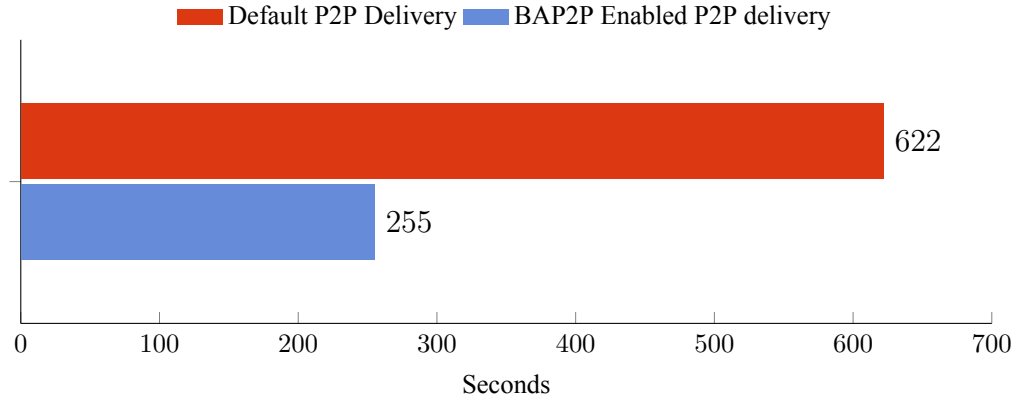
Σχήμα 4.13: 15 χρήστες - Ποσοστό χρησιμοποίησης κατά την παράδοση P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των Blocks που πέρασαν από το DVB-T κανάλι κατά την παράδοση του P2P περιεχομένου σε 15 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον αριθμό των Blocks που περνάνε από το DVB-T καθώς το μεγαλύτερο μέρος τους εξυπηρετείτε τοπικά από τον μηχανισμό.



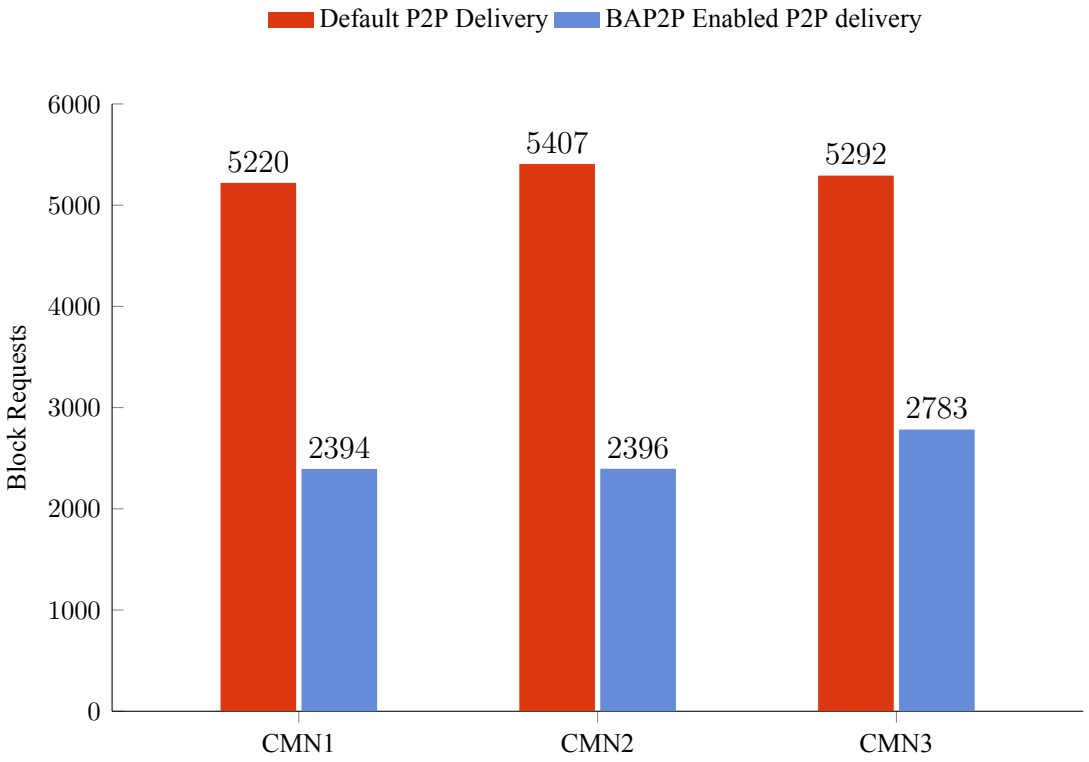
Σχήμα 4.14: 15 χρήστες - Blocks που πέρασαν απο το DVB-T

Στο παρακάτω σχήμα φαίνεται ο χρόνος ολοκλήρωσης της παράδοσης του P2P περιεχομένου σε 15 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον χρόνο παράδοσης του περιεχομένου καθώς κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T.



Σχήμα 4.15: 15 χρήστες - Διάρκεια ολοκλήρωσης της παράδοσης P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των αιτήματα για Blocks που πέρασαν από το DVB-T κανάλι ανά γειτονιά, κατά την παράδοση του P2P περιεχομένου σε 15 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο αριθμός των αιτημάτων έχει μειωθεί σημαντικά καθώς ο μηχανισμός κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T και έτσι τα Blocks που ζητάει έξω από την γειτονία του είναι πολύ λιγότερα.



