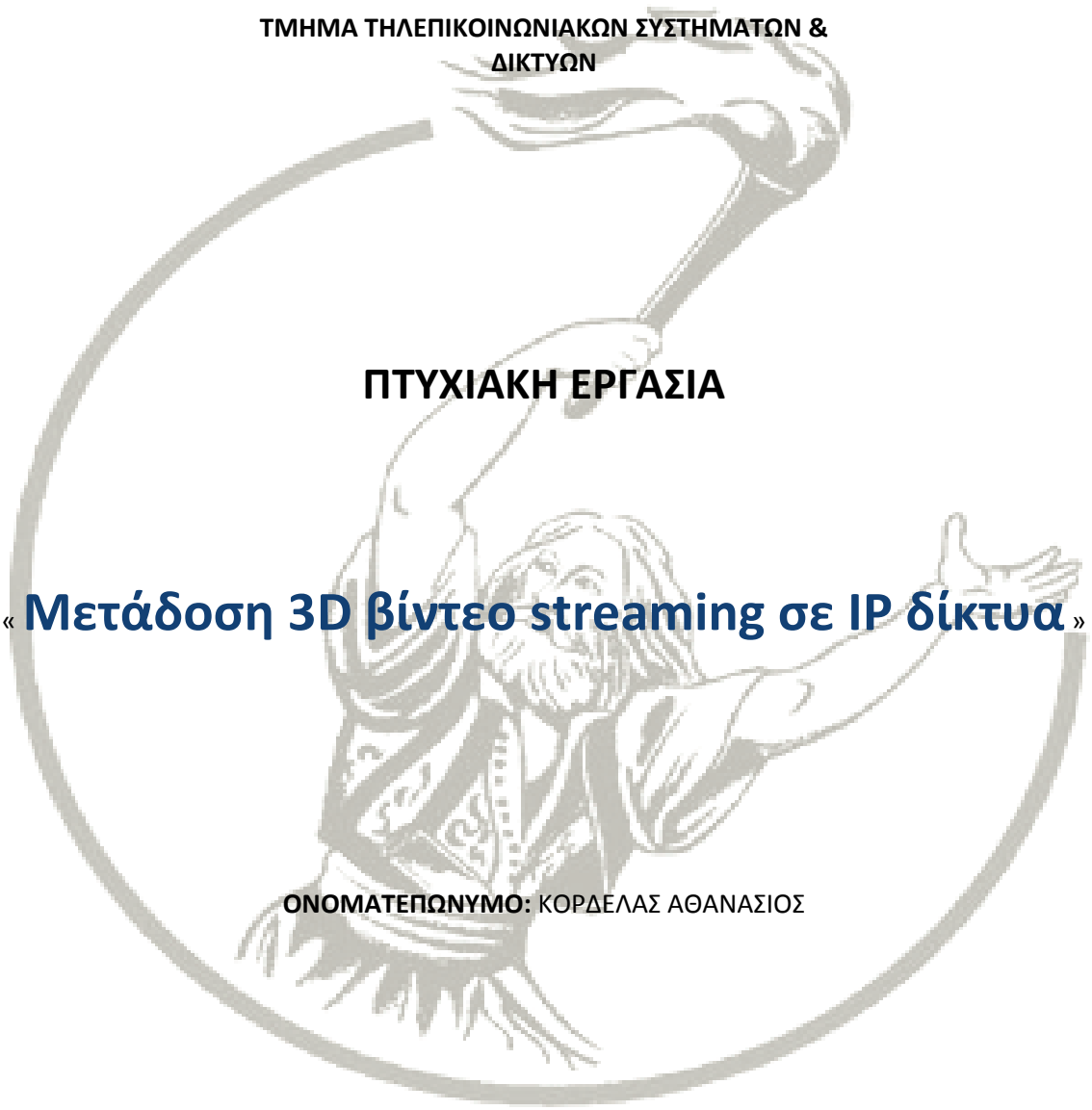
ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΜΕΣΟΛΟΓΓΙΟΥ

ΤΜΗΜΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ & ΔΙΚΤΥΩΝ



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

« Μετάδοση 3D βίντεο streaming σε IP δίκτυα »

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΚΟΡΔΕΛΑΣ ΑΘΑΝΑΣΙΟΣ

ΕΠΙΒΛΕΠΩΝ: ΤΑΣΟΣ ΝΤΑΓΙΟΥΚΛΑΣ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Ναύπακτος, ……/……/2012

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ:

1. Τάσος Νταγιούκλας
2. Βασίλειος Τριανταφύλλου
3. Παναγιώτης Γαλιώτος

Περίληψη

Τα τελευταία χρόνια υπάρχει αυξημένη ερευνητική δραστηριότητα γύρω από το 3D βίντεο. Όπως φαίνεται έχει φτάσει η εποχή όπου το 3D βίντεο θα γίνει ένας σύνηθες τρόπος ψυχαγωγίας μέσω της τρισδιάστατης τηλεόρασης και της τρισδιάστατης κωδικοποίησης. Στον χώρο της πληροφορικής γίνονται συνεχής προσπάθειες επίτευξης αυτού του στόχου, ο οποίος μπορεί να επιτευχθεί μόνο μέσω της συνεχής έρευνας, κάτι το οποίο επιδιώκεται στην συγκεκριμένη πτυχιακή.

Για την επίτευξη αυτού του στόχου, πρωταρχικά κατανοήθηκε η δομή των κωδικοποιημένων στερεοσκοπικών ακολουθιών, χρησιμοποιώντας το πρότυπο H.264/MVC Format, ενώ παράλληλα έγινε μελέτη του προτύπου του RTP για το H.264/MVC. Στην συνέχεια κατασκευάστηκε ένα εργαλείο δημιουργίας RTP πακέτων (MVC MST RTP Packetizer) κάνοντας χρήση όλων των δυνατών δομών ωφέλιμών φορτίων του RTP για το MVC και το αντίστοιχο εργαλείο μετατροπής των RTP πακέτων στην αρχική κωδικοποιημένη ακολουθία (MVC MST RTP Depacketizer). Τέλος, για την αποστολή και την λήψη των 3D ή 2D ακολουθιών κατασκευάστηκαν δύο επιπλέον εργαλεία (MVC MST UDP/IP Streamer - Client).

Για την κωδικοποίηση - αποκωδικοποίηση των στερεοσκοπικών ακολουθιών, χρησιμοποιήθηκε ο H.264/MVC κωδικοποιητής - αποκωδικοποιητής ανοικτού κώδικα της Nokia, όπου εκτός των άλλων προορίζεται για την αποστολή και αναπαραγωγή MVC ακολουθιών από μελλοντικές κινητές συσκευές οι οποίες θα κάνουν χρήση στερεοσκοπικών οθονών. Ποιο συγκεκριμένα, οι MVC κωδικοποιημένες ακολουθίες μετατράπηκαν σε RTP πακέτα και μεταδόθηκαν μέσω UDP/IP , ενώ παράλληλα χρησιμοποιήθηκε ο εξομοιωτής ασυρμάτου δικτύου(Dummynet) για την μεταβολή της κατάστασης του δικτύου (απώλεια πακέτων). Τέλος, έγινε λήψη των ακολουθιών, αποκωδικοποιήθηκαν και συγκρίθηκαν με τα αρχικά video για τον υπολογισμό των PSNR, Overhead και τον αριθμό των ολοκληρωτικά χαμένων πλαισίων.

Abstract

In recent years there is an increased activity and research on 3D video. It seems that the time where the 3D video will become a standard way of entertainment through the 3D TV streaming has arrived. In the area of information technology, continuing efforts are done in order this goal to be achieved, but it can only be resolved through continuous research.

In order to meet this goal, the stereoscopic structure of coded bitstream H.264/MVC Format has been studied. Particular emphasis was given on the standard of RTP for H.264/MVC. Afterwards, an RTP packetizer has been developed (MVC MST RTP Packetizer) using all the RTP payload structures along with the corresponding 'MVC MST RTP Depacketizer'. Finishing, another two tools have been developed for the transmission of 3D or 2D sequences (MVC MST UDP/IP Server - Client).

The Nokia MVC encoder - decoder has been used. Additionally, RTP packets are created through coded sequences and transmitted via UDP / IP, using in parallel a network simulator "Dummynet" to generate packet loss. Thereafter, the transmitted decoded video sequences are compared with the original video sequences for the calculation of PSNR, and Overhead and the number of the lost frames.

Ευχαριστίες

Η περάτωση της συγκεκριμένης πτυχιακής δεν είναι αποτέλεσμα μόνο της προσωπικής μου εργασίας και προσπάθειας αλλά και της συνεργασίας μου με ανθρώπους που με βοήθησαν και με συμβούλευσαν ανελλιπώς.

Καταρχήν θέλω να ευχαριστήσω τον καθηγητή μου, κ. Τάσο Νταγιούκλα, για την συνεχή, άμεση , ουσιαστική και πάνω από όλα επιστημονική καθοδήγηση που μου προσέφερε. Η καθοδήγηση αυτή δεν αποτέλεσε μόνο καθοριστικό παράγοντα για την ολοκλήρωση της παρούσας πτυχιακής εργασίας αλλά μου προσέφερε τα εφόδια για την μετέπειτα πορεία μου.

Ευχαριστώ θερμά την οικογένεια μου και τους φίλους μου για την βοήθεια και την στήριξη που μου παρείχαν όλα αυτά τα χρόνια.

Πίνακας περιεχομένων

Κεφάλαιο 1ο. Εισαγωγή

Τα τελευταία χρόνια υπάρχει μεγάλη εξέλιξη τόσο στην ποιότητα όσο και στην μετάδοση βίντεο. Έχουμε διέλθει από την κάποτε αναλογικής μετάδοσης χαμηλής ανάλυσης βίντεο στην μετάδοση υψηλής ευκρίνειας βίντεο (HD) μέσω IP [19]. Η τεχνολογία εξελίσσεται και όλα δείχνουν πώς στο άμεσο μέλλον θα έχουμε την δυνατότητα να παρακολουθούμε ακόμα και από κινητά τηλέφωνα 3D βίντεο τα οποία θα μεταφέρονται μέσω ευρυζωνικών δικτύων.

Η αναφορά του όρου «μετάδοση 3D βίντεο» δηλώνει ένα μεγάλο εύρος εφαρμογών 3D:

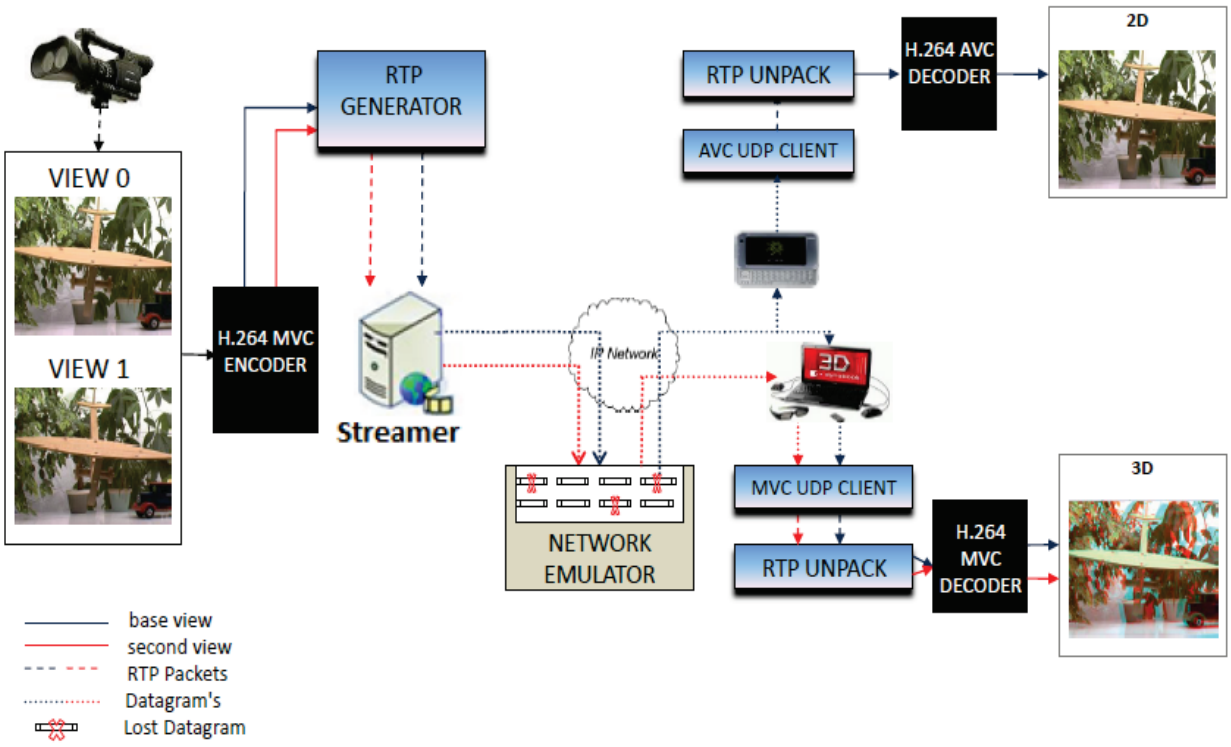
- 3D video streaming
- 3DTV
- 3D teleconference
- Free Viewpoint
- Telepresence ή Virtual Reality

Το Multi-View Video Coding(MVC) έχει αποκτήσει τεράστιο ενδιαφέρον πρόσφατα. Η ομάδα Joint Video (JVT) έχει πρόσφατα επεκτείνει το πρότυπο H.264 ώστε να είναι σε θέση να διαχειριστεί την κωδικοποίηση πολλαπλών όψεων (H.264 / MVC) [10],[3]. Το H.264 / MVC περιέχει αρκετές τεχνικές βελτίωσης της κωδικοποίησης και μείωσης της πολυπλοκότητας αποκωδικοποίησης ενώ παράλληλα έχει κληρονομήσει κάποιους από τους μηχανισμούς του Scalable Video Coding H.264/SVC και του Advance Video Coding H.264/AVC [15] , [18].

Σε αυτήν την εργασία, εκτός από την αποδοτικότητα της κωδικοποίησης, μελετάται η απόδοση του H.264/MVC streaming υπό διαφορετικές συνθήκες δικτύου(απώλεια πακέτων), υπό διαφορετικά μεγέθη πακέτων (Maximum Transmission Unit(MTU)) και υπό διαφορετικά είδη πακετοποίησης (αριθμός Network Abstraction Layer Units(NALUs) ανά πλαίσιο, δομές ωφέλιμου φορτίου του RTP). Τα διαφορετικά είδη πακετοποίησης συγκρίθηκαν σχετικά με το πλεονασμό τον οποίον εισάγουν, τις δυνατότητες που παρέχουν για την διαχείριση λαθών, την διάδοση λαθών λόγω εξάρτησης των όψεων και τέλος σχετικά με το αντιληπτό QoS των διαφορετικών όψεων βίντεο.

Η Εικόνα 1 απεικονίζει το συνολικό μοντέλο της παρούσας πτυχιακής. Διαφορετικές κάμερες υπό διαφορετικές γωνίες συνθέτουν βίντεο πολλαπλών όψεων και κωδικοποιούνται από τον MVC κωδικοποιητή της nokia [10]. Έπειτα με χρήση του RTP packetizer ο οποίος κατασκευάστηκε για την συγκεκριμένη πτυχιακή, η MVC ακολουθία μετατρέπεται σε RTP πακέτα κάνοντας χρήση μίας από τις τρείς διαθέσιμες δομές ωφέλιμου φορτίου που ορίζονται από το πρότυπο του RTP για το MVC [4]. Η RTP ακολουθία στην συνέχεια πακετάρεται από τον διακομιστή ο οποίος κατασκευάστηκε για το συγκεκριμένο πείραμα και αποστέλλεται μέσω UDP/IP καναλιών. Τα πακέτα τα οποία λαμβάνονται από τον χρήστη ξαναμετατρέπονται σε συνεχόμενη ακολουθία από την εφαρμογή που κατασκευάστηκε για τον χρήστη, αφαιρούνται οι RTP Headers και η τρισδιάστατη ακολουθία αποκωδικοποιείται και αναπαράγεται. Τέλος, κάνοντας χρήση του εξομοιωτή δικτύου “Dummynet”[13], διερευνάται η μείωση της ποιότητας των στερεοσκοπικών ακολουθιών όταν αυτά μεταδίδονται σε ένα κανάλι με απώλειες.

Όπως απεικονίζεται , όλα τα εργαλεία είναι συμβατά με την προηγούμενη τεχνολογία H.264/AVC, οπότε εάν ένας χρήστης δεν διαθέτει κατάλληλη συσκευή για αναπαραγωγή τρισδιάστατου βίντεο λόγω τεχνικών χαρακτηριστικών, όπως οθόνη χωρίς δυνατότητα 3D απόδοσης ή μικρότερο από το απαιτούμενο εύρος ζώνης, τότε να έχει την δυνατότητα να κάνει λήψη μόνο της βασικής ακολουθίας. Έπειτα χρησιμοποιώντας είτε έναν H.264/AVC είτε έναν H.264/MVC αποκωδικοποιητή να είναι σε θέση να παρακολουθήσει το βίντεο δισδιάστατα.



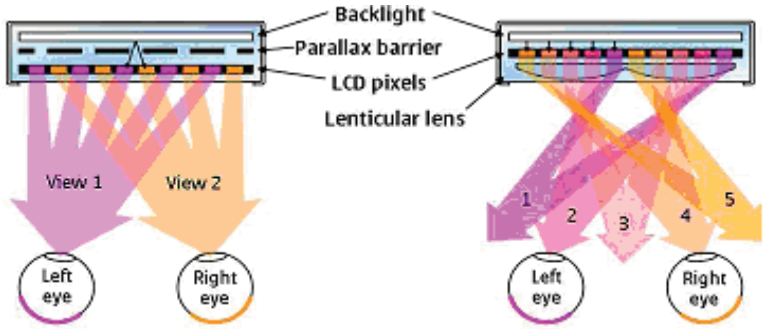
Εικόνα 1. Αρχιτεκτονική MVC συστήματος

Υπάρχουν δύο μηχανισμοί οι οποίοι μπορούν να δημιουργήσουν σε έναν παρατηρητή την αίσθηση της βύθισης σε ένα τρισδιάστατο περιβάλλον [1]. Ο πρώτος μηχανισμός είναι γνωστός ως head-mounted display(HMD), η συσκευή του οποίου τοποθετείται στο κεφάλι παρέχοντας ένα μικρό οπτικό πεδίο μπροστά από τα μάτια. Ενώ ο δεύτερος μηχανισμός είναι η χρήση 3D ή auto stereoscopic οθονών οι οποίες είναι σε θέση να έχουν επίδραση σε ένα συγκεκριμένο μικρό πεδίο γωνιών θέασης και οι οποίες αναμένονται και στα κινητά. Στον δεύτερο μηχανισμό, ο παρατηρητής είναι σε θέση να αλλάξει την γωνία θέασης του όταν προσπαθεί να δει μία σκηνή από μία άλλη οπτική γωνία σαν να ήταν ένα φυσικό αντικείμενο κάτι το οποίο δεν παρέχεται ακόμα από τον πρώτο μηχανισμό.

Κεφάλαιο 2ο. Multiple View Coding

Σε αυτό το κεφάλαιο αναλύεται η MVC κωδικοποίηση καθώς και τα πλεονεκτήματα και τα μειονεκτήματα της σε σχέση με το H.264/AVC. Επιπρόσθετα, αναλύεται η δομή της κωδικοποιημένης ακολουθίας (H.264 / MVC) , περιγράφεται ο τρόπος με τον οποίο γίνεται η ενθυλάκωση της κωδικοποιημένης πληροφορίας σε NAL Units, το είδος των πληροφοριών που μπορούν να αντληθούν από τους NAL Unit Headers και τελικά αναλύονται οι τρόποι διόρθωσης σφαλμάτων μετάδοσης που χρησιμοποιούνται στο H.264/MVC.

Το H.264/MVC είναι βασισμένο στο πρότυπο Annex H της ITU-T Rec H.264 και στο ISO/IEC 14496 Part 10 (MPEG4-10). Είναι σε πολύ μεγάλο βαθμό όμοιο με το H.264/SVC τόσο από άποψη μορφής όσο και από άποψη μετάδοσης, ενώ καλύπτει ένα πολύ μεγάλο εύρος εφαρμογών 3D, όπως αυτές πού αναφέρθηκαν προηγουμένως . Ποίο συγκεκριμένα, οι δύο ποίο συναρπαστικές εφαρμογές του MVC είναι το free viewpoint video (FVV) και η εικονική πραγματικότητα (telepresence or virtual reality) [10]. Η πρώτη εφαρμογή είναι σε θέση να προσφέρει την ίδια λειτουργικότητα η οποία είναι γνωστή από τα 3D γραφικά των υπολογιστών, όπου ο χρήστης είναι σε θέση να διαλέξει όποια όψη και οπτικό προσανατολισμό θέλει σε μία οπτική σκηνή και να έχει ελεύθερη διαδραστική πλοήγηση. Ενώ η δεύτερη εφαρμογή μεταφέρει στον άνθρωπο την ψευδαίσθηση ότι βρίσκεται σε μια διαφορετική τοποθεσία από την πραγματική του . Σε όλες αυτές τις εφαρμογές υπάρχουν δύο διαφορετικοί τρόποι απεικόνισης τρισδιάστατου βίντεο. Όπως απεικονίζεται στην Εικόνα 2, είναι δυνατόν να χρησιμοποιηθεί μια στερεοσκοπική οθόνη η οποία είναι ικανή να αποδώσει 2 διαφορετικές όψεις, οπότε η εικόνα αντιλαμβάνεται ως τρισδιάστατη, ενώ στην δεύτερη και ποιο αποδοτική απεικόνιση, η οθόνη έχει την δυνατότητα να αποδώσει περισσότερες από 2 όψεις έτσι ώστε το αντιληπτό βίντεο στον παρατηρητή να εξαρτάται από τη θέση του σε σχέση με την οθόνη [10]. Στην δεύτερη περίπτωση, ο χρήστης έχει την δυνατότητα εάν κινήσει το κεφάλι του/της τότε να δει τι βρίσκεται πίσω από ένα ορισμένο αντικείμενο.