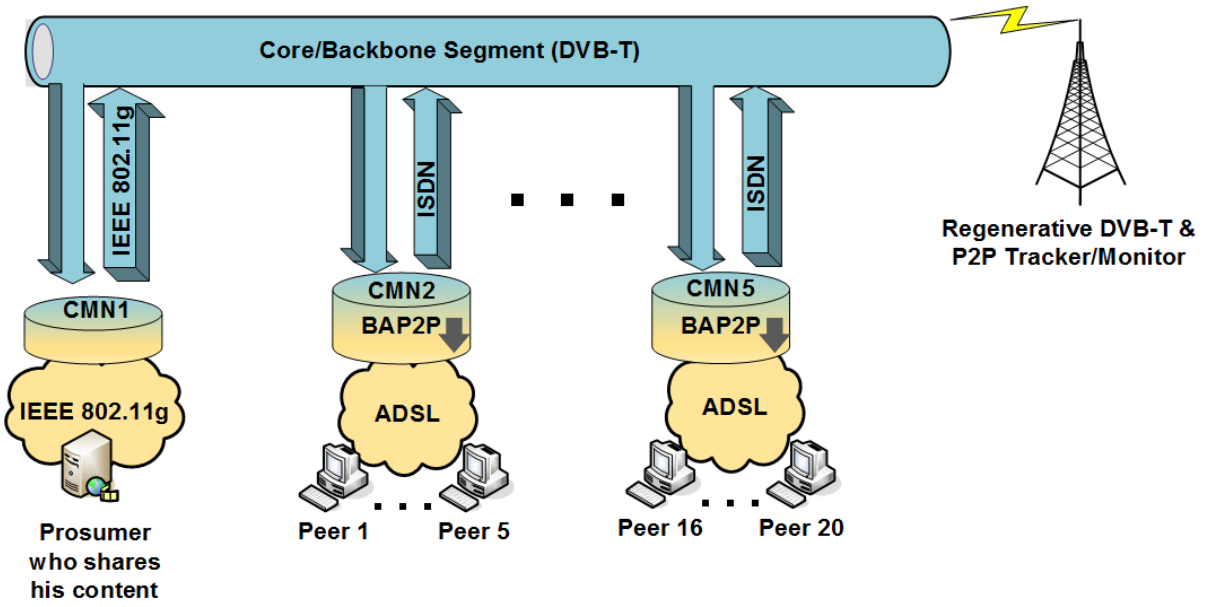
Σχήμα 4.16: 15 χρήστες - Αιτήματα για Blocks που περνάνε από το DVB-T

4.1.3.2 Αξιολόγηση

Όπως προκύπτει από τα αποτελέσματα ο μηχανισμός κατάφερε να ελαττώσει την χρησιμοποίηση του DVB-T δικτύου κατά 17,6%. Πιο συγκεκριμένα το 53,02% του περιεχομένου εξυπηρετήθηκε από τον μηχανισμό μέσω του CMN2, CMN3 και CMN4 μειώνοντας τον αριθμό των Blocks που πέρασαν από το DVB-T κατά 8059 blocks. Τέλος ο χρόνος ολοκλήρωσης της παράδοσης βελτιώθηκε κατά 367 δευτερόλεπτα.

4.1.4 Σενάριο 4 (20 χρήστες)

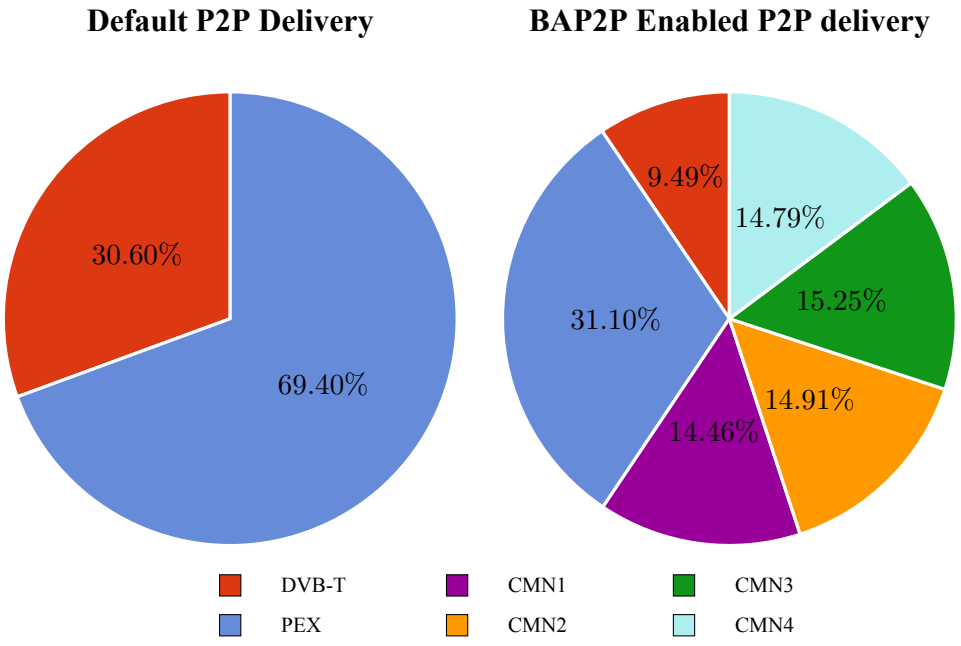
Στο τρίτο πειραματικό σενάριο υπάρχουν πέντε CMN. Στον CMN1 υπάρχει ένας χρήστης όπου έχει το περιεχόμενο. Στους CMN2, CMN3, CMN4 και CMN5 υπάρχουν από 5 χρήστες όπου ζητάνε το περιεχόμενο.



Σχήμα 4.17: 20 χρήστες - Σύνθεση δικτύου

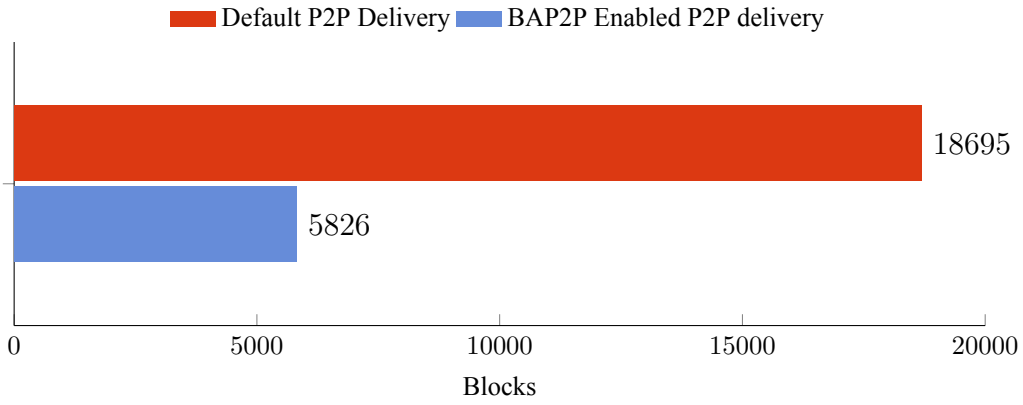
4.1.4.1 Αποτελέσματα

Στο παρακάτω σχήμα φαίνεται το ποσοστό χρησιμοποίησης όλων των καναλιών κατά την παράδοση του P2P περιεχομένου σε 20 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Είναι εμφανές πως υπάρχει σημαντική μείωση στην χρησιμοποίηση του DVB-T καναλιού.



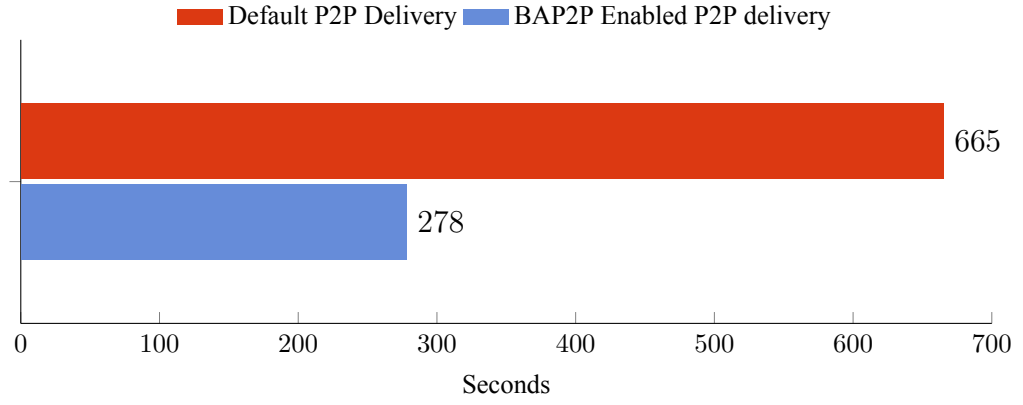
Σχήμα 4.18: 20 χρήστες - Ποσοστό χρησιμοποίησης κατά την παράδοση P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των Blocks που πέρασαν από το DVB-T κανάλι κατά την παράδοση του P2P περιεχομένου σε 20 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον αριθμό των Blocks που περνάνε από το DVB-T καθώς το μεγαλύτερο μέρος τους εξυπηρετείτε τοπικά από τον μηχανισμό.



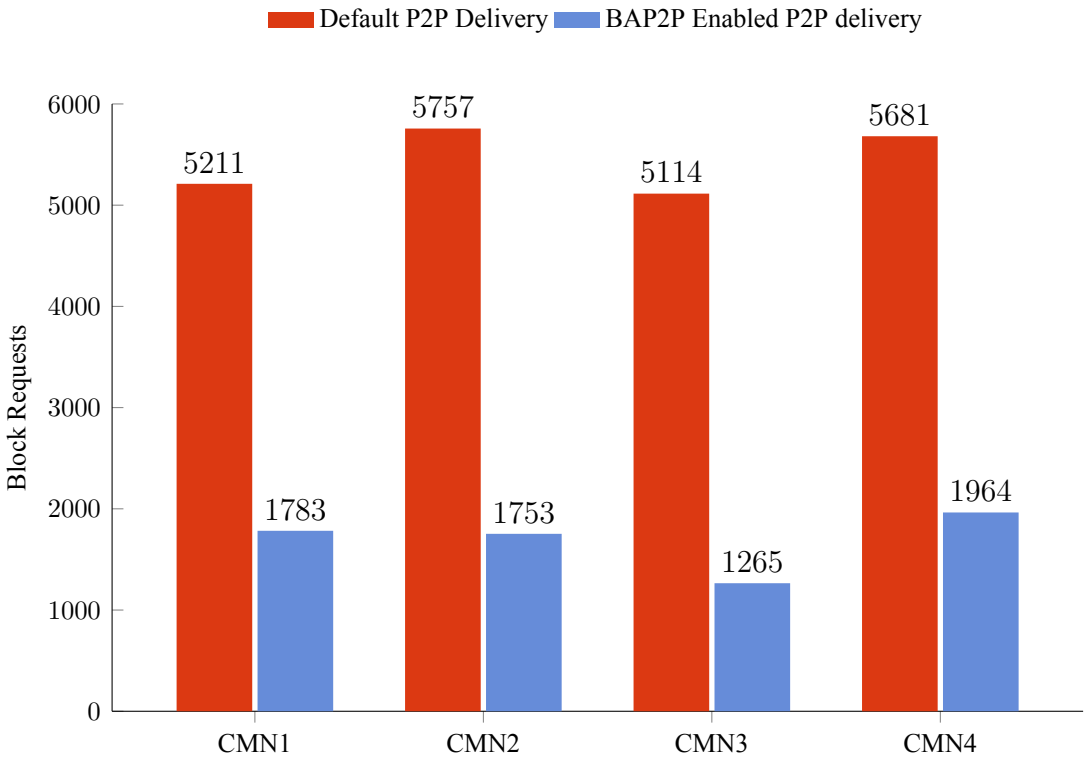
Σχήμα 4.19: 20 χρήστες - Blocks που πέρασαν απο το DVB-T

Στο παρακάτω σχήμα φαίνεται ο χρόνος ολοκλήρωσης της παράδοσης του P2P περιεχομένου σε 20 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο μηχανισμός έχει καταφέρει να μειώσει σημαντικά τον χρόνο παράδοσης του περιεχομένου καθώς κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T.



Σχήμα 4.20: 20 χρήστες - Διάρκεια ολοκλήρωσης της παράδοσης P2P περιεχομένου

Στο παρακάτω σχήμα φαίνεται ο αριθμός των αιτήματα για Blocks που πέρασαν από το DVB-T κανάλι ανά γειτονιά, κατά την παράδοση του P2P περιεχομένου σε 20 χρήστες, πριν και μετά την ενεργοποίηση του μηχανισμού. Ο αριθμός των αιτημάτων έχει μειωθεί σημαντικά καθώς ο μηχανισμός κάνει άμεσα διαθέσιμα σε όλους τοπικά, τα κομμάτια που περνάνε από το DVB-T και έτσι τα Blocks που ζητάει έξω από την γειτονία του είναι πολύ λιγότερα.