Εικόνα 2. Στερεοσκοπική απεικόνιση έναντι Πολλαπλής 3D απεικόνισης

2.1. H.264/MVC Κωδικοποίηση

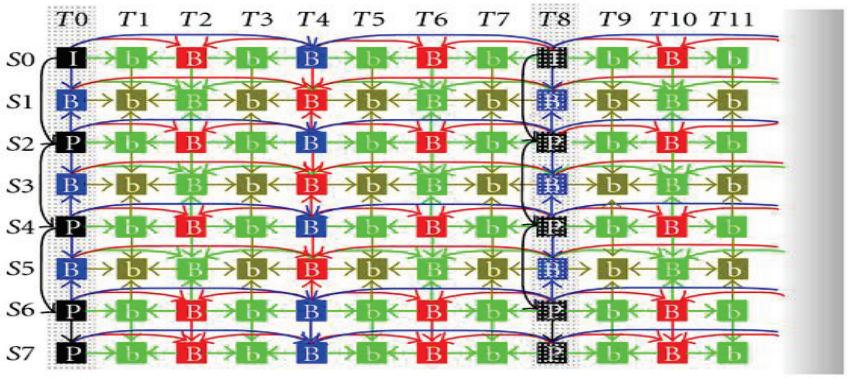
Σε αυτήν την ενότητα αναλύεται ο τρόπος με τον οποίο γίνεται η κωδικοποίηση πολλαπλών όψεων κάνοντας χρήση του H.264/MVC καθώς επίσης και οι μηχανισμοί οι οποίοι κληρονομήθηκαν από τους προγόνους του H.264/AVC και H.264/SVC.

Στο H.264/MVC, λόγω των πολλαπλών όψεων, η μετάδοση τρισδιάστατων ακολουθιών βασίζεται στην συμπίεση, αφού ο όγκος των δεδομένων αυξάνεται δραματικά καθώς αυξάνεται ο αριθμός των διαφορετικών όψεων [1]. Λόγω ενός επιπλέον πλεονασμού ο οποίος ονομάζεται **view redundancy**, το

H.264/MVC παρέχει υψηλότερη αναλογία συμπίεσης από το H.264/AVC. Αυτό επιτυγχάνεται διότι οι τρισδιάστατες ακολουθίες δημιουργούνται από διαφορετικές κάμερες οι οποίες είναι τοποθετημένες σε διαφορετικές θέσεις υπό διαφορετικές γωνίες, οπότε είναι δύο διαφορετικές αναπαραστάσεις της ίδιας σκηνής [10]. Μια από τις εν λόγω όψεις κωδικοποιείται ανεξάρτητα (βασική όψη) και είναι βασισμένη στο H.264 / AVC όπως περιγράφεται στο Annex A του H.264. Αντίθετα, οι υπόλοιπες όψεις ονομάζονται μη-βασικές και βασίζονται στο H.264/MVC, ενώ κωδικοποιούνται εξαρτώμενες μόνο από την βασική όψη (predicted view) ή σε συνδυασμό της βασική όψης με μία μη βασική όψη (Bi-predicted view). Για την αποκωδικοποίηση των μη βασικών όψεων θα πρέπει να γίνει πρόβλεψη των pixels τους (interview prediction), οπότε η ύπαρξη όλων των όψεων από τις οποίες εξαρτάται η κάθε μη βασική όψη είναι αναγκαία [4]. Αυτή η επιπλέον συσχέτιση μεταξύ των όψεων, ειδικά όταν ο αριθμός των διαφορετικών όψεων είναι μεγάλος, απαιτεί έναν αριθμό μνήμης προσωρινής αποθήκευσης ο οποίος είναι απαγορευτικός. Για να επιτευχθεί αυτή η πρόβλεψη, θα πρέπει να γίνεται σωστή διαχείριση της μνήμης των προσωρινά αποθηκευμένων αποκωδικοποιημένων εικόνων. Στο H.264/ MVC, εκτός από την inter-view πρόβλεψη , συνεχίζονται να χρησιμοποιούνται τα εργαλεία του H.264/AVC.

Στο πρότυπο H.264/MVC κληρονομείται η χρονική κλιμάκωση (Temporal Scalability) από το H.264/SVC ενώ παράλληλα εισάγεται η κλιμάκωση όψης (View Scalability). Η κλιμάκωση όψης μπορεί να χρησιμοποιηθεί παράλληλα με την χρονική κλιμάκωση έτσι ώστε να υπάρχει προσαρμογή ανάλογα με τις προτιμήσεις του χρήστη, το εύρος δικτύου του και την πολυπλοκότητα του αποκωδικοποιητή [4]. Ένα παράδειγμα και των δύο κλιμακώσεων απεικονίζεται στο κεφάλαιο 2.2 στην Εικόνα 4, όπου κάθε υποσύνολο κάθε όψης μπορεί να διαχωρίζεται εύκολα από ολόκληρη την ακολουθία , δίνοντας την δυνατότητα στον δέκτη, κάνοντας χρήση της ίδιας κωδικοποιημένης ακολουθίας, ανάλογα με την οθόνη και τον αποκωδικοποιητή που διαθέτει, να αναπαράγει τον αριθμό των όψεων και τον ρυθμό πλαισίων που θέλει και μπορεί. Παραδείγματος χάριν, εάν ένας διακομιστής μεταδίδει μία κωδικοποιημένη ακολουθία 32 όψεων και ό δέκτης έχει στερεοσκοπική οθόνη, τότε να δίδεται η δυνατότητα στον δέκτη να λάβει ,να αποκωδικοποιήσει και να αναπαράγει μόνο τις 2 όψεις στον ρυθμό πλαισίων που θέλει και μπορεί . Η κλιμάκωση όψης έπαιξε καθοριστικό ρόλο στην συμβατότητα του H.264/MVC με το H.264/AVC. Ακόμα οι συνηθισμένες οθόνες πού αναπαράγουν δισδιάστατο βίντεο χρησιμοποιούνται ευρέως, οπότε εάν ένας αποκωδικοποιητής H.264/AVC λάβει μία MVC ακολουθία τότε είναι σε θέση να χρησιμοποιήσει μόνο την βασική όψη και να αποκωδικοποιήσει μία 2D αναπαράσταση και να απορρίψει τα υπόλοιπα δεδομένα .

Μία βασική δομή του MVC η οποία εκμεταλλεύεται και την inter-view πρόβλεψη αλλά και την ιεραρχική χρονική κλιμάκωση αναπαριστάται στην Εικόνα 3 . Για κάθε όψη χρησιμοποιείται μία αρχιτεκτονική πρόβλεψης η οποία χρησιμοποίει ιεραρχικές εικόνες B. Κάθε όψη με άρτιο αναγνωριστικό όψης (view_id) έχει την προηγούμενη άρτια όψη σαν εικόνα εξάρτησης, ενώ μία όψη με περιττό αναγνωριστικό όψης(view_id) έχει την προηγούμενη και την επόμενη άρτια όψη σαν εικόνα εξάρτησης. Κάθε ομάδα εικόνων(GOP) περιέχει μία εικόνα αναφοράς (anchor ή I-Frame) η οποία μπορεί να αποκωδικοποιηθεί μόνη της. Για να αποκωδικοποιηθούν οι υπόλοιπες εικόνες του ίδιου GOP πρέπει πρώτα να έχουν αποκωδικοποιηθεί τα προηγούμενα πλαίσια [1].



Εικόνα 3. Παράλληλη χρήση inter-view πρόβλεψης και ιεραρχικής χρονικής κλιμάκωσης

2.2. Πρότυπο H.264/MVC

Σε αυτήν την ενότητα αναλύεται η δομή της ροής των bit σύμφωνα με το πρότυπο H.264/MVC, περιγράφονται οι τύποι των NAL μονάδων και των NALU Headers που χρησιμοποιούνται , ενώ επίσης , αναλύονται οι όροι: Sequence Parameter Set , Picture Parameter Set, Supplemental Enhancement Information, Access Unit.

2.2.1. Network Abstraction Layer (NAL) & Video Coding Layer (VCL)

Στο H.264/MVC τα **V***ideo* **C***oding* **L***ayer* και **N***etwork* **A***bstraction* **L***ayer* Units κληρονομήθηκαν από το H.264/AVC [4]. Το VCL περιέχει την λειτουργία της επεξεργασίας του σήματος καθώς επίσης και διάφορους μηχανισμούς οι οποίοι παρέχουν την πολύ υψηλή συμπίεση του H.264/MVC (μετασχηματισμό, κβαντισμό, πρόβλεψη κίνησης, πρόβλεψη ενδο-στρώματος (inter-layer prediction)). Τα NAL Units χωρίζονται σε VCL NAL Units και non-VCL NAL Units [1]. Ένα VCL NAL Unit περιέχει δεδομένα τα οποία αναπαριστούν τις τιμές των δειγμάτων των πλαισίων του video, ενώ τα non-VCL NAL Units περιέχουν οποιαδήποτε άλλη πληροφορία πρέπει να περιέχει η κωδικοποιημένη ακολουθία, όπως το parameter set , το **S**upplemental **E**nhancement **I**nformation NAL Unit ή και κάποιες επιπλέον συμπληρωματικές πληροφορίες βελτίωσης ενός πλαισίου. Οπότε κάθε τεμάχιο (slice) το οποίο παράγεται από το VCL, ενθυλακώνεται σε ένα ή περισσότερα VCL NAL Units.

Η ενθυλάκωση της κωδικοποιημένης ακολουθίας σε NAL Units γίνεται με την χρήση των NAL Unit Headers. Οι NALU headers εισάγονται πάντα πριν το ωφέλιμο φορτίο και υποδεικνύουν τον τύπο του NAL Unit. Στο MVC, όπως έχουμε είδη προαναφέρει, η βασική όψη κωδικοποιείται ανεξάρτητα και είναι συμβατή με το H.264/AVC, οπότε οι κωδικοποιημένες πληροφορίες της εικόνας ενθυλακώνονται σε VCL NAL Units κάνοντας χρήση του **H.264/AVC NAL Unit Header** (1 byte). Ο εν λόγω header περιγράφεται εκτός των άλλων στο πρότυπο του RTP για το H.264/AVC [5].

Για την ενθυλάκωση των πληροφοριών των μη-βασικών όψεων σε κωδικοποιημένα τεμάχια έχει εισαχθεί ένας νέος τύπος NALU Header. Το όνομα του εν λόγω NALU Header είναι 'τεμάχιο της προέκτασης MVC' (**coded slice of MVC extension**) (4 byte) [4]. Επιπλέον, κάποιες ιδιότητες όπως οι χρονικές στάθμες(temporal level) οι οποίες δεν δηλώνονται στο H.264/AVC, καθιστούν αναγκαία την εισαγωγή ενός ακόμα νέου NALU Header με το όνομα **Prefix NAL Unit Header** (4 bytes) [4], ο οποίος ορίζεται και στο SVC[6]. Ένας Prefix NALU Header προηγείται πάντα του H.264/AVC NALU Header (base view) και του coded slice of MVC extension(non base view) και περιέχει ουσιώδης χαρακτηριστικά των πλαισίων . Για την δημιουργία αυτών των δύο νέων τύπων NAL μονάδων, το MVC προεκτείνει το ενός-byte H.264/AVC NAL Unit header σε ακόμα 3 bytes, αποτελούμενο δηλαδή από 4 bytes συνολικά. Όταν μία MVC ροή από bit περιέχει και τις νέες NAL μονάδες και εισαχθεί σε έναν H.264/AVC αποκωδικοποιητή τότε αυτές θα αγνοηθούν και θα αποκωδικοποιηθούν μόνο τα NALUs τα οποία ορίζονται στο πρότυπο H.264/AVC, έχοντας ως αποτέλεσμα ένα δισδιάστατο βίντεο.

Ο Πίνακας i περιγράφει την δομή του H.264/AVC NAL Unit Header ο οποίος:

1. ενθυλακώνει τα NALUs την βασικής όψης
2. χρησιμοποιείται ως το πρώτο byte για την ενθυλάκωση των μη βασικών όψεων μέσω του "coded slice of MVC extension"
3. χρησιμοποιείται ως το πρώτο byte του "prefix NALU Header"

Στην συνέχεια αναλύεται η πληροφορία που παρέχει το κάθε πεδίο.

Πίνακας i. H.264/AVC NAL Unit Header

0	1	2	3	4	5	6	7	8
F	NRI		Type					

F: 1 bit
Forbitten_Zero_Bit: Δέχεται την τιμή 0 ή 1. Εάν είναι 1 τότε υποδεικνύει ότι το NAL Unit ίσως να περιέχει παραβίαση της σύνταξης ή λάθος στα bit του ωφέλιμου φορτίου του.

NRI: 2 bits
NAL_Ref_Idc: Το πεδίο αυτό ορίζει εάν το NAL Unit το οποίο ακολουθεί χρησιμοποιείται για ανακατασκευή των εικόνων αναφοράς σε μελλοντικές προβλέψεις. Εάν η τιμή του πεδίου είναι 00 (binary) τότε δεν είναι αναγκαία η αποκωδικοποίηση του NAL Unit οπότε μπορεί και να απορριφτεί χωρίς να υπάρχει ρίσκο ακεραιότητας των εικόνων αναφοράς στην ίδια όψη. Μία τιμή μεγαλύτερη του 00 υποδηλώνει ότι υπάρχει αναγκαιότητα αποκωδικοποίησης του συγκεκριμένου NAL Unit είτε επειδή χρειάζεται για την διατήρηση της ακεραιότητας των εικόνων αναφοράς της ίδιας όψης είτε επειδή περιέχει κάποιο σύνολο παραμέτρων. Η μεγαλύτερη δυνατή τιμή του είναι η 11, ακολουθούμενη από την 10 και έπειτα από την 01 και τελικά από την 00 η οποία είναι και η μικρότερη[5].

Type: 5 bits
NAL_Unit_type: Αυτή η συνιστώσα ορίζει τον τύπο του NAL Unit. Βάση προτύπου, εάν η τιμή είναι 1 (non-IDR) ή 5 (IDR) τότε το NALU ενθυλακώνει ωφέλιμο φορτίο της βασικής όψης (base view) [5] το οποίο αναφέρεται στο παλαιό SPS (Sequence Parameter Set) τύπου 7 [15]. Εάν η τιμή του είναι 14 τότε είναι ένας Prefix NAL Unit Header , ενώ εάν είναι 20 τότε είναι ένας coded slice of mvc extension [4], επισημαίνοντας ότι ενθυλακώνεται ωφέλιμο φορτίο της μη-βασικής όψης το οποίο αναφέρεται στο καινούργιο SPS (Subset Sequence Paramer Set) τύπου 15 [6]. Στους δυο τελευταίους τύπους ακολουθούν τρία bytes τα οποία αναπαριστώνται στον Πίνακα ii και αναλύονται παρακάτω ενώ το SPS αναλύεται στην Ενότητα 2.2.2.

Πίνακας ii. Επέκταση του H.264/AVC NAL Unit Header (3 Bytes)

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
R	I	PRID						VID										TID			A	V	O	

R: 1 bit
Reserverd_One_bit: Bit το οποίο αφήνεται ώστε να χρησιμοποιηθεί από μελλοντικές επεκτάσεις. Η τιμή του πρέπει να είναι μόνιμα 1 και να αγνοείται απτούς αποκωδικοποιητές.

I: 1 bit
Idr_flag: Bit το οποίο εάν οριστεί καθορίζει ότι το NAL Unit είναι μία στιγμιαία αποκωδικοποίηση ανανέωσης όψης (IDR). Οι IDR εικόνες των μη-βασικών ονομάζονται view-IDR(V-IDR) και έχουν την δυνατότητα να βασίζονται σε άλλες εικόνες (στο ίδιο σημείο πρόσβασης) από άλλες όψεις μέσω inter-view πρόβλεψης. Ο όγκος της πληροφορίας των IDR ή V-IDR εικόνων θα είναι μεγαλύτερος, ενώ θα πρέπει να τους παρέχεται μεγαλύτερη προστασία. Τέλος, οι στιγμιαίες IDR εικόνες λειτουργούν ως σημεία τυχαίας προσπέλασης.

PRID: 6 bits
Priority_id: Αυτή η σημαία καθορίζει ένα αναγνωριστικό προτεραιότητας του NAL Unit. Όσο μικρότερη είναι η τιμή του τόσο μεγαλύτερη προτεραιότητα έχει. Μπορεί να χρησιμοποιηθεί για την καλύτερη προστασία των σημαντικότερων NAL Units μέσω αλγορίθμων διόρθωσης σφαλμάτων.

VID: 10 bits
View_id: Αυτό το χρήσιμο πεδίο καθορίζει το αναγνωριστικό της όψης όπου ανήκει το NAL Unit. Χρησιμοποιείται παράλληλα με το temporal_id για την εξαγωγή και προσαρμογή της ροής των bit.

TID: 3 bits
Temporal_id: Καθορίζει την χρονική ιεραρχία στρώματος. Όσο μικρότερη είναι η τιμή του temporal_id τόσο χαμηλότερος είναι ο ρυθμός πλαισίων. Οι κωδικοποιημένες εικόνες με υψηλότερο temporal_id

τυπικά εξαρτώνται από αυτές με χαμηλότερη τιμή του temporal_id μέσα στην ίδια όψη, αλλά ποτέ δεν εξαρτώνται από κωδικοποιημένες εικόνες με υψηλότερο temporal_id.

A: 1 bit
Anchor_pic_flag: Η τιμή 1 καθορίζει ότι το NAL Unit περιέχει μία anchor εικόνα. Οι anchor εικόνες είναι ολόκληρες εικόνες οι οποίες δεν χρειάζεται να προβλέψουν κάποια τιμή τους από κάποια άλλη εικόνα, οπότε μπορούν να χρησιμοποιηθούν σαν σημεία τυχαίας προσπέλασης.

V: 1 bit
Inter_View_Flag: Αυτή η σημαία καθορίζει ότι το ωφέλιμο φορτίο που ακολουθεί χρησιμοποιείται για inter-view πρόβλεψη όταν η τιμή του είναι 1 αλλιώς είναι 0.

O: 1 bits
Reserved_One_Bit: Bit το οποίο αφήνεται ώστε να χρησιμοποιηθεί από μελλοντικές επεκτάσεις. Η τιμή του πρέπει να είναι μόνιμα 1 και αγνοείται απτούς αποκωδικοποιητές.

Στον πίνακα iii γίνεται αναπαράσταση των βασικών τύπων των NALU [15]. Οι τύπου οι οποίοι δεν έχουν ξανά-αναφερθεί αναλύονται παρακάτω στην ίδια ενότητα.

Πίνακας iii . NAL Unit τύποι

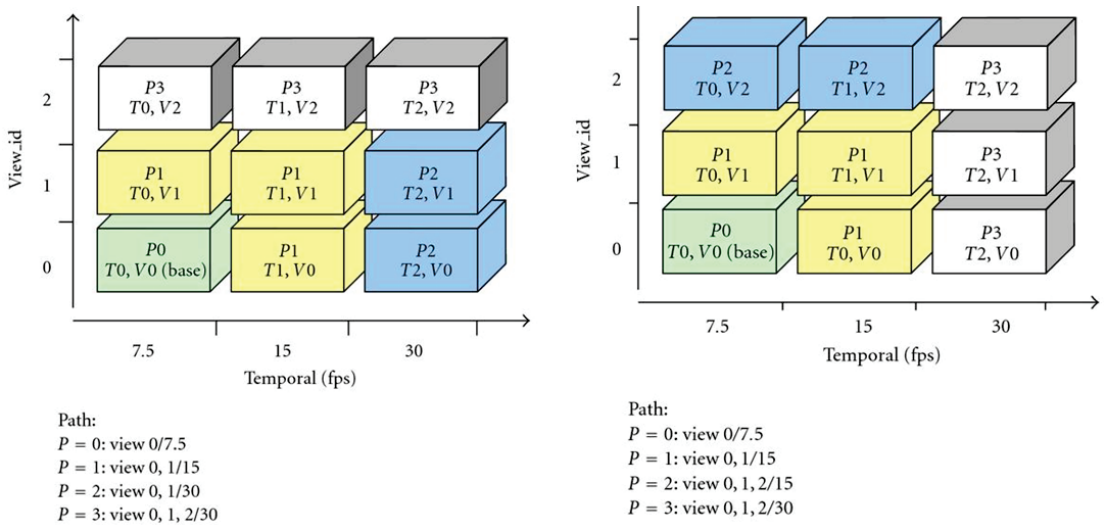
Nal_Unit_Type (Dec)	Content of NAL Unit
1	Coded slice of a non-IDR picture
5	Coded slice of an IDR picture
6	Supplemental enhancement information (SEI)
7	Sequence parameter set
8	Picture parameter set
9	Access unit delimiter
10	End of sequence
11	End of stream
12	Filler data
14	Prefix NAL Unit Header
15	Subset Sequence Paramer Set
20	coded slice of mvc extension

Στην συνέχεια εξετάζεται ο τρόπος με τον οποίον αλλάζοντας τις τιμές των πεδίων των NALU Header μπορούμε να πετύχουμε διάφορες κλιμακώσεις. Η Εικόνα 4 απεικονίζει δύο παραδείγματα σε αναθέσεις τιμών του priority_id [1], τα οποία αποδίδουν δύο διαφορετικά προσαρμοστικά μονοπάτια της ίδιας MVC ροής δεδομένων, η οποία περιέχει τρείς διαφορετικές όψεις και τρία διαφορετικά χρονικά επίπεδα. Στο παρακάτω παράδειγμα μία ποιό «έξυπνη» υλοποίηση θα ήταν η παράλληλη χρήση των πεδίων priority_id, view_id, and temporal_id ώστε να παρέχεται μεγαλύτερη προσαρμογή.