Σχήμα 4.21: 20 χρήστες - Αιτήματα για Blocks που περνάνε από το DVB-T

4.1.4.2 Αξιολόγηση

Όπως προκύπτει από τα αποτελέσματα ο μηχανισμός κατάφερε να ελαττώσει την χρησιμοποίηση του DVB-T δικτύου κατά 21,11%. Πιο συγκεκριμένα το 59,41% του περιεχομένου εξυπηρετήθηκε από τον μηχανισμό μέσω του CMN2, CMN3, CMN4 και CMN5 μειώνοντας τον αριθμό των Blocks που πέρασαν από το DVB-T κατά 12869 blocks. Τέλος ο χρόνος ολοκλήρωσης της παράδοσης βελτιώθηκε κατά 387 δευτερόλεπτα.

4.2 Συμπερασματα

Σε αυτό το κεφάλαιο έγινε η αξιολόγηση του πρότυπου μηχανισμού Broadcast-Aware Peer-to-Peer (δηλαδή BAP2P) ο οποίος επιτρέπει σε πανομοιότυπα κομμάτια περιεχομένου που είναι διαθέσιμα στον “αέρα” (στο DVB-T κανάλι) να λαμβάνονται ταυτόχρονα από τους CMNs που συμμετέχουν στο δίκτυο. Με αυτό τον τρόπο κάθε CMN λειτουργεί ως ένας κόμβος ανίχνευσης BitTorrent περιεχομένου ο οποίος είναι σε θέση να επεξεργάζεται την ροή δεδομένων του DVB-T αποσπόντα χρήσιμα κομμάτια περιεχομένου που μπορούν να αξιοποιηθούν από τον μηχανισμό και αποθηκεύονται προληπτικά σε αυτόν, σε τοπικό επίπεδο πριν το περιεχόμενο αιτηθεί από τους Peers. Όταν κάποιος Peer ζητήσει αυτά τα κομμάτια ο μηχανισμός είναι σε θέση να του τα προσφέρει τοπικά. Τα πειράματα αξιολόγησης του μηχανισμού έδειξαν πως με την χρήση του μηχανισμού BAP2P ένα torrent αρχείο θα μπορούσε να περάσει από το DVB-T κανάλι, το μέγιστο μέχρι 1,2 φορές. Η αύξηση των διασυνδεδεμένων peer δεν επηρεάζει την

απόδοση του μηχανισμού αντίθετα φαίνεται να έχει καλύτερα αποτελέσματα καθώς αυξάνεται ο αριθμός των διασυνδεδεμένων peer. Ο μηχανισμός καταφέρνει να περιορίσει το "βάρος" του φορτίου τόσο στο DVB-T κανάλι όσο και στα κανάλια ανόδου, μεγιστοποιόνταν έτσι την επεκτασιμότητα του συστήματος. Η χρήση του μηχανισμού BAP2P είναι μια καθολική λύση τόσο στο μοντέλο διαδραστικής τηλεόρασης DVB-T όσο και για τα περισσότερα συστήματα ευρυεκπομπής καθώς εκμεταλλεύεται την ευρείας κάλυψης φύση τους.

4.3 Μελλοντικές προτάσεις

Ένας άλλος τομέας για μελλοντική εργασία είναι η χρήση του μηχανισμού BAP2P ως μία καθολική λύση τόσο στα ευρείας κάλυψης δίκτυα όσο και στα ασύρματα δίκτυα, δεδομένου ότι και στις 2 περιπτώσεις η εκπομπή των δεδομένων δίνεται από ένα κεντρικό σημείο σε πολλούς προορισμούς όπου βρίσκονται οι χρήστες, έτσι ο μηχανισμός BAP2P θα μπορούσε να προσφέρει βοήθεια στην διανομή του περιεχομένου. Στην συνέχεια θα μπορούσε ο μηχανισμός BAP2P να ενσωματωθεί στις εφαρμογές BitTorrent και έτσι ο τελικός χρήστης που είναι συνδεδεμένος σε ένα ασύρματο δίκτυο να επωφελείται από τα δεδομένα που διαβιβάζονται σε άλλους χρήστες του δικτύου.

Κεφάλαιο 5

Παράρτημα

```python
import struct
import binascii

UTP_VERSION = 1

ST_DATA = 0
ST_FIN = 1
ST_STATE = 2
ST_RESET = 3
ST_SYN = 4

UTP_TYPE_NAMES = {
    ST_DATA: 'DATA',
    ST_FIN: 'FIN',
    ST_STATE: 'STATE',
    ST_RESET: 'RESET',
    ST_SYN: 'SYN',
}


def is_valid_utp(payload):
    if not payload or len(payload) < 20:
        return False

    version = struct.unpack("!B", payload[0:1])[0] & 0x0F
    type = struct.unpack("!B", payload[0:1])[0] >> 4

    if version != UTP_VERSION or not type in [0, 1, 2, 3, 4]:
        return False

    return True
```

```
32
33
34  class UTPPacketDecodeError(Exception): pass
35
36
37  class UTPPacketExtension():
38      def __init__(self, type, bitmask):
39          self.type = type
40          self.bitmask = bitmask
41
42      def __repr__(self):
43          return "<UTPPacketExtension type:%s length:%s bitmask:%s>" % (
44              self.type,
45              len(self.bitmask),
46              binascii.hexlify(self.bitmask)
47          )
48
49
50  class UTPPacket():
51      def __init__(self, payload=None):
52          self.__version = UTP_VERSION
53          self.type = 0
54          self.__extensions = []
55          self.connection_id = 0
56          self.timestamp_microseconds = 0
57          self.timestamp_difference_microseconds = 0
58          self.wnd_size = 0
59          self.seq_nr = 0
60          self.ack_nr = 0
61          self.data = []
62          if payload:
63              self.decode(payload)
64
65      def __repr__(self):
66          return "<UTPPacket version:%s type:%s connid:%s timestamp:%s timestamp_diff:%s" \
67                 " wnd_size:%s seq_nr:%s ack_nr:%s exts_len:%s data_len:%s>" % (
68              self.__version,
69              UTP_TYPE_NAMES[self.type],
70              self.connection_id,
71              self.timestamp_microseconds,
72              self.timestamp_difference_microseconds,
73              self.wnd_size,
74              self.seq_nr,
75              self.ack_nr,
```

```
76              len(self.__extensions),
77              len(self.data)
78          )
79
80      def add_utp_packet_ext(self, utp_packet_ext):
81          if isinstance(utp_packet_ext, UTPPacketExtension):
82              self.__extensions.append(utp_packet_ext)
83
84      def get_utp_packet_ext(self):
85          return self.__extensions
86
87      def decode(self, payload):
88
89          if not is_valid_utp(payload):
90              raise UTPPacketDecodeError("Invalid or Malformed UTP Packet")
91
92          # protocol version (4 low bits)
93          self.__version = struct.unpack("!B", payload[0:1])[0] & 0x0F
94          # packet_type (4 high bits)
95          self.type = struct.unpack("!B", payload[0:1])[0] >> 4
96          self.connection_id = struct.unpack("!H", payload[2:4])[0]
97          self.timestamp_microseconds = struct.unpack("!L", payload[4:8])[0]
98          self.timestamp_difference_microseconds = struct.unpack("!L", payload[8:12])[0]
99          self.wnd_size = struct.unpack("!I", payload[12:16])[0]
100         self.seq_nr = struct.unpack("!H", payload[16:18])[0]
101         self.ack_nr = struct.unpack("!H", payload[18:20])[0]
102
103         #extract extensions
104         next_extension_type = struct.unpack("!B", payload[1:2])[0]
105         data_offset = 20
106         while next_extension_type != 0:
107             extension_type = next_extension_type
108             extension_length = struct.unpack("!B", payload[data_offset + 1:data_offset + 2])[0]
109             extension_bitmask = payload[data_offset + 2:data_offset + (extension_length + 2)]
110             self.__extensions.append(UTPPacketExtension(extension_type, extension_bitmask))
111
112             next_extension_type = struct.unpack("!B", payload[data_offset:data_offset + 1])[0]
113             data_offset += (extension_length + 2)
114
115         self.data = payload[data_offset:]
116
117     def get_stream(self):
118         bytes = struct.pack("!B", self.__version & 0xf | self.type << 4)
119
```

```
        extension_bytes = ''
        number_of_extensions = len(self.__extensions)

        if number_of_extensions > 0:

            next_extension_type = self.__extensions[0].type
            bytes += struct.pack("!B", next_extension_type)
            next_ext = 1
            while True:

                if next_ext < number_of_extensions:
                    next_extension_type = self.__extensions[next_ext].type
                else:
                    next_extension_type = 0

                extension_bytes += struct.pack("!B", next_extension_type)
                extension_bytes += struct.pack("!B", len(self.__extensions[next_ext - 1].bitmask))
                extension_bytes += self.__extensions[next_ext - 1].bitmask
                next_ext += 1

                if next_extension_type == 0:
                    break
        else:
            bytes += struct.pack("!B", 0)

        bytes += struct.pack("!H", self.connection_id)
        bytes += struct.pack("!L", self.timestamp_microseconds)
        bytes += struct.pack("!L", self.timestamp_difference_microseconds)
        bytes += struct.pack("!I", self.wnd_size)
        bytes += struct.pack("!H", self.seq_nr)
        bytes += struct.pack("!H", self.ack_nr)
        bytes += extension_bytes
        if self.type == ST_DATA:
            if isinstance(self.data, str):
                bytes += ''.join(self.data)
            else:
                bytes += self.data
        return bytes
```

```
import pcapy
import dpkt
import logging
from threading import Thread
import socket
import re
```

```
import binascii
import stats

from tools import HOSTNAME_NUMBER

block_re = re.compile('0000400907(?P<index>\w{8})(?P<offset>\w{8})')

clients = [6, 14, 22, 30, 38]
cmns = [1, 2, 3, 4, 5]

local_client_ips = []
remote_cmn_subsets = []

for c in clients:
    local_client_ips.append('172.16.%s0.%s' % (HOSTNAME_NUMBER, c))

for c in cmns:
    if c != HOSTNAME_NUMBER:
        remote_cmn_subsets.append('172.16.%s0.' % c)

cacher_ip = '10.0.67.%s0' % HOSTNAME_NUMBER
server_ip = '192.168.50.3'


def is_local_peer(ip):
    if ip in local_client_ips:
        return True
    return False


def is_remote_cmn(ip):
    if ip[:10] in remote_cmn_subsets:
        return True
    return False


def is_server(ip):
    if ip == server_ip:
        return True
    return False


def is_cacher(ip):
    if ip == cacher_ip:
```

```
51          return True
52      return False
53
54
55  class InternalTrafficSniffer(Thread):
56      def __init__(self, dev='eth1'):
57          Thread.__init__(self)
58
59          #self.expr = 'udp and dst port %s' % port
60          self.expr = 'udp or tcp'
61          self.dev = dev
62          #self.expr = 'tcp'
63
64          self.maxlen = 65535  # max size of packet to capture
65          self.promiscuous = 1  # promiscuous mode?
66          self.read_timeout = 100  # in milliseconds
67          self.max_pkts = -1  # number of packets to capture; -1 => no limit
68
69          self.active = True
70          self.p = pcapy.open_live(dev, self.maxlen, self.promiscuous, self.read_timeout)
71          self.p.setfilter(self.expr)
72          logging.info('Piece Sniffer started...')
73          logging.debug('Pcap Filter:  \"%s\"' % self.expr)
74
75      def internal_block_stats(self, src, dst):
76
77          #skip outgoing client self traffic (count only cmn exchange outgoing blocks)
78          if is_local_peer(src) and src.endswith('.6') and self.dev == 'eth1.1':
79              if is_remote_cmn(dst): stats.in_cl1['cmn_ex_out'] += 1
80              return
81
82          if is_local_peer(src) and src.endswith('.14') and self.dev == 'eth1.2':
83              if is_remote_cmn(dst): stats.in_cl2['cmn_ex_out'] += 1
84              return
85
86          if is_local_peer(src) and src.endswith('.22') and self.dev == 'eth1.3':