Στο πρώτο παράδειγμα,

- Το priority_id, για 7.5 fps και εξαγωγή της βασικής όψης ορίζεται με την τιμή 0.
- Το priority_id, για 15 fps και εξαγωγή δύο όψεων ορίζεται με την τιμή 1.
- Το priority_id, για 30 fps και εξαγωγή δύο όψεων ορίζεται με την τιμή 2.
- Το priority_id, για 30 fps και εξαγωγή τριών όψεων ορίζεται με την τιμή 3.

- *Στο δεύτερο παράδειγμα*
  - Οι δύο πρώτες αναθέσεις τιμών και η τελευταία είναι παρόμοια με το πρώτο παράδειγμα.
  - Στην τέταρτη ανάθεση τιμής του priority_id, η τιμή 2 έχει ως αποτέλεσμα την εξαγωγή τριών όψεων σε χαμηλότερο ρυθμό πλαισίων (15 fps).



T: Temporal_id || V: View_id || P: Priority_id

Εικόνα 4. Ανάθεση τιμών του priority_id σε bitstream 3 όψεων με τρία επίπεδα χρονικής κλιμάκωσης.

2.2.2. Parameter Set

Μια βασική φιλοσοφία του H.264 είναι η δημιουργία αυτοδύναμων πακέτων έτσι ώστε οι μηχανισμοί όπως ο header duplication του RFC 2429 ή ο Header Extension Code(HEC) του MPEG-4 να καθιστούν περιττοί [5]. Αυτό επιτεύχθηκε αποσυνδέοντας εκείνες τις πληροφορίες οι οποίες ήταν κοινές για περισσότερα από ένα τεμάχια (slices) από την ροή πολυμέσων επιτυγχάνοντας παράλληλα υψηλότερη συμπίεση. Αυτές οι υψηλότερου στρώματος πληροφορίες θα πρέπει να στέλνονται με αξιοπιστία , ασύγχρονα και πριν η RTP ροή πακέτων που περιέχει τα συναφή τεμάχια μεταδοθεί . Η αποστολή αυτών των πληροφοριών μπορεί να γίνει είτε εντός του ίδιου καναλιού (in-band), εάν οι εφαρμογές δεν περιέχουν επιπλέον κανάλι μεταφοράς, είτε με την χρήση επιπλέον καναλιού (out-of-band), κάνοντας χρήση του control protocol. Αυτός ο συνδυασμός υψηλότερου στρώματος παραμέτρων ονομάζεται σύνολο παραμέτρων (**parameter set**) [1] . Το πρότυπο H.264 ορίζει δύο εκδοχές του parameter set: "Sequence Parameter Set" *(Nal_Unit_Type=7)* και"Picture Parameter Set" *(Nal_Unit_Type=8)*. Μία ενεργή ακολουθία συνόλου παραμέτρων (SPS) παραμένει αμετάβλητη καθ' όλη την κωδικοποιημένη ακολουθία, ενώ μία ενεργή εικόνα συνόλου παραμέτρων (PPS) παραμένει αμετάβλητη εντός μίας κωδικοποιημένης εικόνας. Και τα δύο σύνολα παραμέτρων περιέχουν δομές οι οποίες περιέχουν πληροφορίες όπως το μέγεθος τις εικόνας, προαιρετικές λειτουργίες κωδικοποίησης που χρησιμοποιούνται και τα "macroblock to slice group map" . Για να υπάρχει δυνατότητα να αλλαχθούν οι τιμές του συνόλου των παραμέτρων χωρίς να χρειάζεται να ξαναστέλνονται συνέχεια ενημερώσεις, ο κωδικοποιητής και ο αποκωδικοποιητής μπορούν να διατηρούν μια λίστα από περισσότερα από ένα σύνολα παραμέτρων.

Στο MVC, η βασική ακολουθία συνεχίζει να χρησιμοποίει το "παλαιό SPS" *(Nal_Unit_Type=7)* ενώ οι μη βασικές ακολουθίες χρησιμοποιούν ένα νέο subset του SPS με NALU τύπο ίσο με *(Nal_Unit_Type=15)* [5]. Το Subset του SPS συνεχίζει να περιέχει το «παλαιό» SPS αλλά περιέχει και το SPS MVC Extension [16]. Οι τρείς ποιο σημαντικές πληροφορίες οι οποίες δίνονται από το SPS extension είναι:

- ***Αναγνώριση όψης*** - περιέχει τον αριθμό των διαφορετικών όψεων και την λίστα των αναγνωριστικών των όψεων (View_id).
- **Πληροφορίες εξάρτησης όψεων** – αποτελεί μια σειρά από σήματα τα οποία υποδεικνύουν τον αριθμό των inter-view εικόνων αναφοράς. Ξεχωριστές πληροφορίες εξάρτησης όψεων παρέχονται για τις εικόνες «άγκυρες» και τις υπόλοιπες εικόνες για την παροχή ευελιξίας κατά την πρόβλεψη.
- **Ένδειξη επιπέδων για τα σημεία λειτουργίας** – Η ένδειξη επιπέδων είναι ένας δείκτης των απαιτήσεων σε πόρους (για τον αποκωδικοποιητή) που έχει σχέση με ένα συγκεκριμένο επίπεδο (σημείο λειτουργίας) [16].

Αναφορικά, ο όρος σημείο λειτουργίας (operation point) αναφέρεται σε ένα σημείο του bitstream το οποίο αναπαριστά ένα συγκεκριμένο επίπεδο χρονικής κλιμάκωσης και κλιμάκωσης όψης. Περιέχει μόνο εκείνα τα NALUs τα οποία είναι αναγκαία για την αναπαραγωγή ενός βίντεο με συγκεκριμένες όψεις σε συγκεκριμένο ρυθμό πλαισίων [4].

Ο κάθε header ενός τεμαχίου περιέχει μία «κωδική» λέξη η οποία προσδιορίζει ποιο σύνολο παραμέτρων της λίστας πρέπει να χρησιμοποιηθεί. Αυτός ο διαχωρισμός δίνει την δυνατότητα στα τεμάχια (slices) των διαφορετικών όψεων να χρησιμοποιούν είτε τα ίδια SPS - PPS είτε διαφορετικά [4]. Τέλος, ο κωδικοποιητής θα πρέπει να θέτει την τιμή του NRI πεδίου του H.264/AVC header των SPS με την τιμή 11 (bin) επισημαίνοντας την αναγκαιότητα αποκωδικοποίησης των NALUs του SPS [5].

2.2.3. Supplemental enhancement information (SEI)

Στις εφαρμογές πολλαπλών όψεων, εκτός από τα σύνολα παραμέτρων, γίνεται χρήση SEI (Supplemental enhancement information) μηνυμάτων [16]. Τα SEI μηνύματα παρέχουν συμπληρωματική πληροφορία η οποία δεν χρησιμοποιείται εντός της διαδικασίας αποκωδικοποίησης (βάση προτύπου) μιας κωδικοποιημένης εικόνας. Η χρήση SEI μηνυμάτων μπορεί να μην είναι απαραίτητη ή κατάλληλη σε κάποια MVC συστήματα.
Στο H.264/MVC κληρονομούνται τα SEI μηνύματα τα οποία ορίζονται στο H.264/MPEG-4 AVC, ενώ παράλληλα ορίζονται αρκετά νέα SEI μηνύματα.

Στην συνέχεια δίδεται μία σύντομη περίληψη των SEI μηνυμάτων:

- *Parallel decoding information SEI message*

  Υποδεικνύει ότι οι όψεις ενός σημείου προσπέλασης είναι κωδικοποιημένες κατά τέτοιο τρόπο ώστε να επιτρέπουν την παράλληλη αποκωδικοποίηση. Παραδείγματος χάριν, η σηματοδότηση ενός περιορισμού (ο οποίος επιβάλλεται από το MVC κωδικοποιητή) σύμφωνα με τον οποίον μόνο ένα macroblock μιας συγκεκριμένης όψης εξαρτάται από ένα υποσύνολο από macroblocks άλλων όψεων, έχει ως αποτέλεσμα την υλοποίηση καλύτερου παραλληλισμού κατά της διαδικασία αποκωδικοποίησης.

- *MVC scalable nesting SEI message:*

  Επιτρέπει την επαναχρησιμοποίηση των υφιστάμενων SEI μηνυμάτων δείχνοντας τις όψεις (ή τις χρονικές στάθμες) για τις οποίες συνεχίζουν να ισχύουν τα παλαιότερα SEI μηνύματα.

- *View scalability information SEI message:*

  Περιέχει πληροφορίες όψεων και επεκτασιμότητας για συγκεκριμένα σημεία λειτουργίας εντός της κωδικοποιημένης ακολουθίας. Μεταξύ άλλων, σηματοδοτεί πληροφορίες όπως τον ρυθμό των bit και των πλαισίων ως μέρος του μηνύματος για ένα υποσύνολο των σημείων λειτουργίας. Αυτή η πληροφορία μπορεί να είναι χρήσιμη για την καθοδήγηση της διαδικασίας εξαγωγής της ροής των bit.

- *Multiview scene information SEI message:*

  Υποδεικνύει την μέγιστη διαφορά των όψεων σε ένα access unit. Αυτό το μήνυμα μπορεί να χρησιμοποιηθεί είτε για την επεξεργασία της αποκωδικοποιημένης όψης πριν την απόδοση της στην 3D οθόνη είτε για την τοποθέτηση υπότιτλων και λεζάντων σε μια 3-D σκηνή.

- *Multiview acquisition information SEI message:*

  Αυτό το SEI μήνυμα προσδιορίζει ποικίλες παραμέτρους όπως τις ενδογενής και εξωγενής παραμέτρους της κάμερας.

- *Nonrequired view component SEI message:*

  Υποδεικνύει ότι ένα μέρος των όψεων της κωδικοποιημένης ακολουθίας δεν χρειάζεται να αποκωδικοποιηθεί. Αυτό μπορεί να συμβεί εάν έχει οριστεί προς αναπαραγωγή ένα μέρος των όψεων (όχι όλες) οι οποίες όμως προς αναπαραγωγή όψεις δεν βασίζονται στις υπόλοιπες όψεις (*Nonrequired*) για την αποκωδικοποίηση τους.

- *View dependency change SEI message:*

  Με αυτό το SEI μήνυμα είναι δυνατή η σηματοδότηση αλλαγών του δέντρου εξάρτησης όψεων.

- *Operation point not present SEI message:*

  Υποδεικνύει σημεία λειτουργίας τα οποία δεν «εμφανίζονται» εντός της ροής των bit [16].

- *Base view temporal HRD SEI message:*

  Αυτό το SEI Μήνυμα συνδέεται με ένα IDR (Instantaneous Decoding Refresh) σημείο προσπέλασης και σηματοδοτεί πληροφορίες σχετικές με τις HRD (Hypothetical Reference Decoder) παραμέτρους που συνδέονται με την βασική όψη [16].

2.2.4. Access Unit

Ένα σημείο προσπέλασης περιέχει όλες τις NAL μονάδες οι οποίες σχετίζονται με ένα στιγμιότυπο για κάθε όψη [1]. Η αποκωδικοποίηση ενός σημείου προσπέλασης πάντα έχει σαν αποτέλεσμα την παραγωγή μίας αποκωδικοποιημένης εικόνας για κάθε όψη.

Σύμφωνα με το πρότυπο του MVC , ένα IDR σημείο προσπέλασης, είναι ένα σημείο προσπέλασης όπου οι εικόνες των διαφορετικών όψεων είναι IDR. Τα εν λόγω IDR σημεία προσπέλασης παρέχουν υποστήριξη τυχαίων σημείων προσπέλασης στα εκάστοτε στιγμιότυπα των όψεων. Πρέπει να σημειωθεί ότι το πρότυπο επιτρέπει σε μερικές από τις όψεις (όχι όλες) να μην περιέχουν IDR εικόνες στα εκάστοτε σημεία προσπέλασης.

Όλα τα RTP πακέτα τα οποία φέρουν τα NALUs για ένα σημείο προσπέλασης πρέπει να έχουν το ίδιο “timestamp” [14]. Το τέλος ενός σημείου προσπέλασης επισημαίνεται από το πεδίο “marker bit” του RTP Header ενώ η αρχή ενός νέου σημείου προσπέλασης ανιχνεύεται από ένα ξαφνικό άλμα του “timestamp”μεταξύ δύο διαδοχικών RTP πακέτων (ο αριθμός ακολουθίας να διαφέρει κατά ένα). Οι όροι “RTP πακέτο”, “timestamp”,“marker bit” αναλύονται στο κεφάλαιο 3.3.2 .

2.3. Nokia MVC Headers

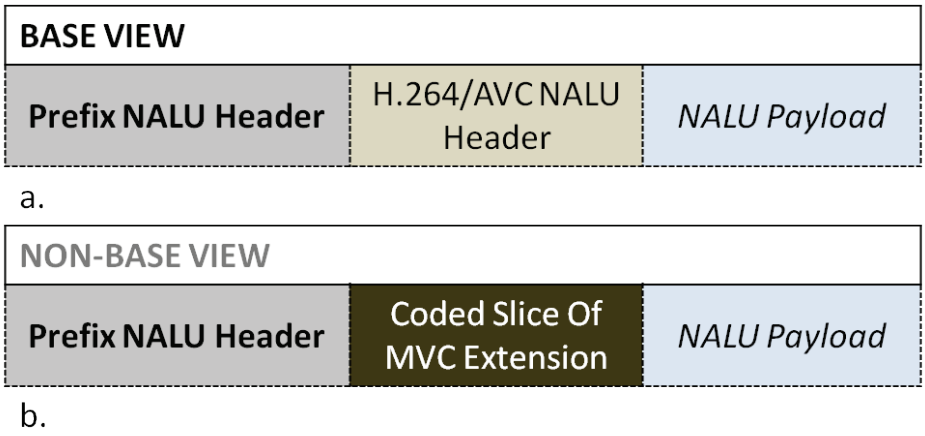
Σε αυτήν την ενότητα αναλύεται η δομή της ροής των bit του ανοιχτού κώδικα H.264/MVC κωδικοποιητή της Nokia, των διαφορετικών τύπων των NALUs, των τιμών που έχουν τα πεδία των header τους, δηλαδή κάθε πότε αλλάζουν και γιατί, καθώς και των πληροφοριών που αντλούνται από τα πεδία είτε κατά την αποκωδικοποίηση είτε κατά την επεξεργασία έπειτα από την μετάδοση τους. Ο κωδικοποιητής τροποποιήθηκε ώστε η κάθε όψη να αποθηκεύετε σε ένα διαφορετικό κωδικοποιημένο αρχείο αντί και οι δύο όψεις να αποθηκεύονται στο ίδιο κωδικοποιημένο αρχείο. Η τροποποίηση αυτή δίνει την δυνατότητα μετάδοσης της κάθε όψης σε διαφορετικό κανάλι.

Αφού κατανοήθηκε ο κώδικας του κωδικοποιητή της Nokia και ο τρόπος με τον οποίο ενθυλακώνει το ωφέλιμο φορτίο σε NALUs, υπάρχει η δυνατότητα εύρεσης:

1. Όλων των NALUs των διαφορετικών όψεων.
2. Του ωφέλιμου φορτίου που ενθυλακώνεται σε κάθε NALU.
3. Κάθε πότε χρησιμοποιείται διαφορετικού τύπου NALU, δηλαδή κάθε πότε εισάγετε ένας νέος NALU header.
4. Πόσα είναι τα επιπλέον bytes που προστίθενται στο ωφέλιμο φορτίο (Κεφάλαιο 4.2.1).

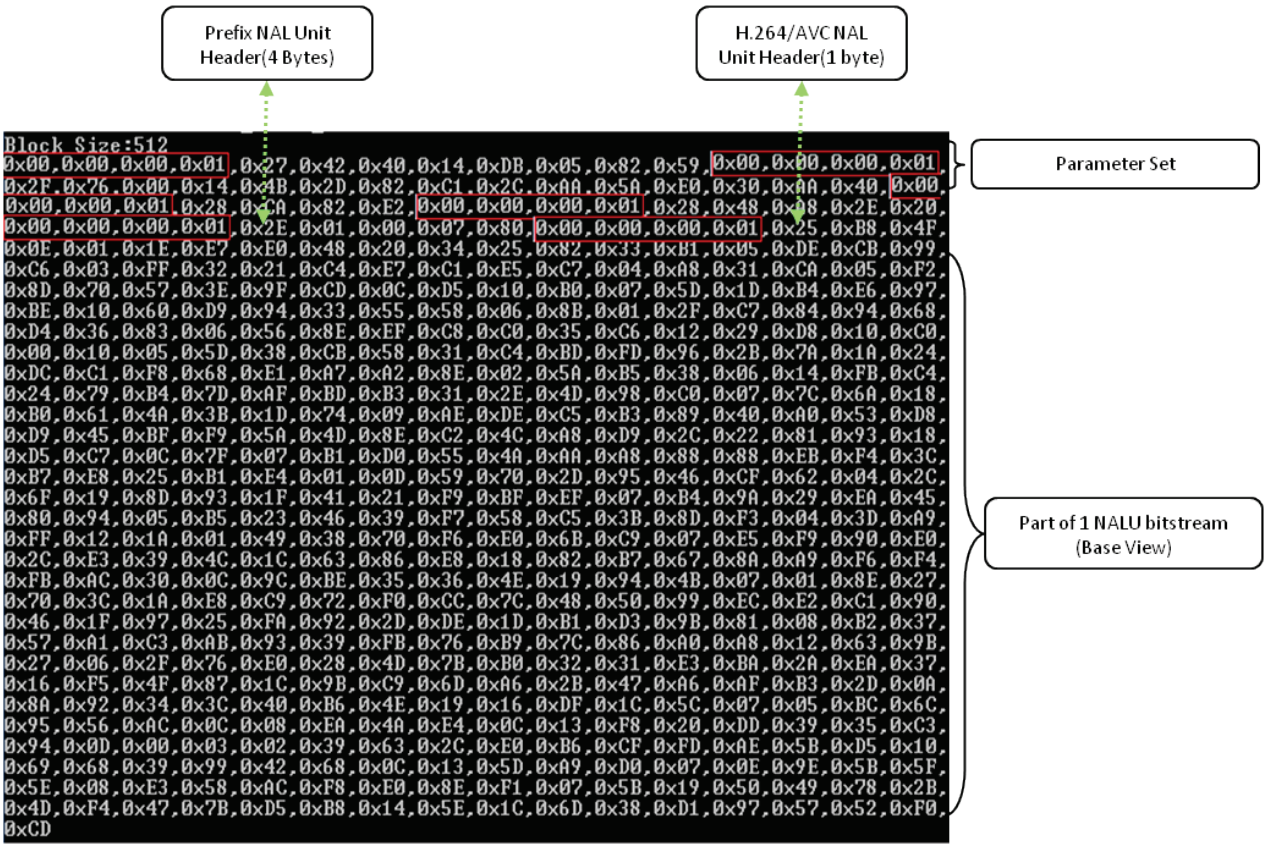
Στην Εικόνα 5.α. αναπαριστάται η δομή και ο τύπος των NALU headers που εισάγονται σε κάθε πλαίσιο της βασικής όψης, ενώ στην Εικόνα 5.β. γίνεται η αντίστοιχη αναπαράσταση της μη-βασική όψης. Για την ενθυλάκωση του ωφέλιμου φορτίου και των δύο όψεων σε NALUs, εισάγονται δύο NALU headers πριν το ωφέλιμο φορτίο. Όπως αναλύθηκε σε προηγούμενο κεφάλαιο, και στις δύο όψεις πρωταρχικά εισάγεται ο Prefix NAL Unit Header (4 bytes). Έπειτα, στην βασική όψη εισάγεται ο H.264/AVC NAL Unit Header (1 byte) όπου η ύπαρξη του καθιστά δυνατή την αποκωδικοποίηση της βασικής όψης από έναν παλαιού τύπου H.264/AVC αποκωδικοποιητή. Τέλος, εισάγεται ο δεύτερος NALU header της μη-βασικής όψης ο coded slice of MVC extension(4 bytes).

Για να είναι δυνατή η εύρεση των NALU header της κωδικοποιημένης ακολουθίας και για να αποτρέπεται η εύρεση κάποιων byte που έχουν τιμές αντίστοιχες με αυτές ενός NAL Unit Header αλλά είναι τιμές ωφέλιμου φορτίου, τοποθετείται ένα πρόθεμα τεσσάρων Byte(Prefix Bytes) πριν από κάθε NAL Unit Header, με της τιμές [0 0 0 1] (Hex).