87              if is_remote_cmn(dst): stats.in_cl3['cmn_ex_out'] += 1
88              return
89
90          if is_local_peer(src) and src.endswith('.30') and self.dev == 'eth1.4':
91              if is_remote_cmn(dst): stats.in_cl4['cmn_ex_out'] += 1
92              return
93
94          if is_local_peer(src) and src.endswith('.38') and self.dev == 'eth1.5':
```

```
95              if is_remote_cmn(dst): stats.in_cl5['cmn_ex_out'] += 1
96              return
97
98          # if destination is local peer
99          if is_local_peer(dst):
100
101             # if source is local peer
102             if is_local_peer(src):
103
104                 # if dst is client 1
105                 if str(dst).endswith('.6'):
106                     if str(src).endswith('.14'): stats.in_cl1['cl2'] += 1
107                     elif str(src).endswith('.22'): stats.in_cl1['cl3'] += 1
108                     elif str(src).endswith('.30'): stats.in_cl1['cl4'] += 1
109                     elif str(src).endswith('.38'): stats.in_cl1['cl5'] += 1
110
111                 # if dst is client 2
112                 if str(dst).endswith('.14'):
113                     if str(src).endswith('.6'): stats.in_cl2['cl1'] += 1
114                     elif str(src).endswith('.22'): stats.in_cl2['cl3'] += 1
115                     elif str(src).endswith('.30'): stats.in_cl2['cl4'] += 1
116                     elif str(src).endswith('.38'): stats.in_cl2['cl5'] += 1
117
118                 # if dst is client 3
119                 if str(dst).endswith('.22'):
120                     if str(src).endswith('.6'): stats.in_cl3['cl1'] += 1
121                     elif str(src).endswith('.14'): stats.in_cl3['cl2'] += 1
122                     elif str(src).endswith('.30'): stats.in_cl3['cl4'] += 1
123                     elif str(src).endswith('.38'): stats.in_cl3['cl5'] += 1
124
125                 # if dst is client 4
126                 if str(dst).endswith('.30'):
127                     if str(src).endswith('.6'): stats.in_cl4['cl1'] += 1
128                     elif str(src).endswith('.14'): stats.in_cl4['cl2'] += 1
129                     elif str(src).endswith('.22'): stats.in_cl4['cl3'] += 1
130                     elif str(src).endswith('.38'): stats.in_cl4['cl5'] += 1
131
132                 # if dst is client 5
133                 if str(dst).endswith('.38'):
134                     if str(src).endswith('.6'): stats.in_cl5['cl1'] += 1
135                     elif str(src).endswith('.14'): stats.in_cl5['cl2'] += 1
136                     elif str(src).endswith('.22'): stats.in_cl5['cl3'] += 1
137                     elif str(src).endswith('.30'): stats.in_cl5['cl4'] += 1
138
```

```
139             # if source is server
140             if is_server(src):
141                 if str(dst).endswith('.6'): stats.in_cl1['srv'] += 1
142                 elif str(dst).endswith('.14'): stats.in_cl2['srv'] += 1
143                 elif str(dst).endswith('.22'): stats.in_cl3['srv'] += 1
144                 elif str(dst).endswith('.30'): stats.in_cl4['srv'] += 1
145                 elif str(dst).endswith('.38'): stats.in_cl5['srv'] += 1
146 
147             # if source is remote cmn
148             if is_remote_cmn(src):
149                 if str(dst).endswith('.6'): stats.in_cl1['cmn_ex_in'] += 1
150                 elif str(dst).endswith('.14'): stats.in_cl2['cmn_ex_in'] += 1
151                 elif str(dst).endswith('.22'): stats.in_cl3['cmn_ex_in'] += 1
152                 elif str(dst).endswith('.30'): stats.in_cl4['cmn_ex_in'] += 1
153                 elif str(dst).endswith('.38'): stats.in_cl5['cmn_ex_in'] += 1
154 
155             # if source is cacher
156             if is_cacher(src):
157                 if str(dst).endswith('.6'): stats.in_cl1['cache'] += 1
158                 elif str(dst).endswith('.14'): stats.in_cl2['cache'] += 1
159                 elif str(dst).endswith('.22'): stats.in_cl3['cache'] += 1
160                 elif str(dst).endswith('.30'): stats.in_cl4['cache'] += 1
161                 elif str(dst).endswith('.38'): stats.in_cl5['cache'] += 1
162 
163     #@staticmethod
164     def cb(self, hdr, data):
165 
166         if not data:
167             return
168 
169         eth = dpkt.ethernet.Ethernet(str(data))
170 
171         #ip validation
172         if eth.type != dpkt.ethernet.ETH_TYPE_IP:
173             return
174 
175         ip = eth.data
176 
177         try:
178             if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
179                 return
180         except:
181             return
182 
```

```
        #tcp udp validation
        #if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
        # return

        src_ip = socket.inet_ntoa(ip.src)
        dst_ip = socket.inet_ntoa(ip.dst)

        if ip.p == dpkt.ip.IP_PROTO_TCP:
            tcp = ip.data

            try:
                stats.bitrate_current += len(tcp.data)
            except:
                return

            for m in block_re.finditer(binascii.hexlify(tcp.data)):
                self.internal_block_stats(src_ip, dst_ip)

    def stop(self):
        logging.info('Piece Sniffer stopped...')
        self.active = False

    def run(self):
        while self.active:
            self.p.dispatch(0, self.cb)
```

```
import pcapy
import dpkt
import logging
from btcacher import PieceManager
from threading import Thread
import socket
import re
import binascii
import stats

block_re = re.compile('0000400907(?P<index>\w{8})(?P<offset>\w{8})')

from tools import HOSTNAME_NUMBER

class PieceSniffer(Thread):
    def __init__(self, dev='dvb0_0'):
        Thread.__init__(self)

        #self.expr = 'udp and dst port %s' % port
```

```
20          self.expr = 'udp or tcp'
21          #self.expr = 'tcp'
22
23          self.maxlen = 65535  # max size of packet to capture
24          self.promiscuous = 1  # promiscuous mode?
25          self.read_timeout = 100  # in milliseconds
26          self.max_pkts = -1  # number of packets to capture; -1 => no limit
27
28          self.active = True
29          self.p = pcapy.open_live(dev, self.maxlen, self.promiscuous, self.read_timeout)
30          self.p.setfilter(self.expr)
31          logging.info('Piece Sniffer started...')
32          logging.debug('Pcap Filter:  \"%s\"' % self.expr)
33
34      @staticmethod
35      def cb(hdr, data):
36
37          if not data:
38              return
39
40          eth = dpkt.ethernet.Ethernet(str(data))
41
42          #ip validation
43          if eth.type != dpkt.ethernet.ETH_TYPE_IP:
44              return
45
46          ip = eth.data
47
48          try:
49              if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
50                  return
51          except:
52              return
53
54          #tcp udp validation
55          #if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
56          # return
57
58          src_ip = socket.inet_ntoa(ip.src)
59          dst_ip = socket.inet_ntoa(ip.dst)
60
61          if ip.p == dpkt.ip.IP_PROTO_TCP:
62              tcp = ip.data
63
```

```
            try:
                stats.bitrate_current += len(tcp.data)
            except:
                return

            for m in block_re.finditer(binascii.hexlify(tcp.data)):
                stats.blocks_count += 1

                block = m.groups()
                block_index = int(block[0], 16)
                block_offset = int(block[1], 16)

                #print 'block %s %s' %  (block_index, block_offset)
                #PieceSniffer.dvbt_block_stats(src_ip, dst_ip)
                PieceManager.add_block(block_index, block_offset, '')
                #PieceManager.print_stats()

    def stop(self):
        logging.info('Piece Sniffer stopped...')
        self.active = False

    def run(self):
        while self.active:
            self.p.dispatch(0, PieceSniffer.cb)
```

```
import pcapy
import dpkt
import logging
from btcacher import PieceManager
from threading import Thread
import re
import binascii
import stats

piece_request_re = re.compile('0000000d06(?P<index>\w{8})(?P<offset>\w{8})(?P<length>\w{8})')


class PieceRequestSniffer(Thread):
    def __init__(self, dev='eth0'):
        Thread.__init__(self)

        self.expr = 'udp or tcp'

        self.maxlen = 65535  # max size of packet to capture
        self.promiscuous = 1  # promiscuous mode?
```

```
        self.read_timeout = 100  # in milliseconds
        self.max_pkts = -1  # number of packets to capture; -1 => no limit

        self.active = True
        self.p = pcapy.open_live(dev, self.maxlen, self.promiscuous, self.read_timeout)
        self.p.setfilter(self.expr)
        logging.info('Piece Request Sniffer started...')
        logging.debug('Pcap Filter:  \"%s\"' % self.expr)

    @staticmethod
    def cb(hdr, data):

        if not data:
            return

        eth = dpkt.ethernet.Ethernet(str(data))

        #ip validation
        if eth.type != dpkt.ethernet.ETH_TYPE_IP:
            return

        ip = eth.data

        try:
            if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
                return
        except:
            return

        if ip.p == dpkt.ip.IP_PROTO_TCP:
            tcp = ip.data

            try:
                hex_data = binascii.hexlify(tcp.data)
            except:
                return
            request_blocks = piece_request_re.findall(hex_data)
            if len(request_blocks) == 0:
                return

            for block in request_blocks:
                stats.block_requests += 1
                block_index = int(block[0], 16)
                block_offset = int(block[1], 16)
```

```
                length = int(block[2], 16)

                is_cached = PieceManager.do_i_have_block(block_index, block_offset)
                if is_cached:
                    stats.block_requests_already_have += 1

        if ip.p == dpkt.ip.IP_PROTO_UDP:
            udp = ip.data

            #btutp validation
            if not btutp.is_btutp(udp.data):
                return

            #utp_packet = btutp.btutp_decoder(udp.data)
            pkt = BtUtpPacket()
            pkt.decode(udp.data)

            is_cached = PieceManager.do_i_have_block(pkt.block_index, pkt.block_offset)
            if is_cached:
                stats.block_requests_already_have += 1

    def stop(self):
        logging.info('Piece Request Sniffer stopped...')
        self.active = False

    def run(self):
        while self.active:
            self.p.dispatch(0, PieceRequestSniffer.cb)
```

```
import pcapy
import dpkt
import logging
from btcacher import PieceManager
from threading import Thread
import socket
import re
import binascii
import stats

block_re = re.compile('0000400907(?P<index>\w{8})(?P<offset>\w{8})')

from tools import HOSTNAME_NUMBER

class PieceSniffer(Thread):
    def __init__(self, dev='dvb0_0'):
```

```
        Thread.__init__(self)

        #self.expr = 'udp and dst port %s' % port
        self.expr = 'udp or tcp'
        #self.expr = 'tcp'

        self.maxlen = 65535  # max size of packet to capture
        self.promiscuous = 1  # promiscuous mode?
        self.read_timeout = 100  # in milliseconds
        self.max_pkts = -1  # number of packets to capture; -1 => no limit

        self.active = True
        self.p = pcapy.open_live(dev, self.maxlen, self.promiscuous, self.read_timeout)
        self.p.setfilter(self.expr)
        logging.info('Piece Sniffer started...')
        logging.debug('Pcap Filter:  \"%s\"' % self.expr)


    @staticmethod
    def cb(hdr, data):

        if not data:
            return

        eth = dpkt.ethernet.Ethernet(str(data))

        #ip validation
        if eth.type != dpkt.ethernet.ETH_TYPE_IP:
            return

        ip = eth.data

        try:
            if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
                return
        except:
            return

        #tcp udp validation
        #if ip.p != dpkt.ip.IP_PROTO_UDP and ip.p != dpkt.ip.IP_PROTO_TCP:
        # return

        src_ip = socket.inet_ntoa(ip.src)
        dst_ip = socket.inet_ntoa(ip.dst)
```

```
61
62          if ip.p == dpkt.ip.IP_PROTO_TCP:
63              tcp = ip.data
64
65              try:
66                  stats.bitrate_current += len(tcp.data)
67              except:
68                  return
69
70              for m in block_re.finditer(binascii.hexlify(tcp.data)):
71                  stats.blocks_count += 1
72
73                  block = m.groups()
74                  block_index = int(block[0], 16)
75                  block_offset = int(block[1], 16)
76
77                  #print 'block %s %s' %  (block_index, block_offset)
78                  #PieceSniffer.dvbt_block_stats(src_ip, dst_ip)
79                  PieceManager.add_block(block_index, block_offset, '')
80                  #PieceManager.print_stats()
81
82          #udp validation
83          if ip.p == dpkt.ip.IP_PROTO_UDP:
84              udp = ip.data
85
86              #btutp validation
87              if not btutp.is_btutp(udp.data):
88                  return
89
90              pkt = BtUtpPacket()
91              pkt.decode(udp.data)
92
93              #data type only
94              if pkt.type == 0:
95                  PieceManager.packetManipulate(pkt)
96
97      def stop(self):
98          logging.info('Piece Sniffer stopped...')
99          self.active = False
100
101     def run(self):
102         while self.active:
103             self.p.dispatch(0, PieceSniffer.cb)
```

```
import stats

NEW_PIECES_HAVE = []

class PieceManager():
    __pieces = []

    def __init__(self):
        pass

    @staticmethod
    def add_block(index, offset, data):
        piece = PieceManager.__get_piece(index)
        if piece:
            if not piece.is_completed() and not PieceManager.__block_exists(piece, offset):
                piece.add_block(Block(offset, data))
                if piece.is_completed():
                    NEW_PIECES_HAVE.append(index)
        else:
            new_piece = Piece(index)
            new_piece.add_block(Block(offset, data))
            PieceManager.__pieces.append(new_piece)

    @staticmethod
    def __get_piece(index):
        for p in PieceManager.__pieces:
            if p.index == index:
                return p
        return None

    @staticmethod
    def do_i_have_block(index, offset):
        for p in PieceManager.__pieces:
            if p.index == index:
                for block in p.blocks:
                    if block.offset == offset:
                        return True
                break
        return False

    @staticmethod
    def do_i_have(index):
        for p in PieceManager.__pieces:
            if p.index == index and p.is_completed():
```

```
45                  return True
46          return False
47
48      @staticmethod
49      def sort_pieces():
50          PieceManager.__pieces.sort(key=lambda x: x.index)
51
52      @staticmethod
53      def __block_exists(piece, offset):
54          for block in piece.blocks:
55              if block.offset == offset:
56                  return True
57          return False
58
59
60  class Piece():
61      def __init__(self, index):
62          self.index = index
63          self.blocks = []
64
65      def add_block(self, block):
66          stats.cached_blocks_count += 1
67          self.blocks.append(block)
68          #sort blocks if completed
69          if self.is_completed():
70              self.blocks.sort(key=lambda x: x.offset)
71
72      def is_completed(self):
73          if len(self.blocks) == 2:
74              return True
75          return False
76
77
78
79  class Block():
80      def __init__(self, offset, data):
81          self.offset = offset
82          self.data = data
```