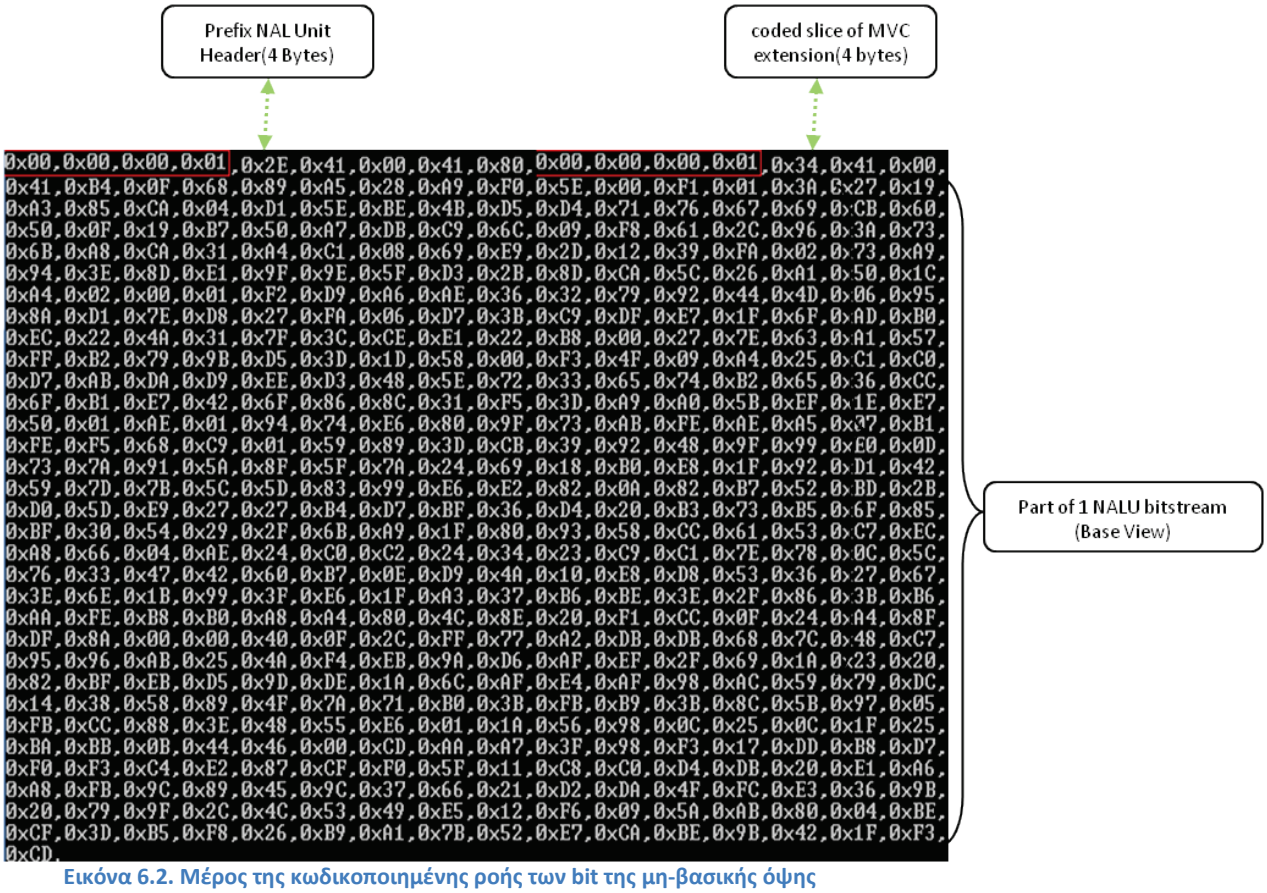


Εικόνα 5. Δομή MVC κωδικοποιημένης ακολουθίας (Nokia encoder)

Στην Εικόνα 6.1. αναπαριστάται μέρος της κωδικοποιημένης ροής από bit ενός πλαισίου της βασικής όψης. Σε κάθε εικόνα αναπαριστάται 1 block το οποίο αποτελείται από 512 bytes. Πριν την έναρξη των NAL Unit της βασικής όψης υπάρχουν τέσσερις NALU headers. Ο πρώτος header (0X27) ενθυλακώνει το sequence parameter set, αφού το πεδίο type έχει την τιμή 7 (5 τελευταία bits του πρώτου byte). Ο δεύτερος header (0X2F) ενθυλακώνει το subset sequence parameter set αφού το πεδίο type έχει την τιμή 15. Ο τρίτος και ο τέταρτος header (0X28) ενθυλακώνουν τα Picture Parameter Sets για την κάθε όψη αφού το πεδίο type και στους δύο headers έχει την τιμή 8. Οι εν λόγω headers, χρησιμοποιούνται από τον αποκωδικοποιητή για την αναγνώριση του τύπου της ακολουθίας και είναι αναγκαίοι για την αποκωδικοποίηση της ακολουθίας. Έπειτα ακολουθεί το πρώτο NALU της βασική όψης το οποίο αναγνωρίζεται από τα prefix bytes και τους NALU headers. Μετά το τέλος της κωδικοποιημένης ακολουθίας του πλαισίου της βασικής όψης, υπάρχουν οι πρώτοι NALU Headers του πρώτου πλαισίου της μη-βασικής όψης όπως απεικονίζεται στην Εικόνα 6.2 . Πρέπει να τονιστεί ότι οι τιμές των headers ισχύουν μόνο για τις reference εικόνες ,οι υπόλοιπες έχουν διαφορετικές τιμές στα πεδία των NALU headers και αναλύονται περεταίρω μετέπειτα στην ίδια ενότητα.



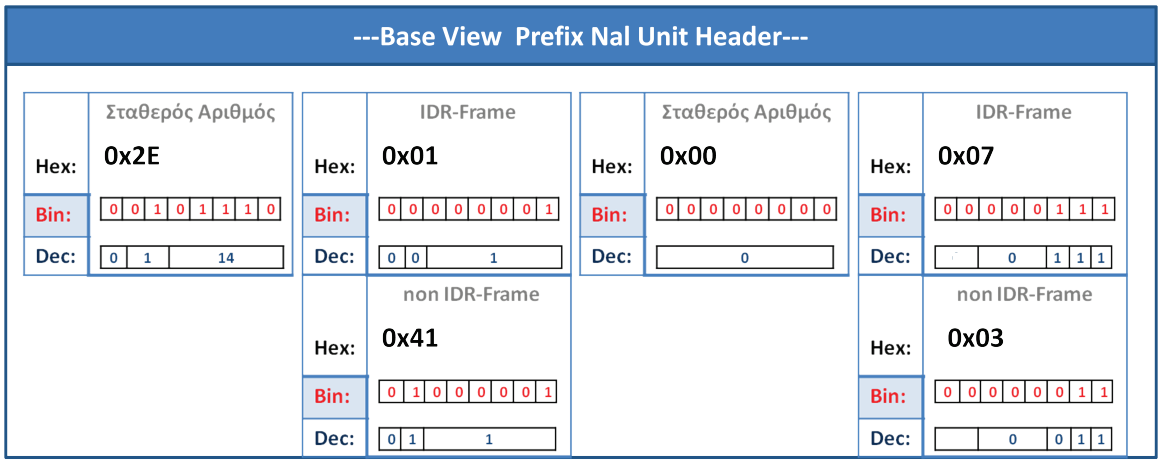
Εικόνα 6.1. Μέρος της κωδικοποιημένης ροής των bit της βασικής όψης



Εικόνα 6.2. Μέρος της κωδικοποιημένης ροής των bit της μη-βασικής όψης

Στους παρακάτω πίνακες γίνεται ανάλυση των των τιμών πεδίων των NALU headers της κωδικοποιημένης ακολουθίας (nokia Encoder) .

Πίνακας iv. Base View - Prefix NALU Header (Nokia Encoder)



F-*Forbitten_Zero_Bit (0)*: Το NAL Unit δεν περιέχει παραβίαση της σύνταξης ή λάθος στα bit του ωφέλιμου φορτίου του.

NRI-*NAL_Ref_Idc (1)*: Υπάρχει αναγκαιότητα αποκωδικοποίησης όλων των NAL Units.

Type-*Nal_Unit_Type (14)*: Ορίζεται ο τύπος του NALU Header (Prefix NAL Unit header).

IDR-Frame

R-*Reserverd_One_bit (0)*: Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

I-*Idr_flag (0)*: Το NALU ενθυλακώνει ωφέλιμο φορτίο μιας IDR εικόνας.

PRID -*Priority_id (1)*: Όλα τα NALUs έχουν την ίδια προτεραιότητα.

VID-*View_id (0):* Ορίζεται ως βασική όψη.

TID-*Temporal_id (0)*: Δεν καθορίζεται ιεραρχία χρονικού στρώματος (temporal level) [Σταθερή τιμή].

A-*Anchor_pic_flag (1)*: Το NALU ενθυλακώνει ωφέλιμο φορτίο μιας εικόνας 'άγκυρας'.

V-*Inter_View_Flag (1)*: Όλα τα NALU's χρησιμοποιούνται για inter-view πρόβλεψη.

O-*Reserved_One_Bit (1):* Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

non IDR-Frame

R-*Reserverd_One_bit (0)*: Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

I-*Idr_flag (1)*: Το NALU **δεν** ενθυλακώνει ωφέλιμο φορτίο μιας IDR εικόνας.

PRID -*Priority_id (1)*: Όλα τα NALUs έχουν την ίδια προτεραιότητα.

VID-*View_id (0):* Ορίζεται ως βασική όψη.

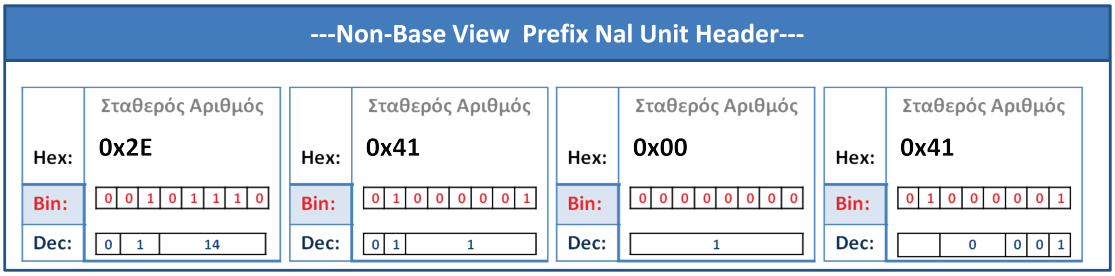
TID-*Temporal_id (0)*: Δεν καθορίζεται ιεραρχία χρονικού στρώματος (temporal level) [Σταθερή τιμή].

A-*Anchor_pic_flag (0)*: Το NALU **δεν** ενθυλακώνει ωφέλιμο φορτίο μιας εικόνας 'άγκυρας'.

V-*Inter_View_Flag (1)*: Όλα τα NALU's χρησιμοποιούνται για inter-view πρόβλεψη.

O-*Reserved_One_Bit (1):* Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

Πίνακας v. Non Base View - Prefix NALU Header (Nokia Encoder)



Οι τιμές των πεδίων όλων των πλαισίων της μη βασικής όψης περιέχουν σταθερές τιμές.

F-*Forbitten_Zero_Bit (0)*: Τα NAL Units δεν περιέχουν παραβίαση της σύνταξης ή λάθος στα bit του ωφέλιμου φορτίου τους.

NRI-*NAL_Ref_Idc (1)*: Υπάρχει αναγκαιότητα αποκωδικοποίησης όλων των NAL Units.

Type-*Nal_Unit_Type (14)*: Ορίζεται ο τύπος του NALU Header (Prefix NAL Unit header).

R-*Reserverd_One_bit (0)* : Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

I-*Idr_flag (1)*: Όλα τα πλαίσια ορίζονται ως V-IDR.

PRID -*Priority_id (1)*: Όλα τα NALUs έχουν την ίδια προτεραιότητα.

VID-*View_id (1):* Ορίζεται ως **μη-βασική όψη**

TID-*Temporal_id (0)*: Δεν καθορίζεται ιεραρχία χρονικού στρώματος (temporal level).

A-*Anchor_pic_flag (0)*: Κανένα NALU δεν ενθυλακώνει ωφέλιμο φορτίο μιας εικόνας 'άγκυρας'.

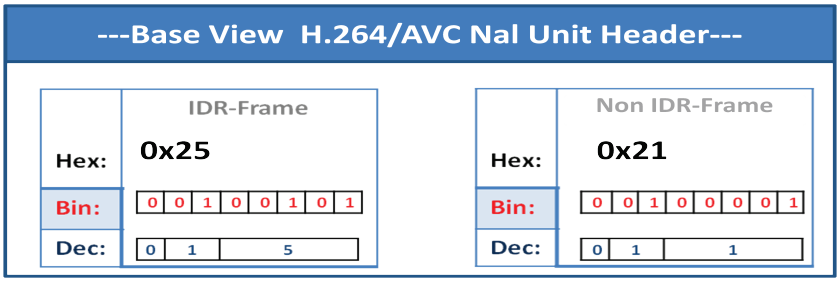
V-*Inter_View_Flag (0)*: Κανένα NALU δεν χρησιμοποιείται για inter-view πρόβλεψη.

O-*Reserved_One_Bit (1):* Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

Τα πεδία των Prefix NAL Unit Headers παρέχουν την δυνατότητα της αναγνώρισης και μεγαλύτερης προστασίας της ποιό χρήσιμης πληροφορίας κατά την μετάδοση της. Το πεδίο **View_id** μπορεί να χρησιμοποιηθεί τόσο για την αναγνώριση και την προστασία της βασικής όψης όσο για την εξαγωγή και αποστολή μόνο της βασικής όψης. Το πεδίο ***Anchor_pic_flag*** δίνει την δυνατότητα αναγνώρισης των εικόνων 'άγκυρας' ενώ το ***Inter_View_Flag*** μπορεί να χρησιμοποιηθεί για την καλύτερη προστασία των εικόνων εκείνων που χρησιμοποιούνται για πρόβλεψη από άλλες εικόνες, έχοντας ως αποτέλεσμα την μείωση της διάδοσης λάθους.

Γίνεται εύκολα αντιληπτό ότι ο κωδικοποιητής της Nokia δεν δίνει ιδιαίτερη σημασία σε μερικά πεδία. Το πεδίο ***Priority_id*** έχει μόνιμα την τιμή 1 επισημαίνοντας ότι όλα τα NAL Units έχουνε την ίδια προτεραιότητα. Επίσης το πεδίο ***NAL_Ref_Idc*** είναι μόνιμα 1 τονίζοντας ότι όλα NAL Units έχουν την ίδια ανάγκη αποκωδικοποίησης. Ενώ το ποιο σημαντικό είναι ότι δεν δίνεται τιμή στο ***Temporal_id*** κάτι το οποίο σημαίνει ότι δεν υπάρχει ιεραρχία χρονικού στρώματος μεταξύ των NAL Units. Παρόλα αυτά ο κωδικοποιητής λειτουργεί σύμφωνα με το πρότυπο H.264/MVC .

Πίνακας vi. Base View – H.264/AVC NALU Header (Nokia Encoder)

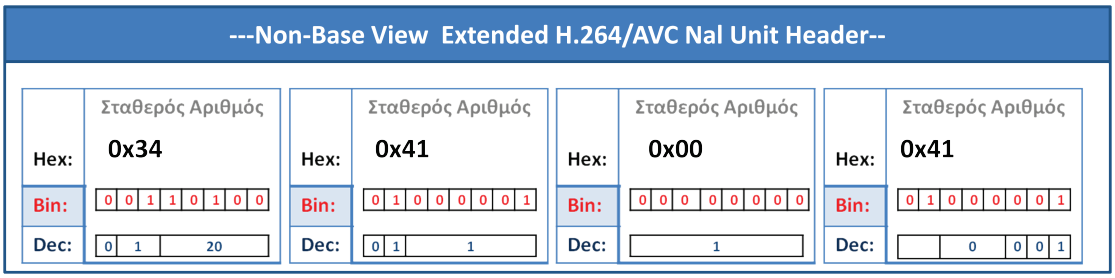


Ο Πίνακας vi απεικονίζει τις τιμές πού δέχεται ο H.264/AVC NALU Header ο οποίος ακολουθεί πάντα τον Prefix NalU Header. Όταν το ωφέλιμο φορτίο του NALU ενθυλακώνει μία IDR εικόνα (η μέρος της) τότε ο H.264/AVC NALU Header έχει ως τιμή την 0x25 αντίθετα έχει ως τιμή την 0x21.

Και στις 2 περιπτώσεις, η τιμή του **F-*Forbitten_Zero_Bit*** είναι 0, οπότε σε κανένα NALU δεν εμπεριέχεται παραβίαση της σύνταξης ή λάθος στα bit του ωφέλιμου φορτίου του. Επίσης, το **NRI-*NAL_Ref_Idc*** είναι 1 για όλα τα NALUs, το οποίο υποδηλώνει ότι υπάρχει η ίδια αναγκαιότητα αποκωδικοποίησης όλων των NALUs.

Τέλος το πεδίο **Type-*NAL_Unit_type*** για τις IDR εικόνες είναι 5 ενώ για τις υπόλοιπες είναι 1. Και οι δύο περιπτώσεις υποδεικνύουν ότι το NALU ενθυλακώνει μέρος του ωφέλιμου φορτίου της βασικής όψης . Επίσης επισημαίνουν ότι το NALU αναφέρεται στο παλαιού τύπου SPS (Sequence Paramer Set) τύπου 7 [6]. Η διαφοροποίηση των τιμών του πεδίου του τύπου για τις IDR εικόνες από τις υπόλοιπες θα μπορούσε να χρησιμοποιηθεί τόσο κατά την αποκωδικοποίηση όσο και κατά την μετάδοση για την εύρεση και την καλύτερη προστασία των IDR Frames.

Πίνακας vii. Non Base View – Coded Slice of MVC Extension (Nokia Encoder)



Οι τιμές των πεδίων του δεύτερου NAL Unit Header (coded slice of MVC extension) της μη-βασικής όψης είναι πάντα ίδιες ενώ επισημαίνουν ότι το NALU αναφέρεται στο καινούργιου τύπου SPS (Subset Sequence Paramer Set) τύπου 15 [6].

F-*Forbitten_Zero_Bit (0)*: Τα NAL Units δεν περιέχουν παραβίαση της σύνταξης ή λάθος στα bit του ωφέλιμου φορτίου τους.

NRI-*NAL_Ref_Idc (1)*: Υπάρχει αναγκαιότητα αποκωδικοποίησης όλων των NAL Units.

Type-*Nal_Unit_Type (20)*: Ορίζεται ο τύπος του NALU Header (Coded slice of MVC extension).

R-*Reserverd_One_bit (0)*: Αγνοείται απτούς αποκωδικοποιητές [Σταθερή τιμή].

I-*Idr_flag (1)*: Όλες οι εικόνες θεωρούνται ως V-IDR.

PRID -*Priority_id (1)*: Όλα τα πλαίσια έχουν την ίδια προτεραιότητα.

VID-*View_id (1):* Ορίζεται ως μη-βασική όψη.

TID-*Temporal_id (0)*: Δεν καθορίζεται ιεραρχία χρονικού στρώματος (temporal level).

A-*Anchor_pic_flag (0)*: Κανένα NALU δεν ενθυλακώνει ωφέλιμο φορτίο μιας εικόνας 'άγκυρας'.

V-*Inter_View_Flag (0)*: Κανένα NAL Unit δεν χρησιμοποιείται για inter-view πρόβλεψη

V-*Inter_View_Flag (1)*: Αγνοείται απτούς αποκωδικοποιητές.

2.4. Διόρθωση Σφαλμάτων Μετάδοσης σε επίπεδο εφαρμογής (Error Concealment)

Κατά την μετάδοση MVC ακολουθιών σε απωλέσθηκα κανάλια η ροή των bit του MVC έχει την δυνατότητα της ανεκτικότητας σε σφάλματα μετάδοσης. Αυτό γίνεται εφικτό μέσω των εργαλείων που παρέχονται από το H.264/MVC [9] και αναλύονται σε αυτήν την ενότητα.

Μια από τις ποίο απλές μεθόδους απόκρυψης λάθους είναι η αντιγραφή πλαισίου(error concealment of frame copy). Σε αυτόν τον αλγόριθμο μία χαμένη εικόνα χειρίζεται σαν να είναι ένα P-πλαίσιο και η διαδικασία κατασκευής της λίστας των εικόνων αναφοράς επικαλείται να δημιουργήσει μια νέα λίστα εικόνων αναφοράς. Εισάγεται δηλαδή σαν πρώτη εικόνα αναφοράς στην λίστα η εικόνα η οποία είναι η προηγούμενη εικόνα από την εικόνα που χάθηκε. Έπειτα, γίνεται αντιγραφή των τιμών των εικονοστοιχείων της πρώτης εικόνας στις τιμές των εικονοστοιχείων της χαμένης εικόνας. Ο συγκεκριμένος αλγόριθμος χρησιμοποιείτε στα 'πειραματικά αποτελέσματα' της συγκεκριμένης πτυχιακής για την αναπλήρωση των χαμένων πλαισίων.

Μία άλλη ποιο πολύπλοκη μέθοδος είναι η παραγωγή χρονικού διανύσματος κίνησης (Temporal direct motion vector generation). Αυτή η μέθοδος αναγκαστικά χρησιμοποιεί την λειτουργία της παρεμβολής(interpolation mode) ενώ συνήθως χρησιμοποιεί πληροφορία της ίδιας όψης. Για κάθε χαμένο macro-block(MB), τα διάνυσμα κίνησης και οι αναφορές του παράγονται σαν να είχε κωδικοποιηθεί με χρήση χρονικής άμεσης λειτουργίας(Temporal direct mode). Η χρονική λειτουργία υπολογίζει το διάνυσμα της εμπρός κίνησης του χαμένου block μέσω της πρώτης εικόνας αναφοράς της λίστας. Στην περίπτωση του nokia κωδικοποιητή όπου δεν παράγονται B-πλαίσια γίνεται χρήση των P – πλαισίων.

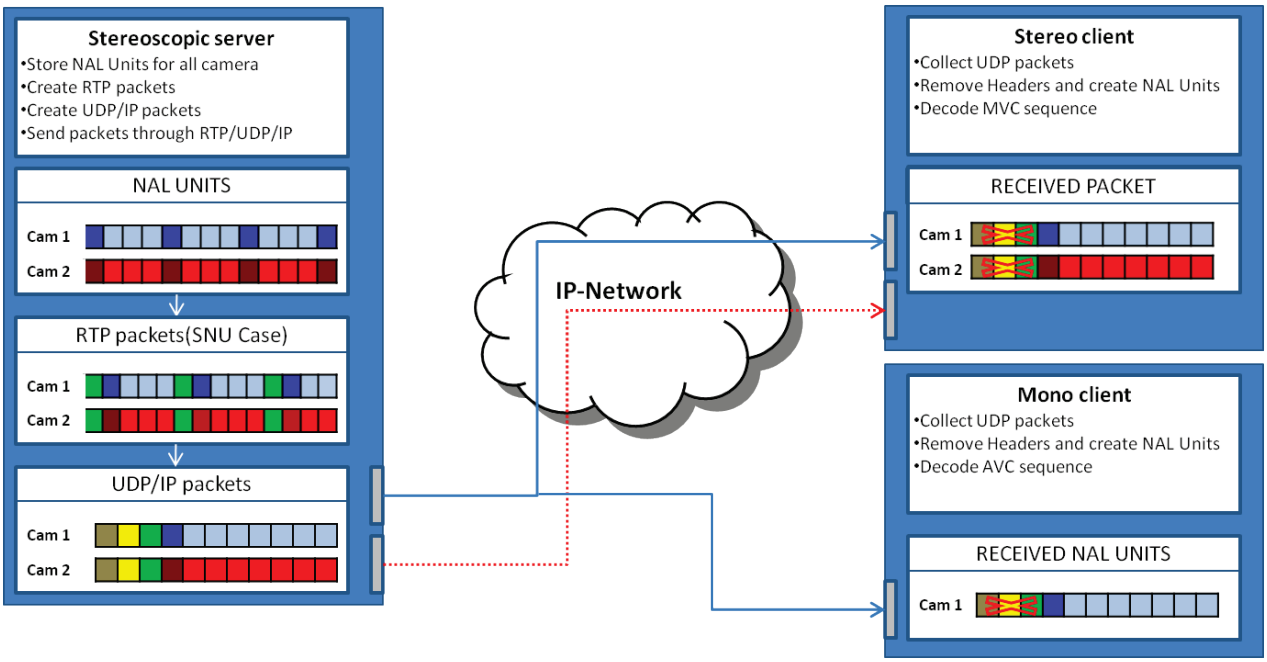
Η Τρίτη και ποιο αποδοτική μέθοδος ονομάζεται πρόβλεψη κίνησης(Motion Prediction). Για κάθε εικόνα άγκυρας της βασικής όψης κωδικοποιείται η ολική άνιση κίνηση(global disparity motion). Όταν ο αποκωδικοποιητής αντιληφθεί την απώλεια ενός πλαισίου, βρίσκει το αντίστοιχο MB (CMB) για κάθε MB του χαμένου πλαισίου μέσω μίας εξαρτώμενης όψης σύμφωνα με την ολική άνιση κίνηση. Τα διανύσματα κίνησης και λειτουργιών του CMB έπειτα αντιγράφονται για το χαμένο MB. Τελικά γίνεται συμψηφισμός κίνησης (**motion compensation**) για την παραγωγή της «κρυμμένης» εικόνας.

Υπάρχουν περιπτώσεις κάποιων ειδικών MBs ή ολόκληρων τεμαχίων όπου αυτός ο αλγόριθμος δεν λειτουργεί, οπότε είναι αναγκαία η χρήση άλλων μεθόδων στον αλγόριθμο MP που καθιστούν εφικτή την «απόκρυψη» της χαμένης πληροφορίας. Εάν το CMB δεν περιέχει πληροφορίες κίνησης (intra coded) τότε εφαρμόζεται χωρική απόκρυψη σφάλματος (spatial error concealment), ενώ εάν είναι μία εικόνα B τότε εφαρμόζεται χωρική άμεση λειτουργία (spatial direct mode).Όταν μία εικόνα άγκυρας μίας μη-βασικής όψης χαθεί, τότε αποκρύπτεται αντιγράφοντας τις αντίστοιχες τιμές από την συσχετισμένη εικόνα άγκυρας της βασικής όψης.

Κεφάλαιο 3ο. 3D Video Streaming

Σε αυτό το κεφάλαιο αναλύονται σε βάθος τα πρωτόκολλα τα οποία καθιστούν εφικτή την μετάδοση MVC ακολουθιών σε πραγματικό χρόνο. Ποιο συγκεκριμένα, αναλύεται τόσο το **R**eal-Time **T**ransport **P**rotocol (δομές ωφέλιμου φορτίου , τρόποι πακετοποίησης και μετάδοσης) όσο και τα streaming πρωτόκολλα (UDP/IP , DCCP/IP). Τελικά αναλύονται οι τρόποι με τους οποίους γίνεται εφικτή η διόρθωση σφαλμάτων μετάδοσης και αναλύεται ο μηχανισμός ο οποίος αναπτύχθηκε για την συγκεκριμένη πτυχιακή με την χρήση του οποίου είναι δυνατή η «διάσωση» όλων των χαμένων πλαισίων.

Στην Εικόνα 7 αναπαριστάται η διαδικασία μετάδοσης βίντεο σε πραγματικό χρόνο. Η διαδικασία της κωδικοποίησης και της ενθυλάκωσης των πλαισίων σε NAL Units έχει είδη περιγραφτεί στο Κεφάλαιο 2. Σε αυτό το κεφάλαιο θα περιγραφεί ο τρόπος με τον οποίο γίνεται η πακετοποίηση και μετάδοση της κωδικοποιημένης ακολουθίας.



Εικόνα 7. Μετάδοση και αναπαραγωγή τρισδιάστατης ακολουθίας βίντεο

3.1. Real-Time Transport Protocol

Σε αυτήν την ενότητα αναλύονται ο τρόποι με τους οποίους γίνεται η πακετοποίηση και η μετάδοση τρισδιάστατων ακολουθιών σύμφωνα με το **R**eal-Time **T**ransport **P**rotocol.

Το RTP αναπτύχτηκε από την οργάνωση δημιουργίας προτύπων IETF και χρησιμοποιείται σε συνδυασμό με άλλα πρωτόκολλα όπως και το H.264. Το πρότυπο του RTP ορίζει δύο πρωτόκολλα, το RTP (Real-Time Transport Protocol) και το RTCP (Real-Time Transport Control Protocol). Το RTP είναι το πρωταρχικό πρότυπο που χρησιμοποιείται για την μετάδοση πολυμέσων μέσω IP δικτύων παρέχοντας πληροφορίες όπως σφραγίδες χρόνου (timestamp) και αριθμούς ακολουθίας. Είναι σχεδιασμένο για πραγματικού χρόνου μεταδόσεις πολυμέσων από «άκρο-σε-άκρο» ενώ δίνει την δυνατότητα της μετάδοσης δεδομένων σε πολλαπλούς προορισμούς μέσω της πολύ-εκπομπής(multicasting). Το RTCP χρησιμοποιείται για περιοδική αποστολή πληροφοριών ελέγχου (QoS παραμέτρων) και περιέχει κανάλι ανάδρασης (feedback) παρέχοντας συγχρονισμό μεταξύ των ακολουθιών. Το εύρος ζώνης του RTCP είναι περίπου 5% (συγκριτικά με το RTP).

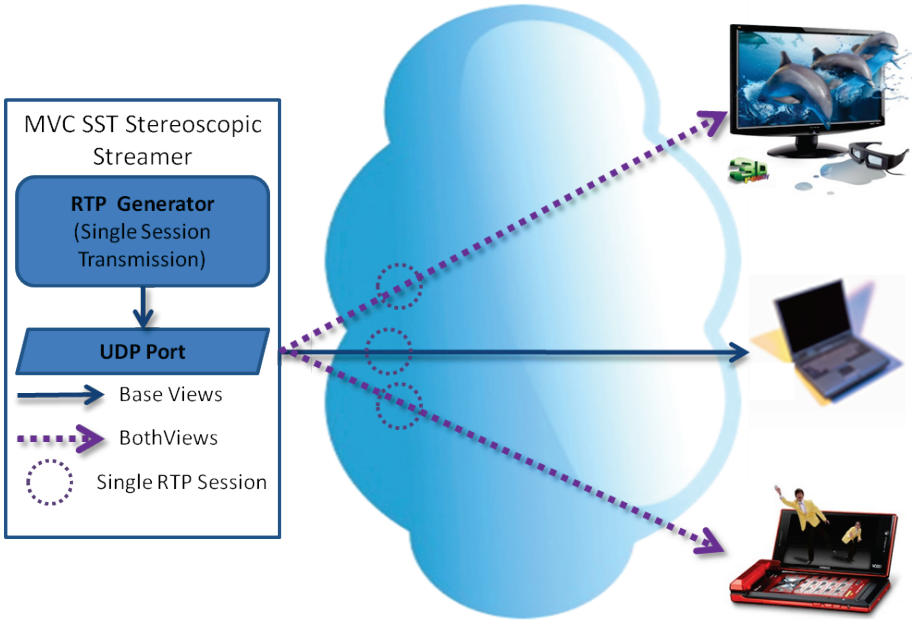
3.1.1. Τρόποι Μετάδοσης / Πακετοποίησης του RTP

Αυτή η ενότητα περιγράφει τους δυνατούς τρόπους μετάδοσης H.264/MVC ακολουθιών και τους δυνατούς τρόπους πακετοποίησης σύμφωνα με το RTP για το H.264/MVC. Οι τρόποι μετάδοσης και πακετοποίησης που ορίζονται από το πρότυπο του RTP για το H.264/MVC [4] είναι παρόμοιοι με αυτούς του H.264/SVC [8].

Μια RTP ροή δεδομένων μπορεί να περιέχει έναν από τους παρακάτω συνδυασμούς [4]:

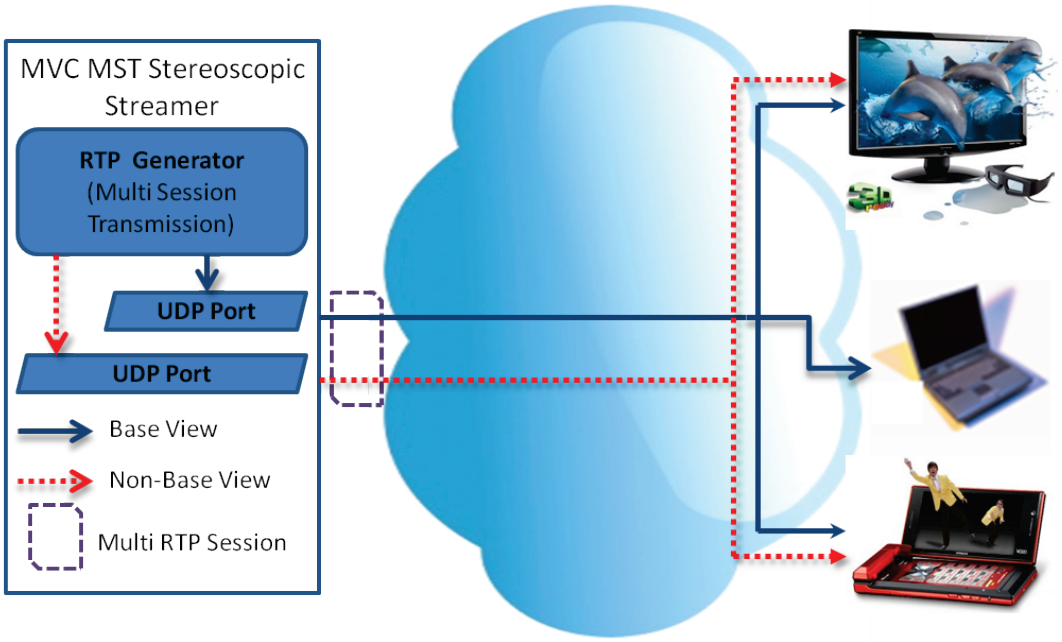
i. Την βασική όψη
ii. Μία ή περισσότερες όχι-βασικές όψεις
iii. Την βασική όψη παράλληλα με μία ή περισσότερες μη-βασικές όψεις.

Στην περίπτωση μετάδοσης με χρήση ενιαίας συνόδου (**S**ingle **S**ession **T**ransmission), ο διακομιστής δημιουργεί μία ροή δεδομένων για την μεταφορά όλων των MVC δεδομένων (NALUs) μέσω 'unicast' μετάδοσης (μία διεύθυνσης και πόρτας). Η προσαρμοσμένη RTP ροή δεδομένων δημιουργείται μέσω του 'single RTP session generator' ο οποίος συσσωρεύει όλα τα NALUs των κατάλληλων όψεων που είναι σε θέση (και θέλει) να λάβει το εκάστοτε τερματικό. Η αναπαράσταση του SST γίνεται στην Εικόνα 8 ενώ οι συνδυασμοί οι οποίοι επιτρέπετε να χρησιμοποιηθούν στο SST είναι οι (i) , (iii) αφού δεν είναι δυνατή η μετάδοση και αποκωδικοποίηση μόνο των μη-βασικών όψεων [6]. Εφόσον τα νέα NAL Units που εισήχθηκαν από το MVC όπως το coded slice of MVC extension αγνοούνται από τους H.264/AVC αποκωδικοποιητές, υπάρχει η δυνατότητα ενθυλάκωσης αυτών των νέων NALUs στην ίδια RTP ροή πακέτων κρατώντας την συμβατότητα προς τα πίσω.



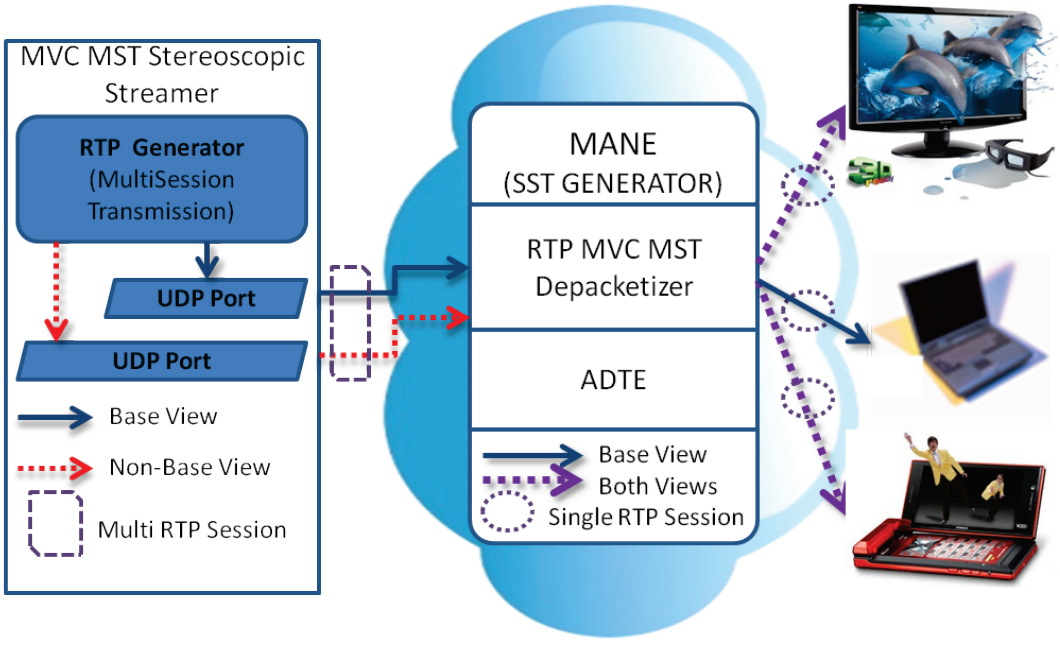
Εικόνα 8. Single Session MVC Transmission

Στην περίπτωση μετάδοσης με χρήση πολλαπλών συνόδων (**M**ulti **S**ession **T**ransmission) ο διακομιστής μεταφέρει τις όψεις (NALUs) μέσω διαφορετικών RTP συνόδων (διαφορετική διεύθυνση ή πόρτα). Η κάθε RTP σύνοδος πρέπει να έχει το ίδιο SSRC[4]. Κάθε όψη μεταδίδεται μέσω της δικιάς της IP multicast ομάδας (multicast IP διεύθυνση) ενώ τα τερματικά συντονίζονται μόνο στις επιθυμητές όψεις που θέλουν να λάβουν χρησιμοποιώντας συνήθως το IGMP (Internet group management protocol) πρωτόκολλο. Ο διακομιστής, για κάθε RTP ροή δεδομένων πρέπει να ανοίξει νέα UDP (User Diagram Protocol) πόρτα, οπότε ο αριθμός των πορτών είναι ανάλογος με τον αριθμό των όψεων που είναι προς αποστολή εκτός εάν μία RTP ροή ενθυλακώνει περισσότερες από μία μη-βασικές όψεις. Η αναπαράσταση του MST γίνεται στην Εικόνα 9 όπου ο διακομιστής κάνει χρήση multicast μετάδοσης. Η κάθε RTP σύνοδος μπορεί να μεταφέρει δεδομένα χρησιμοποιώντας οποιονδήποτε από τους τρείς συνδυασμούς (i),(ii),(iii) [6].



Εικόνα 9. Multi Session MVC Transmission

Ο MST εάν και είναι πλήρως υλοποιήσιμος σε ακαδημαϊκά και ερευνητικά περιβάλλοντα είναι δύσκολο να λειτουργήσει σε πραγματικό δίκτυο το οποίο περιέχει πρακτικούς περιορισμούς (NAT πρωτόκολλο ,firewalls). Αυτός είναι ο λόγος της αναγκαιότητας ύπαρξης ενός ενδιάμεσου σταθμού ο οποίος ονομάζεται MANE (**M**edia-**A**ware **N**etwork **E**lement) και απεικονίζεται στην Εικόνα 10. Ο διακομιστής λειτουργεί όπως στην περίπτωση του MST χρησιμοποιώντας τόσες RTP συνόδους όσες είναι για οι διαφορετικές όψεις, αλλά ο αποδέκτης τους δεν είναι ο πελάτης αλλά το MANE. Στο MANE γίνεται το ξεπακετάρισμα των RTP MST πακέτων και στην συνέχεια γίνεται χρήση ενός ADTE (**A**daptation **D**ecision **T**aking **E**ngine) μηχανισμού για την εύρεση των απαιτήσεων του χρήστη. Τέλος οι όψεις οι οποίες είναι προς αποστολή συσσωρεύονται σε μία RTP σύνοδο για την αποστολή τους με χρήση SST σε μία διεύθυνση και πόρτα. Όλη την πληροφορία την οποία χρειάζεται το MANE την παίρνει είτε μέσω σηματοδοσίας είτε μέσω των RTP και των NAL Unit Headers.Το σενάριο του MANE υπερβαίνει τα συνηθισμένους δρομολογητές αφού πρέπει να έχει επίγνωση της σηματοδοσίας.



Εικόνα 10. MANE-based system

Υπάρχουν 3 βασικοί τρόποι πακεταρίσματος κατά την χρήση του SST:

- Single NAL Unit mode
- Non-interleaved mode
- Interleaved mode

Ο single NAL Unit mode χρησιμοποιείται σε συστήματα τα οποία ορίζονται σύμφωνα με την σύσταση της ITU-T για το H.241. Αντίθετα, ο non-interleaved mode μπορεί να μην ορίζεται σύμφωνα με αυτήν την σύσταση ενώ τα NALUs πρέπει να μεταφέρονται με την σωστή σειρά αποκωδικοποίησης. Τέλος, ο interleaved τρόπος χρησιμοποιείται σε συστήματα που δεν απαιτούν χαμηλή καθυστέρηση και επιτρέπει την μετάδοση των NALUs χωρίς την σωστή σειρά αποκωδικοποίησης. Όλα τα NALUs με τύπο από 1 έως 29 μπορούν να μεταφερθούν χρησιμοποιώντας οποιονδήποτε από τους τρείς διαθέσιμους τρόπους πακετοποίησης.

Κατά την χρήση του MST η βασική όψη η οποία μεταφέρεται μόνη της μέσω μιας συνόδου πρέπει να πακεταριστεί σύμφωνα με τους τρείς παραπάνω τρόπους έτσι ώστε να κρατηθεί η συμβατότητα προς τα πίσω. Για το πακετάρισμα την υπόλοιπης MVC ακολουθίας (μη βασικές όψεις) έχουν εισαχθεί τέσσερις νέοι τρόποι πακεταρίσματος:

- Non-Interleaved Timestamp based Mode (NI-T)
- Non-Interleaved Cross-Layer Decoding Order Number(CS-DON) Based Mode(NI-C)
- Non-Interleaved Combined Timestamp and CS-DON Mode(NI-TC)
- Interleaved CS-DON based Mode (I-C)

Οι τέσσερις νέοι τρόποι πακεταρίσματος ξανά-χρησιμοποιούν τους τρείς βασικούς τρόπους πακεταρίσματος που περιγράφτηκαν παραπάνω και εισήχθηκαν από το H.264/AVC. Η βασική διαφορά των ανωτέρω είναι εάν επιτρέπεται ή όχι η παρεμβολή μεταξύ των NAL Units ,δηλαδή εάν επιτρέπεται η μεταφορά των NAL Units με τυχαία σειρά. Οι NI-T , NI-C και NI-TC τρόποι, αντίθετα με το I-C, επιτρέπουν την αποστολή των πακέτων με την σωστή σειρά αποκωδικοποίησης και για αυτό το λόγο χρησιμοποιούνται από συστήματα τα οποία απαιτούν μικρή καθυστέρηση. Η δεύτερη βασική διαφορά των διαθέσιμων τρόπων πακετοποίησης έχει να κάνει με τους μηχανισμούς που παρέχει ο κάθε ένας για την ανάκτηση μέσω των πολλαπλών συνόδων της σωστής σειράς αποκωδικοποίησης. Το NI-T χρησιμοποιεί σφραγίδες χρόνου για την ανάκτηση της σωστής σειράς αποκωδικοποίησης ενώ οι NI-C και I-C χρησιμοποιούν μηχανισμούς **C**ross **S**ession-**D**ecoding **O**rder **N**umber. Τέλος, το NI-TC παρέχει και τους δύο μηχανισμούς, ενώ ο μηχανισμός που θα εκτελεστεί τελικά επιλέγεται από τους δέκτες.