Βιβλιογραφία

[1] G. Mastorakis V. Zacharopoulos E. Pallis, C. Mantakas. Digital switchover in uhf: the athena concept for broadband access. 2005. URL http://www.eurasip.org/Proceedings/Ext/IST05/papers/110.pdf.

[2] C. Skianis V. Zacharopoulos E. Markakis, E. Pallis. Optimised network resource exploitation in interactive broadcasting environments via p2p constellations. 2012. URL http://www.ict-alicante.eu/validation/download/papers/t4.1_icnc12.pdf.

[3] S. MJ Dempsey J. Fiedler K. Koutsopoulos E. Markakis S. Garvey N. Pereira S. Denazis S. Tombros E. Pallis O. Koufopavlou . Christakidis, N. Efthymiopoulos. Integrating peer-to-peer with next generation networks. 2011. URL http://www.ict-vitalpp.upatras.gr/pdf/papers/Integrating%20P2P%20with%20Next%20Generation%20Networks.pdf.

[4] ETSI EN 300 744. Digital video broadcasting (dvb); framing structure, channel coding and modulation for digital terrestrial television. 2009. URL http://www.etsi.org/deliver/etsi_en/300700_300799/300744/01.06.01_60/en_300744v010601p.pdf.

[5] ETSI EN 301 192. Digital video broadcasting (dvb); dvb specification for data broadcasting. 2006. URL http://www.etsi.org/deliver/etsi_en/301100_301199/301192/01.04.01_40/en_301192v010401o.pdf.

[6] J.C. Rault. The coded orthogonal frequency division multiplexing (cofdm) technique, and its application to digital radio broadcasting towards mobile receivers. 1989. URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=64008.

[7] U. Reimers. Dvb-t: the cofdm-based system for terrestrial television. 1997. URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=581222.

[8] ETS 300 801. Digital video broadcasting (dvb); interaction channel through public switched telecommunications network (pstn) / integrated services digital networks (isdn). 1997. URL http://www.etsi.org/deliver/etsi_i_ets/300800_300899/300801/01_60/ets_300801e01p.pdf.

[9] T. Ahmed I. Djama. A cross-layer interworking of dvb-t and wlan for mobile iptv service delivery. 2007. URL http://www.ict-alicante.eu/validation/download/papers/t4.1_icnc12.pdf.

[10] H. Zimmermann. Osi reference model – the iso model of architecture for open systems interconnection. 28:425–432, April 1980. URL http://www.comsoc.org/livepubs/50_journals/pdf/RightsManagement_eid=136833.pdf.

[11] J. Postel. Rfc791 internet protocol. September 1981. URL https://www.ietf.org/rfc/rfc791.txt.

[12] A. Norberg. utp utorrent transmission protocol. 2009. URL http://www.bittorrent.org/beps/bep_0029.html.

[13] G. Camarillo. Rfc5694 peer-to-peer (p2p) architecture: Definition, taxonomies, examples, and applicability. November 2009. URL http://tools.ietf.org/html/rfc5694.

[14] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, 2004. URL http://dl.acm.org/citation.cfm?id=1041681.

[15] Bram Cohen. The bittorrent protocol specification. 2008. URL http://www.bittorrent.org/beps/bep_0003.html.

[16] Andrew Loewenstern and Arvid Norberg. Dht protocol. 2008. URL http://www.bittorrent.org/beps/bep_0005.html.

[17] pcapy. URL http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Pcapy.

[18] Autonomotorrent. URL https://github.com/abhijeeth/AutonomoTorrent.

[19] Transmission. URL https://www.transmissionbt.com/.

[20] Vuze. URL http://www.vuze.com/.