Ο τρόπος ο οποίος θα επιλεχθεί διέπει ποια σύνοδος του RTP επιτρέπεται , η οποία σύνοδος στην συνέχεια διέπει ποια NAL Units επιτρέπεται να είναι άμεσα χρησιμοποιούμενα σαν ωφέλιμο φορτίο ενός RTP πακέτου. Στα πλαίσια της συγκεκριμένης πτυχιακής χρησιμοποιείται ο **NI-T** σε μία MST μετάδοση ο οποίος επιτρέπει την μεταφορά και της βασικής και της μη-βασικής όψης κάνοντας χρήση του Non-Interleaved mode. Αυτός ο τρόπος επιτρέπει την ενθυλάκωση όλων των NAL Units που χρησιμοποιούνται από τον κωδικοποιητή της nokia και έχουν σαν τύπους από 1 έως 23 , 24(STAP-A) και 28(FU-A) σε RTP πακέτα [5]. Τέλος, όλες οι δομές ωφέλιμου φορτίου επιτρέπεται να χρησιμοποιηθούν και χρησιμοποιούνται.

Αναφορικά, ο μηχανισμός DON είναι ένας αριθμός ο οποίος υποδεικνύει την σωστή σειρά αποκωδικοποίησης και χρησιμοποιείται για την ανάκτηση της σωστής σειράς αποκωδικοποίησης όταν γίνεται χρήση της (interleaving) στο SST. Στην περίπτωση χρήσης του MST, ο μηχανισμός CS-DON είναι μία προέκταση του DON και χρησιμοποιείται για την ανάκτηση της σωστής σειράς αποκωδικοποίησης μέσω των διάφορων RTP συνόδων. Η διαφορά των ανωτέρω είναι ότι ο CS-DON χρησιμοποιείται ανεξάρτητα εάν χρησιμοποιείται (interleaving) ή όχι. Το εύρος τιμών του DON είναι από 0 έως 65535 ενώ εάν η τιμή του φτάσει την μέγιστη τότε επιστρέφει στην τιμή 0 [5].

3.1.2. RTP Header

Η εισαγωγή του RTP Header ο οποίος απεικονίζεται στον Πίνακα viii επιτρέπει ενθυλάκωση των NAL Units σε RTP πακέτα [7].

Πίνακας viii. RTP Header

RTP Header																															
Byte 1								Byte 2								Byte 3								Byte 4							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
V		P	X		CC			M		PT						Sequence number															
timestamp																															
synchronization source identifier(SSRC)																															
contributing source identifier(CSRC)																															

Τα πρώτα 12 Bytes πρέπει να βρίσκονται σε όλα τα RTP πακέτα , ενώ η λίστα των CSRC υπάρχει μόνο όταν η ακολουθία έχει εισαχθεί χρησιμοποιώντας έναν μείκτη [7]. Στην συνέχεια αναλύεται η σημασία των πεδίων του RTP Header [7].

V: 2 bits
Version: Προσδιορίζει την έκδοση του RTP.

P: 1 bit
Padding: Εάν οριστεί αυτό το πεδίο τότε το πακέτο περιέχει ένα ή περισσότερα επιπλέον byte στο τέλος του NAL unit τα οποία δεν είναι μέρος του ωφέλιμου φορτίου. Το τελευταίο byte του padding περιέχει τον αριθμό των padding bytes τα οποία πρέπει να αγνοηθούν, συμπεριλαμβανομένου και του δικού του.

X:1 bit
Extension: Εάν το πεδίο οριστεί τότε ο header πρέπει να ακολουθείται από μία επέκταση ενός επιπλέον header (RTP Header Extension) [7].

CC: 4 bits
CSRC count: Περιέχει τον αριθμό των αναγνωριστικών των CSRC οι οποίοι ακολουθούν τον header.

M: 1 bit
Marker: Η διερμηνεία του marker bit ορίζεται από ένα προφίλ. Επιτρέπει την είσοδο σημαδιού σε σημαντικά γεγονότα όπως τον καθορισμό των ορίων των πλαισίων ή των “access units”. Ένα προφίλ μπορεί να ορίσει περισσότερα marker bits ή να ορίσει ότι δεν υπάρχει marker bit αλλάζοντας τον αριθμό των bits στο πεδίο PT [7]. Κατά την χρήση του STAP ή MTAP το πεδίο πρέπει να έχει ως τιμή την τιμή που θα είχε το τελευταίο συσσωρευμένο NALU εάν είχε ενθυλακωθεί στο δικό του RTP πακέτο(SNU). Οι αποκωδικοποιητές μπορούν να χρησιμοποιήσουν αυτό το πεδίο σαν μία ένδειξη του τελευταίου πακέτου του “access unit”, αλλά θα πρέπει να μην βασίζονται σε αυτήν την ένδειξη.[5]

PT: 7 bits
Payload type: Αυτό το πεδίο ορίζει την μορφή του ωφέλιμου φορτίου του RTP και καθορίζει την ερμηνεία του από την εφαρμογή. Μία RTP πηγή μπορεί να αλλάξει την μορφή του ωφέλιμου φορτίου της κατά την διάρκεια μιας συνόδου αλλά δεν μπορεί με την χρήση αυτού του πεδίου να δημιουργεί πολυπλεξία ξεχωριστών ροών πολυμέσων. Ο δέκτης πρέπει να αγνοεί πακέτα τα οποία περιέχουν άγνωστα ‘payload types’.

Sequence number: 16 bits
Ο αριθμός ακολουθίας αυξάνει κατά ένα για κάθε RTP πακέτο δεδομένων και μπορεί να χρησιμοποιηθεί από τον δέκτη για την ανίχνευση της απώλεια ενός πακέτου. Η αρχική τιμή θα πρέπει να επιλέγεται τυχαία ώστε να γίνεται ποιο αποδοτική κρυπτογράφηση.

Timestamp: 32 bits
Οι σφραγίδες χρόνου αντικατοπτρίζουν την στιγμή της δειγματοληψίας του πρώτου byte του RTP πακέτου δεδομένων. Η στιγμή της δειγματοληψίας προέρχεται από το ρολόι το οποίο αυξάνει μονότονα και γραμμικά στο χρόνο ώστε να επιτρέπει τον συγχρονισμό και τον υπολογισμό του jitter. Η συχνότητα του ρολογιού εξαρτάται από τον τύπο των δεδομένων που μεταφέρονται και ορίζεται στατικά στο προφίλ ή στην προδιαγραφή του τύπου των δεδομένων ή μπορεί να ορίζεται και δυναμικά. Η αρχική τιμή πρέπει να είναι τυχαία όπως και στον αριθμό ακολουθίας για μεγαλύτερη ασφάλεια κατά την μετάδοση(κρυπτογράφηση). **Εάν δημιουργηθούν περισσότερα του ενός RTP πακέτα για ένα πλαίσιο τότε αυτά θα έχουν όλα την ίδια σφραγίδα χρόνου [7].**

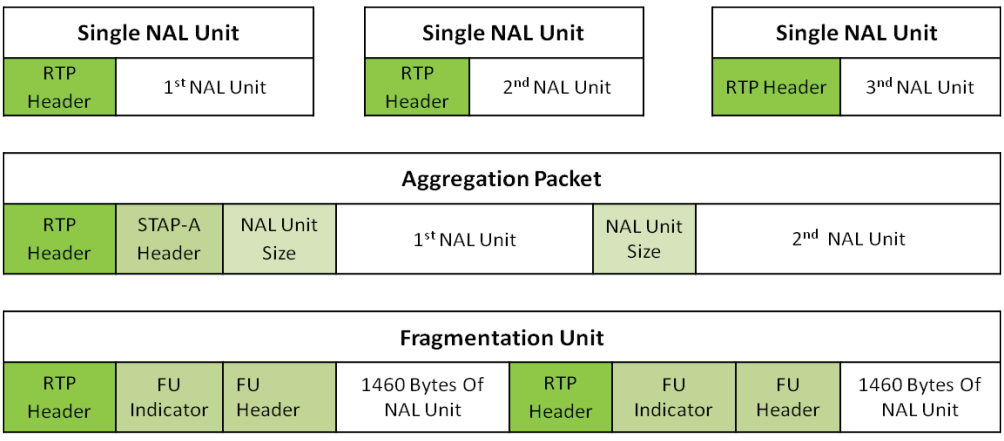
SSRC: 32 bits
Αυτό το πεδίο ορίζει την πηγή του συγχρονισμού. Το πεδίο πρέπει να επιλέγεται τυχαία έχοντας ως σκοπό οι διαφορετικές πηγές συγχρονισμού εντός της ίδιας RTP συνόδου να έχουν διαφορετικό SSRC [7].

CSRC list: 0-380 bits
Η CSRC λίστα προσδιορίζει τις συμβαλλόμενες πηγές για το ωφέλιμο φορτίο του πακέτου. Το πεδίο CC δείχνει τον αριθμό των αναγνωριστικών που ακολουθούν τα οποία μπορεί να είναι από κανένα έως 15 ενώ το καθένα έχει μήκος 32 bit [7].

3.1.3. Δομές Ωφέλιμου Φορτίου

Σύμφωνα με το πρότυπο του RTP [4], η ενθυλάκωση των NALUs σε RTP πακέτα γίνεται κάνοντας χρήση ενός εκ των τριών διαθέσιμων δομών ωφέλιμου φορτίου (payload structure). Οι εν λόγω δομές ονομάζονται Single NAL Unit, Aggregation Packet και Fragmentation Unit. Στην συγκεκριμένη πτυχιακή έχουν υλοποιηθεί, αναλύονται και συγκρίνονται και οι τρείς δομές ωφέλιμου φορτίου, ενώ η πρώτη δομή (Single NAL Unit) χρησιμοποιείται από τον κωδικοποιητή της nokia (εάν ζητηθεί να παράγει RTP πακέτα).

Όπως αναπαριστάται στην Εικόνα 11 , όταν γίνεται χρήση του Single NALU τότε το κάθε RTP πακέτο περιέχει ένα ολόκληρο NALU, όταν γίνεται χρήση Aggregation Packet τότε περισσότερα του ενός NALUs ενθυλακώνονται σε ένα RTP πακέτο και τέλος όταν γίνεται χρήση του Fragmentation Unit τότε μέρος ενός NALU ενθυλακώνεται σε ένα RTP πακέτο.



Εικόνα 11. Δομές ωφέλιμου φορτίου

Ο πίνακας ix συνοψίζει τους τύπους των NAL Unit που ορίζονται από το πρότυπο και τους αντίστοιχους τύπους των δομών ωφέλιμου φορτίου των RTP πακέτων[5],[6]. Κάθε NAL unit το οποίο ορίζεται στο H.264 (τύπου 1 έως 23) μπορεί είτε να ενθυλακωθεί κάνοντας χρήση του Single NAL Unit, είτε να συσσωρευτεί κάνοντας χρήση του Aggregation Packet είτε τέλος να κατακερματιστεί κάνοντας χρήση του Fragmentation Unit.

Για να επιτευχθεί η συσσώρευση ή ο κατακερματισμός των NAL Units διασφαλίζοντας ότι το ωφέλιμο φορτίο κάθε RTP πακέτου αποτελείται μόνο από NAL Units εισήχθηκαν 6 νέοι τύποι πακέτων(24-29) έτσι ώστε να χρησιμοποιηθούν σαν δομές ωφέλιμου φορτίου. Τα aggregation πακέτα υποδιαιρούνται σε τέσσερεις εκδόσεις χρησιμοποιώντας τέσσερις νέους τύπους οι οποίοι ισχύουν και για το AVC ενώ τα Fragmentation Units υποδιαιρούνται σε δύο εκδόσεις εισάγοντας με την σειρά τους δύο νέους τύπους πακέτων. Πρέπει να επισημανθεί ότι τόσο για το SVC όσο και για το MVC έχουν εισαχθεί δύο νέοι τύποι πακέτων με το όνομα PACSI NAL Unit και NI-MTAP NAL Unit. Το PASCI NAL Unit μπορεί να χρησιμοποιηθεί μόνο στο Single NAL Unit ενώ το NI-MTAP μπορεί να χρησιμοποιηθεί και στο Aggregation Packet.

Ο δέκτης κάνοντας χρήση του πρώτου byte του RTP πακέτου (έπειτα από τον RTP Header) μέσω του πεδίου ‘nal_u_type’ είναι σε θέση να αναγνωρίσει την δομή ωφέλιμου φορτίου που χρησιμοποιήθηκε. Το εν λόγω byte ανεξαρτήτου δομής δομείται όπως ένας H.264/AVC NAL Unit Header [6].

Πίνακας ix. NALU τύποι με τις δομές ωφέλιμου φορτίου τους

Type	Packet Type	Packet Type Name
0	reserved	– – –
1-13	NAL unit	Single NAL Unit packet
14	NAL Unit	Prefix NAL Unit packet
15-19	NAL unit	Single NAL Unit packet
20	NAL Unit	coded slice of MVC extension
21-23	NAL unit	Single NAL Unit packet
24	STAP-A	Single-time aggregation packet
25	STAP-B	Single-time aggregation packet
26	MTAP-16	Multi-time aggregation packet(2 byte offset)
27	MTAP-24	Multi-time aggregation packet(3 byte offset)
28	FU-A	Fragmentation unit
29	FU-B	Fragmentation unit
30	NAL unit	PACSI NAL Unit packet
31	NI-MTAP	NI-MTAP NAL Unit packet

3.1.3.1. Single NAL Unit

Το **S**ingle **N**AL **U**nit packet μπορεί να περιέχει μόνο ένα NAL Unit ενώ ο τύπος του πρέπει να είναι από 1 έως 23 [6] ή από 30 έως 31 [5]. Τα 3 πεδία τα οποία εμφανίζονται (F,NRI,type) είναι ο header του SNU ο οποίος βρίσκεται στην κωδικοποιημένη ακολουθία (H.264/AVC NAL Unit Header) και ακολουθεί έπειτα από τον Prefix NAL Unit Header.

Πίνακας x. Δομή Single NAL Unit Packet

RTP Header	Prefix NALU Header	F	NRI	Type
NAL Unit Bytes				
Optional RTP padding				

3.1.3.2. Aggregation Packet

Η δομή των Aggregation πακέτων δημιουργήθηκε για να λυθούν προβλήματα μεταξύ ετερογενών δικτύων (διαφορετικό μέγεθος **M**aximum **T**ransmission **U**nit) .

Κάθε NAL Unit το οποίο μεταφέρεται μέσω ενός aggregation πακέτου ενθυλακώνεται πρώτα σε ένα aggregation unit. Όπως έχει είδη προαναφερθεί, υπάρχουν πέντε τύποι aggregation πακέτων. Στα πλαίσια αυτής της πτυχιακή θα αναλυθεί μόνο ο STAP-A ο οποίος χρησιμοποιεί σφραγίδες χρόνου (timestamps) για την εύρεση της σωστής σειράς αποκωδικοποίησης. Πρέπει να επισημανθεί ότι το μέγιστο μέγεθος ενός NALU το οποίο ενθυλακώνεται σε ένα aggregation πακέτο είναι 65535 bytes [5].

Ο πίνακας xi αναπαριστά την δομή του **ωφέλιμου φορτίου** του RTP για τα aggregation πακέτα η οποία ανεξαρτήτου τύπου aggregation unit που θα χρησιμοποιηθεί δομείται πάντα με τον παρακάτω τρόπο.

Πίνακας xi . Δομή Aggregation packet

F	NRI	Type
One Or more aggregation units		
Optional RTP padding		

Παρακάτω αναλύονται οι βασικοί κανόνες οι οποίοι ισχύουν για όλους τους τύπους των aggregation πακέτων:

- Το πεδίο ‘timestamp’ πρέπει να οριστεί στο πρωταρχικό NALU.
- Το πεδίο του τύπου ‘Type’ μπορεί να πάρει ως τιμές τις (24) έως (27) και (31).
- Το πεδίο ‘F’ δεν ορίζεται εάν όλα τα πεδία F των NAL Units είναι 0 αλλιώς πρέπει να οριστεί.
- Η τιμή του ‘NRI’ πρέπει να περιέχει την μεγαλύτερη τιμή όλων των πεδίων NRI των NAL Units που συσσωρεύονται.
- Το πεδίο ‘market’ στον RTP header παίρνει σαν τιμή την τιμή πού θα είχε το τελευταίο NAL Unit του aggregated πακέτου εάν είχε μεταδοθεί μόνο του σε ένα Single NAL Unit RTP πακέτο.

Στην συνέχεια στον πίνακα xii περιγράφεται το Single time aggregation unit κάνοντας χρήση του STAP-A. Το πεδίο ‘NAL Unit Size’ αποτελείται από 2 byte υποδεικνύοντας το μέγεθος του NAL Unit που ενθυλακώνεται (bytes) συμπεριλαμβανομένων των headers .

Πίνακας xii. Δομή του Single-Time Aggregation unit

NAL Unit Size
NAL Unit

Συνοψίζοντας, ο πίνακας xiii απεικονίζει την πλήρη δομή ενός aggregation RTP πακέτου κάνοντας χρήση του STAP-A (δύο single-time aggregation units) .

Πίνακας xiii. Aggregation packet (STAP-A)

RTP Header		
STAP-A NAL HDR	**First NALU Size**	**First NALU Header**
FIRST NALU Bitstream		
Second NALU Size	**Second NALU Header**	
Second NALU Bitstream		
		Optional Padding

3.1.3.3. Fragmentation Unit

Αυτή η μορφή ωφέλιμου φορτίου επιτρέπει τον «τεμαχισμό» και την ενθυλάκωση ενός NAL Unit σε περισσότερα του ενός RTP πακέτα. Η χρήση του τεμαχισμού σε επίπεδο εφαρμογής και όχι σε επίπεδο IP έχει δύο πλεονεκτήματα [5]. Πρώτον, η μορφοποίηση του ωφέλιμου φορτίου επιτρέπει την μετάδοση των NAL Unit με μεγαλύτερο μέγεθος των 64 Kbytes πάνω από ένα IPv4 δίκτυο κάτι το οποίο είναι πολύ χρήσιμο ειδικά κατά την μετάδοση βίντεο υψηλής ανάλυσης(HD) όπου υπάρχει ένα όριο στον αριθμό των τεμαχίων (slices) ανά πλαίσιο (ύπαρξη μεγάλων NAL Units). Δεύτερον, ο «μηχανισμός του τεμαχισμού» επιτρέπει την εφαρμογή του μηχανισμού FEC ή ARQ μόνο σε εκείνα τα τεμάχια ενός Single NAL Unit τα οποία έχουν αλλοιωθεί κατά την μετάδοση. Ο τεμαχισμός ορίζεται μόνο για τα Single NAL Units και όχι για τα aggregation πακέτα ενώ ένα FU δεν μπορεί να εμφωλεύει άλλο FU.

Ο τεμαχισμός ενός NAL Unit αποτελείται από έναν ακέραιο αριθμό από διαδοχικά bytes ενός NAL unit. Κάθε byte ενός NAL Unit πρέπει να βρίσκεται μόνο σε ένα τεμάχιο του NAL Unit(Fragmentation Unit). **Τα τεμάχια ενός NAL Unit πρέπει να στέλνονται διαδοχικά με αύξοντες αριθμούς ακολουθίας(sequence numbers).**

Όπως έχει είδη προαναφερθεί, υπάρχουν δύο τύποι FU. Ο FU-A οποίος δεν χρησιμοποιεί DON και ο οποίος αναλύεται και χρησιμοποιείται στην συγκεκριμένη πτυχιακή, και ο FU-B ο οποίος διαφέρει μόνο στο ότι κάνει χρήση του DON μηχανισμού (εισαγωγή ενός επιπλέον header μήκους 2 byte).

Ο Πίνακας xiv αναπαριστά την μορφή της δομής ωφέλιμου φορτίου του FU-A. Το FU-A αποτελείται από δύο headers με το όνομα ‘fragmentation unit indicator’ και ‘fragmentation unit header’ μήκους ενός byte ο καθένας. Τα πεδία και των δύο header αναπαριστώνται στην Εικόνα 12 και αναλύονται στην συνέχεια.

Πίνακας xiv. Δομή του FU-A

RTP Header	FU indicator	FU header	
FU payload			
			Optional RTP padding

Bytes	0	1	2	3	4	5	6	7
FU indicator	F	NRI		Type				

Bytes	0	1	2	3	4	5	6	7
FU Header	S	E	R	Type				

Εικόνα 12. FU-A Headers

Τόσο το πεδίο F όσο και το NRI στο FU indicator πρέπει να έχουν ως τιμές αυτές τις οποίες έχει το H.264/AVC NAL Unit Header του τεμαχισμένου NAL unit. Το πεδίο Type παίρνει την τιμή 28 όταν χρησιμοποιείται FU-A και την τιμή 29 όταν χρησιμοποιείται FU-B.

S: 1 bit
Start_Bit: Πρέπει να ορίζεται μόνο το πρώτο τεμάχιο ενός NALU ενώ όλα τα υπόλοιπα τεμάχια πρέπει να έχουν την τιμή 0.

E:1 bit
End_Bit: Πρέπει να ορίζεται μόνο το τελευταίο τεμάχιο ενός NALU ενώ όλα τα υπόλοιπα τεμάχια πρέπει να έχουν την τιμή 0.

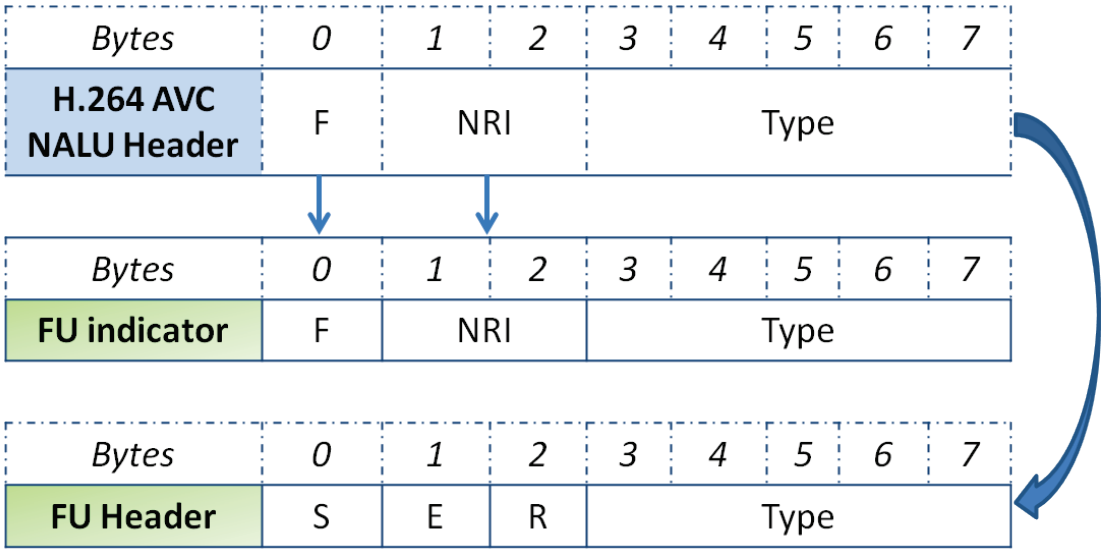
R: 1 bit

Reserved_Bit: Έχει πάντα την τιμή 0 και αγνοείται απτούς δέκτες.

Type: 5 bits
Ορίζει το τύπο του ωφέλιμου φορτίου που μεταφέρει το NAL Unit. Δέχεται την τιμή του πεδίου type του H.264/AVC NAL Header.

Ο H.264/AVC NAL Header του τεμαχισμένου NAL Unit δεν συμπεριλαμβάνεται στο fragmentation unit, ως εκ τούτου ούτε στο ωφέλιμο φορτίο, αλλά τα πεδία F και NRI του H.264/AVC NAL Header μεταφέρονται στο FU indicator και το πεδίο type μεταφέρεται στο FU header. Η Εικόνα 13 αναπαριστά την διαδικασία μεταφοράς των τιμών του H.264/AVC NALU Header στους FU-A headers.

Κατά την διαδικασία του «ξεπακεταρίσματος» ο H.264/AVC NAL Header πρέπει να ανακατασκευαστεί μέσω των hearer του FU. Τόσο στην περίπτωση του MVC όσο και του SVC όπου γίνεται χρήση του εκτεταμένου H.264/AVC NAL Header συνολικού μήκους 4 bytes,τα 3 bytes συνεχίζουν να βρίσκονται στο ωφέλιμο φορτίο του NAL Unit, ενώ όπως είναι προφανές τοποθετούνται μόνο στο πρώτο τεμάχιο του τεμαχισμένου NAL Unit.



Εικόνα 13. Μεταφορά τιμών του H.264/AVC NALU Header στους FU-A headers

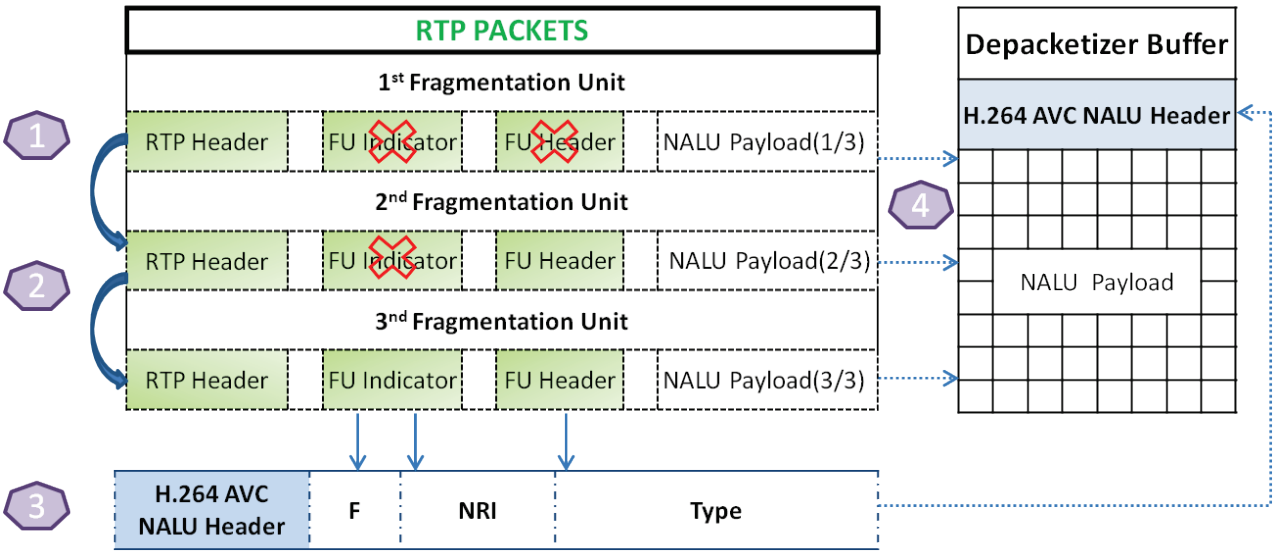
3.1.4. Μηχανισμός διαχείρισης λαθών στο RTP επίπεδο(Error Resilience)

Προκειμένου να αντιμετωπιστεί η απώλεια των αναγκαίων προς την αποκωδικοποίηση headers, ένας RTP αλγόριθμος ανάκαμψης από σφάλματα βασισμένος στην FU δομή ωφέλιμου φορτίου υλοποιήθηκε στον RTP depacketizer. Ο εν λόγω μηχανισμός είναι σε θέση να ανακατασκευάσει (σε επίπεδο εφαρμογής) **όλους** τους χαμένους NALU headers και να επιτρέψει στον αποκωδικοποιητή να αναγνωρίσει και να αποκωδικοποιήσει όλα τα πλαίσια (NALUs) του βίντεο και των δύο όψεων. Ως εκ τούτου, ο προτεινόμενος αλγόριθμος βελτιώνει σημαντικά την αντιληπτή ποιότητα του αποκωδικοποιημένου βίντεο.

Ποιο συγκεκριμένα, απώλειες πλαισίων συμβαίνουν συχνότερα όταν ένα ολόκληρο πλαίσιο ενθυλακώνεται μόνο σε ένα NALU. Σε αυτήν την περίπτωση, ο H.264/AVC NALU header ενθυλακώνεται μόνο σε ένα IP πακέτο. Όταν ένα τέτοιο πακέτο χαθεί, ο αποκωδικοποιητής δεν είναι σε θέση να αναγνωρίσει τον τύπο του NALU , έχοντας ως αποτέλεσμα την απόρριψη όλου του NALU(πλαισίου) .

Ο προτεινόμενος αλγόριθμος ο οποίος αναπαριστάται στην Εικόνα 14 βασίζεται στο γεγονός ότι τα αναγκαία πεδία του H.264/AVC Header βρίσκονται σε όλα τα FU's ενός NALU και περιλαμβάνει τα ακόλουθα βήματα:

1. Όταν ο de-packetizer ανακαλύπτει ότι το πρώτο τεμάχιο του NALU λείπει.
2. Ψάχνει στα επόμενα τεμάχια του ίδιου NALU να ανακαλύψει ένα τεμάχιο το οποίο να έχει μεταδοθεί χωρίς σφάλματα.
3. Μετά την ανακάλυψη ενός τέτοιου τεμαχίου, αντιγράφει τις αναγκαίες τιμές από τα πεδία του "FU Indicator" και του "FU header" και ανακατασκευάζει τον αναγκαίο για την αποκωδικοποίηση H.264/AVC NALU header.
4. Τέλος, ο de-packetizer απορρίπτει τους υπόλοιπους RTP headers και αποθηκεύει το ωφέλιμο φορτίο.



Εικόνα 14. Προτεινόμενος μηχανισμός διαχείρισης λαθών

3.2. Streaming πρωτόκολλα

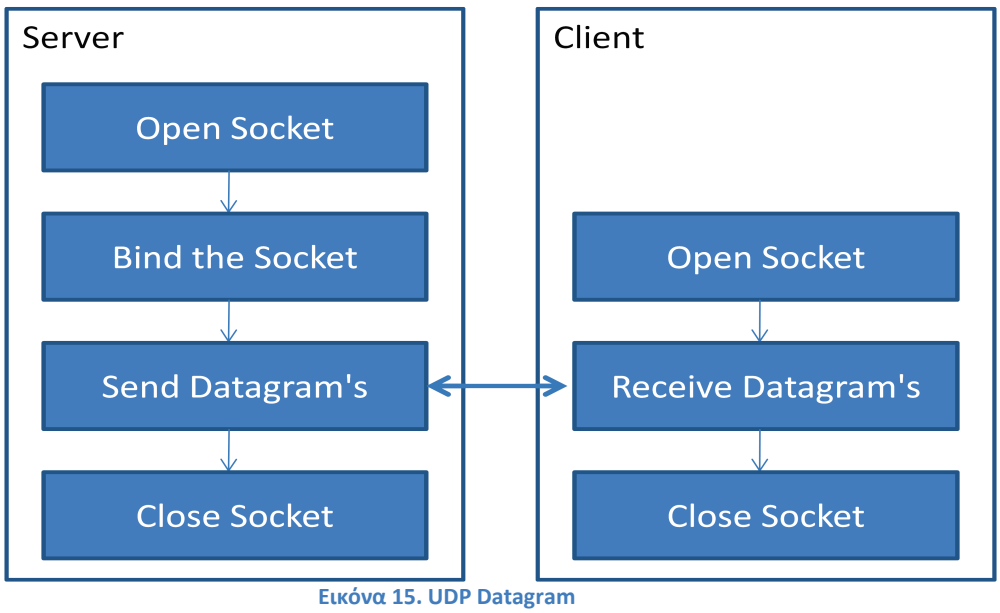
Το βασικό πρωτόκολλο συνεχής ροής MVC βίντεο το οποίο είναι η τεχνολογία αιχμής είναι το **RTP/UDP/IP**(User Datagram Protocol). Το RTP πάνω από **DCCP/IP** (Datagram Congestion Control Protocol) είναι το πρωτόκολλο επόμενης γενιάς ενώ το **TCP/IP** εάν και έχει τυποποιηθεί για την χρήση του με το RTP, δεν χρησιμοποιείται συχνά λόγο της μεγάλης καθυστέρησης που δημιουργεί η εγκαθίδρυση σύνδεσης και η διόρθωση λαθών.

Υπάρχουν τέσσερις Αρχιτεκτονικές 3DTV streaming [2]:

1. Server unicasting (σε ένα τερματικό)
2. Server multicasting (σε πολλαπλά τερματικά)
3. Peer-to- Peer (P2P) unicasting (το κάθε τερματικό προωθεί τα πακέτα του σε άλλο τερματικό)
4. P2P multicasting(το κάθε τερματικό προωθεί τα πακέτα του σε πολλαπλά τερματικά)

3.2.1. UDP over IP

Το UDP/IP δεν εμπεριέχει κάποιο μηχανισμό ελέγχου συμφόρησης και δεν εγγυάται αξιόπιστη επικοινωνία [2]. Στο MVC, ο μεγάλος όγκος της προς αποστολής πληροφορίας μπορεί να προκαλέσει κατάρρευση εφόσον δεν γίνεται έλεγχος συμφόρησης. Παρόλα αυτά, τόσο οι εφαρμογές ήχου όσο και οι εφαρμογές βίντεο (streaming) προτιμούν αυτό το πρωτόκολλο αφού για τις εφαρμογές αυτές το σημαντικότερο είναι τα πακέτα να παραδοθούν στον παραλήπτη σε μικρό χρονικό διάστημα ώστε να μην υπάρχει διακοπή στην ροή. Πρέπει να τονιστεί ότι οι εφαρμογές οι οποίες χρησιμοποιούν το πρωτόκολλο UDP θα πρέπει να είναι σε θέση να αγνοούν κάποια αλλοίωση των πακέτων ή ακόμα και ολόκληρες απώλειες πακέτων ανάλογα με τις συνθήκες που επικρατούν στο κανάλι μεταφοράς. Κατά την μετάδοση, τα διαγράμματα δεν αριθμούνται, οπότε ο δέκτης τα λαμβάνει με τυχαία σειρά. Η Εικόνα 15 αναπαριστά ένα βασικό διάγραμμα ενός διακομιστή και ενός χρήστη (unicasting) οι οποίοι συνδέονται μέσω ενός socket και μεταφέρουν τα πακέτα (datagram) τους μέσω αυτών.



Εικόνα 15. UDP Datagram

3.2.2. DCCP over IP

Το datagram congestion control protocol (DCCP) σχεδιάζεται για την αντικατάσταση του UDP και «τρέχει» απευθείας πάνω από το Internet Protocol (IP) για να παρέχει έλεγχο συμφόρησης αλλά χωρίς πλήρη αξιοπιστία [2]. Το DCCP μπορεί να αναπαρασταθεί ως TCP χωρίς αξιοπιστία και μεταφορά δεδομένων σε σειρά , ή σαν UDP κάνοντας χρήση έλεγχου συμφόρησης, εγκατάσταση σύνδεσης και βεβαιώσεων σωστής λήψης(acknowledgements).

Το DCCP/IP είναι ένα πρωτόκολλο μεταφοράς που υλοποιεί αμφίδρομες συνδέσεις unicast με χρήση ελέγχου συμφόρησης σε αναξιόπιστα δίκτυα. Παρά την αναξιόπιστη ροή πακέτων, το DCCP παρέχει αξιόπιστες χειραψίες για την εγκαθίδρυση και την διαπραγμάτευση των χαρακτηριστικών τις σύνδεσης ενώ επίσης παρέχει μηχανισμούς τμηματικού ελέγχου συμφόρησης. Στο DCCP ορίζονται δύο μηχανισμοί ελέγχου συμφόρησης , ένας εκ των οποίων επιλέγεται κατά την εγκαθίδρυση της σύνδεσης. Ονομαστικά είναι το **TCP like Congestion Control** και το **TCP-Friendly Rate Control (TFRC).**

Το TCP-like Congestion Control προσδιορίζεται από το Control Identifier 2 (CCID2) στο DCCP, και συμπεριφέρεται παρόμοια με το μηχανισμό ελέγχου συμφόρησης AIMD (**A**dditive **I**ncrease / **M**ultiplicative Decrease) του TCP. Ο εν λόγω μηχανισμός διχοτομεί το παράθυρο έπειτα από την απώλεια ενός πακέτου. Εφαρμογές που χρησιμοποιούν αυτόν τον μηχανισμό ελέγχου συμφόρησης θα ανταποκριθούν γρήγορα στις αλλαγές του διαθέσιμου εύρους ζώνης , αλλά θα πρέπει να είναι ανεχτικές στις απότομες αλλαγές του μεγέθους του παραθύρου συμφόρησης.

Αντίθετα το TFRC προσδιορίζεται από το Congestion Control Identifier 3 (CCID3) και είναι μια μορφή ελέγχου ροής βασισμένη σε εξισώσεις που ελαχιστοποιούν τις απότομες αλλαγές ρυθμού μετάδοσης. Είναι καταλληλότερο για εφαρμογές πού προτιμούν έναν ποιό ομαλό ρυθμό μετάδοσης, συμπεριλαμβανομένων των εφαρμογών μετάδοσης πολυμέσων με μικρό ή μέτριο buffer στον δέκτη. Το CCID3/TFRC υπολογίζει ένα επιτρεπόμενο ποσοστό αποστολής (ρυθμός TFRC) χρησιμοποιώντας την εξίσωση απόδοσης του TCP η οποία παρέχεται στην εφαρμογή αποστολέα κατόπιν αιτήσεως. Ο αποστολέας μπορεί να χρησιμοποιήσει αυτές τις πληροφορίες ρυθμού για να προσαρμόσει τον ρυθμό μετάδοσης του.

3.2.3. Διόρθωση σφαλμάτων μετάδοσης (Error Correction)

Οι εφαρμογές streaming «υποφέρουν» από απώλειες πακέτων εφόσον τα περισσότερα πρωτόκολλα απορρίπτουν ολόκληρο το πακέτου όταν αυτό έχει αλλοιωθεί [2]. Ειδικά στην περίπτωση ασύρματων συνδέσεων, η ύπαρξη διαφόρων ειδών θορύβων και παρεμβολών στο είδη υπάρχων περιορισμένο εύρος ζώνης προκαλεί ιδιαίτερα αυξημένη συμφόρηση. Έχουν αναπτυχθεί διάφορες τεχνικές αποδοτικής μετάδοσης ακολουθιών βίντεο κάποιες εκ των οποίων αναλύονται σε αυτήν την ενότητα (**A**utomatic **R**epeat **R**equest , **F**orward **E**rror **C**orrection , **F**lexible **M**acroblock **O**rdering).

Η χρήση του ARQ απαιτεί μηνύματα ανάδρασης(ACK) τα οποία πληροφορούν τον διακομιστή για την σωστή λήψη του πακέτου. Αυτός ο μηχανισμός είναι αποτελεσματικός στις απώλειες πακέτων αλλά προκαλεί πολύ μεγάλη καθυστέρηση. Οπότε είναι προτιμότερη η χρήση περιορισμένου χρόνου ARQ σε επίπεδο εφαρμογής (UDP ή DCCP) επιτρέποντας την αναμετάδοση μόνο εντός μιας περιορισμένης χρονικής περιόδου από την χρήση του TCP όπου γίνεται απεριόριστη χρήση του ARQ σε επίπεδο δικτύου.

Σε περίπτωση όπου δεν είναι δυνατή η χρήση καναλιού ανάδρασης, μπορεί να γίνει χρήση τεχνικών κωδικοποίησης καναλιών όπως το εργαλείο FMO του H.264/AVC το οποίο ταξινομεί τα Macro block των τεμαχίων σε ομάδες άνισης αναγκαιότητας. Οι συστηματικοί LT κώδικες χρησιμοποιούνται για την πρόβλεψη λαθών λόγω της χαμηλής πολυπλοκότητας τους και της υψηλής απόδοσής τους. Ο βέλτιστος τρόπος ομαδοποίησης των τεμαχίων και της κατανομής του ρυθμού των καναλιών καθορίζονται από έναν επαναληπτικό αλγόριθμο βελτιστοποίησης βασισμένο στο δυναμικό προγραμματισμό [2].

Μια εξελιγμένη χρήση FEC μηχανισμού σε στερεοσκοπικό βίντεο γίνεται ταξινομώντας τα πλαίσια σύμφωνα με τη συμβολή τους στη γενική ποιότητα. Η απώλεια ενός I-frame προκαλεί μεγάλες διαστρεβλώσεις λόγω της διάδοσης λάθους, για αυτόν τον λόγο τα I-frames πρέπει προστατεύονται περισσότερο. Τα P-frames της βασικής ακολουθίας είναι σημαντικότερα δεδομένου ότι μπορούν να κωδικοποιηθούν χωρίς τη βοήθεια πλαισίων άλλων όψεων. Σύμφωνα με τους παραπάνω καθορισμούς ιεραρχίας πλαισίων διαμορφώνονται τρία στρώματα. Αυτά τα τρία στρώματα του στερεοσκοπικού βίντεο μπορούν να χρησιμοποιηθούν για την άνιση προστασία λάθους (unequal error protection) (UEP).

Κεφάλαιο 4ο. Υλοποίηση - Αποτελέσματα

Στόχοι αυτής της πτυχιακής είναι η διερεύνηση της μείωσης της αντιληπτής ποιότητας των MVC ακολουθιών έπειτα από την μετάδοση τους σε κανάλια με απώλειες, η εύρεση του πλεονασμού ο οποίος προστίθεται κατά την κωδικοποίηση και την πακετοποίηση της ακολουθίας και τέλος η εύρεση του ποσοστού των πλαισίων που χάνονται κατά την μετάδοση τους σε κανάλια με απώλειες. Στην ενότητα 4.1 δίδεται ο τρόπος με τον οποίο υλοποιήθηκε το MVC σύστημα ενώ στην ενότητα 4.2 δίδονται όλα τα παραπάνω ερευνητικά αποτελέσματα.

4.1. Υλοποίηση

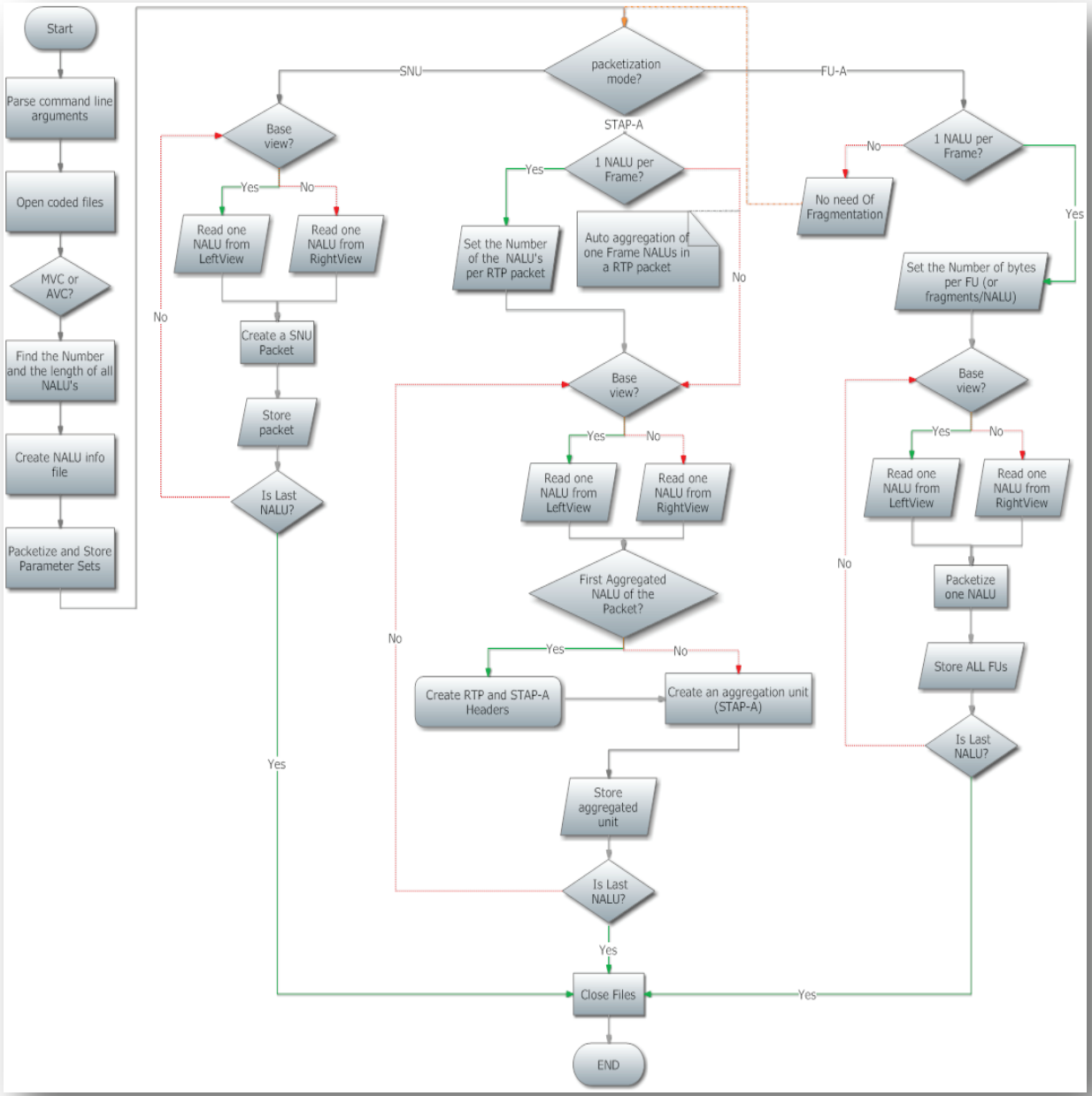
Στην Εικόνα 1 γίνεται αναπαράσταση της πλατφόρμας που δημιουργήθηκε για την διεξαγωγή των πειραμάτων της συγκεκριμένης πτυχιακής. Η μετάδοση MVC ακολουθιών μέσω IP επιτυγχάνεται με την χρήση αρκετών εργαλείων. Πρωταρχικά κατασκευάστηκε ο MVC RTP MST Packetizer και Depacketizer χρησιμοποιώντας ως γλώσσα προγραμματισμού την C, ενώ έπειτα δημιουργήθηκε ένας Multithreading MVC MST UDP/IP Server και Client στην java για την μετάδοση των IP πακέτων στο δίκτυο. Για την κωδικοποίηση των MVC ακολουθιών χρησιμοποιήθηκε ο κωδικοποιητής στερεοσκοπικών ακολουθιών της nokia ενώ για τον έλεγχο του δικτύου χρησιμοποιήθηκε το ανοιχτού κώδικα εργαλείο Dummynet.

4.1.1. H.264/MVC MST RTP Packetizer

Ο RTP Packetizer κατασκευάστηκε για να δημιουργεί RTP πακέτα μιας H.264/MVC κωδικοποιημένης ακολουθίας ανεξαρτήτου εάν τα NALUs περιέχουν ένα ολόκληρο πλαίσιο ή μέρος του. Είναι σε θέση να δημιουργήσει RTP πακέτα κάνοντας χρήση όλων των δομών ωφέλιμου φορτίου που ορίζονται από το πρότυπο (Single NAL Unit , Aggregation Packet(STAP-A) , Fragmentation Unit (FU-A)).

Η Εικόνα 16 αναπαριστά το “Flow Chart” διάγραμμα στο οποίο εμπεριέχονται τα βασικά βήματα που εκτελεί ο RTP Packetizer (μέθοδος main). Λόγω του μεγάλου όγκου της πληροφορίας, είναι δυνατή η μετέπειτα ανάλυση μόνο των τριών βασικών συναρτήσεων δημιουργίας RTP πακέτων .

Ανεξαρτήτου τρόπου κωδικοποίησης (1 NALU/Frame ή more NALUs/Frame) ο RTP Packetizer κατά την χρήση του SNU ενθυλακώνει κάθε NALU σε ένα RTP πακέτο. Κατά την χρήση του STAP-A, ο αριθμός των συσσωρευμένων NALUs ανά RTP ορίζεται μόνο εάν κάθε NALU περιέχει ένα ολόκληρο πλαίσιο ενώ εάν κάθε πλαίσιο ενθυλακώνεται σε περισσότερα του ενός NALU τότε αυτόματα όλα τα NALUs κάθε πλαισίου ενθυλακώνονται σε ένα RTP πακέτο. Τέλος, η χρήση του FU είναι δυνατή μόνο όταν κάθε NALU περιέχει ένα ολόκληρο πλαίσιο, εφόσον δεν υπάρχει λόγος περεταίρω κατακερματισμού των είδη μικρών NALUs(512 ,1024 , 1500 bytes). Στο FU υπάρχει η δυνατότητα ορισμού των αριθμών των τεμαχίων ανά NALU ενώ ο ποιο «έξυπνος» τρόπος είναι ο ορισμός του μεγέθους του κάθε FU (σε Bytes).

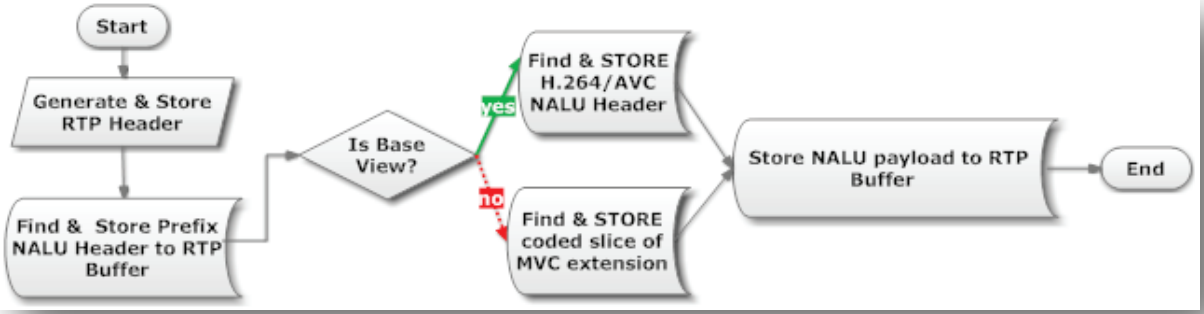


Εικόνα 16. Flow Chart διάγραμμα του H.264/MVC RTP MST Packetizer

3.1.1.1. Συνάρτηση δημιουργίας Single NAL Unit πακέτων

Η Εικόνα 17 αναπαριστά το "Flow Chart" διάγραμμα της συνάρτησης δημιουργίας SNU πακέτων. Η εν λόγω συνάρτηση καλείτε από το βασικό αλγόριθμο (main) που αναπαριστάται στην Εικόνα 16 ("*Create a SNU packet*"). Η ενθυλάκωση του κάθε NALU σε ένα RTP πακέτο γίνεται κάνοντας χρήση της SNU δομής, οπότε η παρακάτω συνάρτηση καλείτε για κάθε ένα NAL Unit και των δύο όψεων της κωδικοποιημένης ακολουθίας. Τα ίδια RTP πακέτα δημιουργούνται τόσο από τον RTP Packetizer (SNU δομή) τόσο και από τον κωδικοποιητή της nokia, οπότε ο RTP Depacketizer είναι συμβατός και με τα δύο εργαλεία.

Το μειονέκτημα χρήσης αυτής της δομής ωφέλιμου φορτίου είναι ότι στην περίπτωση που το κάθε πλαίσιο ενθυλακώνεται σε ένα NALU υπάρχει το πρόβλημα της απώλειας ενός ολόκληρου πλαισίου όταν χαθεί ο μοναδικός H.264/AVC NALU header που ενθυλακώνει και προσδιορίζει το συγκεκριμένο NALU.

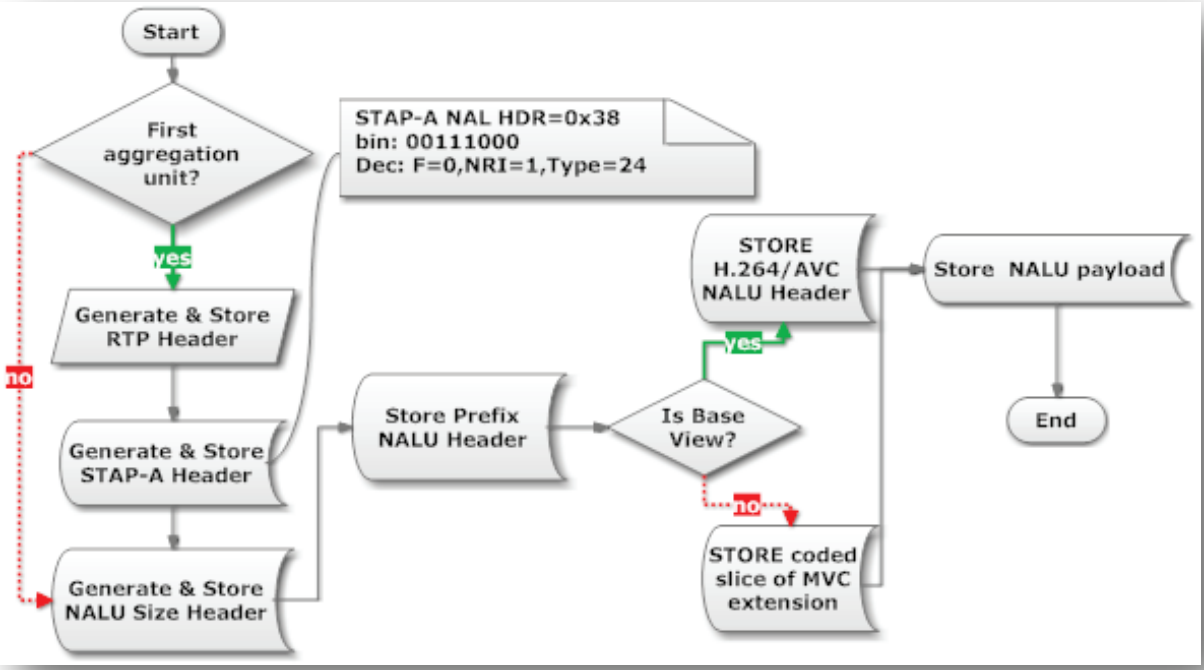


Εικόνα 17. Συνάρτηση του Single NAL Unit

4.1.1.2. Συνάρτηση δημιουργίας Aggregation πακέτων

Η Εικόνα 18 αναπαριστά το "Flow Chart" διάγραμμα της συνάρτησης δημιουργίας Aggregation πακέτων (STAP-A). Η εν λόγω συνάρτηση καλείτε από το βασικό αλγόριθμο (main) που αναπαριστάται στην εικόνα 16 ("*Create an Aggregation Unit*"). Για την ενθυλάκωση όλων των aggregation unit σε ένα RTP πακέτο γίνεται εισαγωγή του RTP και του STAP-A header μόνο στο πρώτο aggregation unit. Η ενθυλάκωση των NALUs σε aggregation units (STAP-A) γίνεται με την εισαγωγή του NALU Size header πριν από την έναρξη του ωφέλιμου φορτίου του εκάστοτε NALU.

Το πλεονέκτημα χρήσης αυτής της δομής ωφέλιμου φορτίου σε σχέση με τις υπόλοιπες είναι ότι εισάγει το μικρότερο "overhead". Το μειονέκτημα χρήσης αυτής της δομής ωφέλιμου φορτίου είναι ότι στην περίπτωση που κάθε πλαίσιο ενθυλακώνεται σε ένα NALU συνεχίζει να υπάρχει το πρόβλημα της απώλειας ενός ολόκληρου πλαισίου έπειτα από την απώλεια του H.264/AVC NALU header που ενθυλακώνει το συγκεκριμένο NALU.



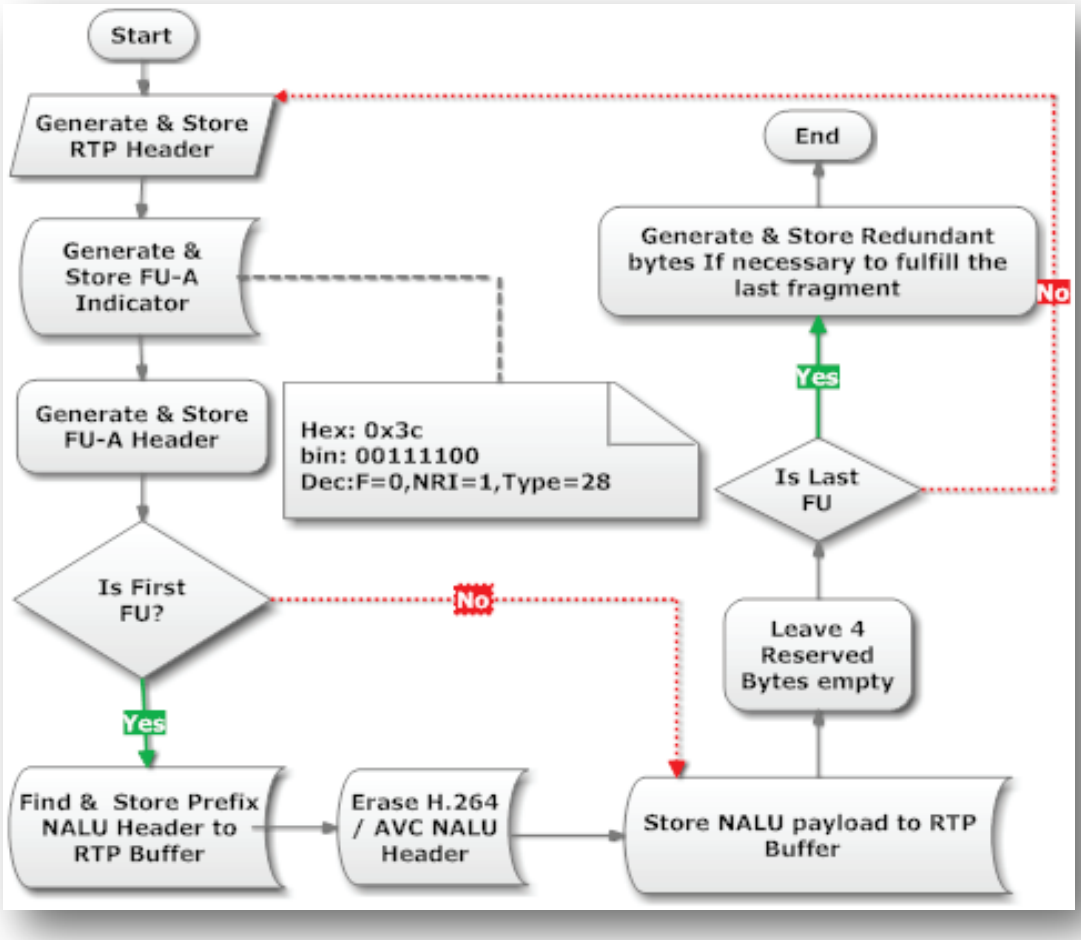
Εικόνα 18. Συνάρτηση του Aggregation Packet (STAP-A)

4.1.1.3. Συνάρτηση δημιουργίας Fragmentation Unit πακέτων

Η Εικόνα 19 αναπαριστά το "Flow Chart" διάγραμμα της συνάρτησης δημιουργίας Fragmentation Units (FU-A) . Η εν λόγω συνάρτηση καλείτε από το βασικό αλγόριθμο (main) που αναπαριστάται στην εικόνα 16 ("*Packetize one NALU*"). Για την δημιουργία RTP πακέτων (FU-A), ένας βρόγχος επαναλαμβάνεται τόσες φορές όσα είναι τα τεμάχια του κάθε NALU.

Κάθε NALU μπορεί είτε να κατακερματίζεται σε έναν συγκεκριμένο αριθμό τεμαχίων ανά πλαίσιο (κάθε RTP πακέτο περιέχει έναν τυχαίο αριθμό από Bytes) είτε να ενθυλακώνει ένα συγκεκριμένο αριθμό από bytes ανά τεμάχιο (τυχαίος αριθμό τεμαχίων). Κατά την δεύτερη διαδικασία, το μέγεθος του ωφέλιμου φορτίου που ενθυλακώνεται σε κάθε RTP πακέτο ελέγχεται έτσι ώστε να μην ξεπερνάει το MTU size. Επιπρόσθετα, για κάθε RTP πακέτο εισάγονται 4 bytes πλεονασμού για μελλοντική χρήση ενώ στο τελευταίο τεμάχιο του κάθε NALU εισάγονται τόσα byte πλεονασμού όσα είναι αναγκαία για την συμπλήρωση του τελευταίου FU.

Το πλεονέκτημα χρήσης αυτής της δομής ωφέλιμου φορτίου σε σχέση με τις υπόλοιπες είναι ότι δίνει τη δυνατότητα διάσωσης των χαμένων H.264/AVC NALU Header οπότε και των αντίστοιχων πλαισίων (3.1.4). Αντίθετα, το μειονέκτημα χρήσης αυτής της δομής ωφέλιμου φορτίου είναι ότι εισάγει αρκετά μεγάλο πλεονασμό.



Εικόνα 19. Συνάρτηση του Fragmentation Unit(FU-A)

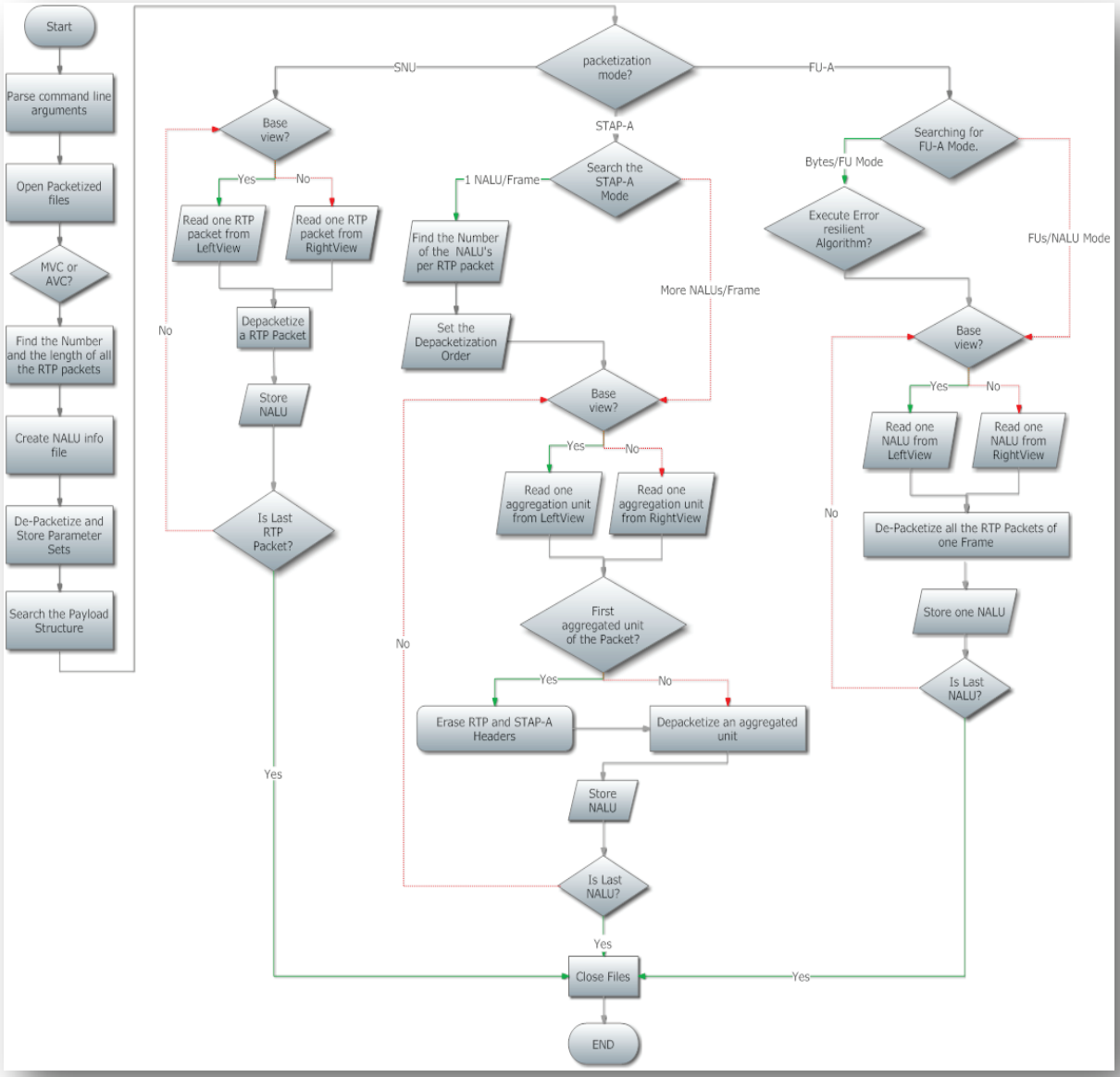
4.1.2. H.264/MVC MST RTP De-Packetizer

Ο RTP De-Packetizer κατασκευάστηκε για να ξεπακετάρει τα RTP πακέτα (SNU , STAP-A , FU-A) τα οποία λαμβάνει το εκάστοτε τερματικό και περιέχουν το ωφέλιμο φορτίο μιας H.264/MVC κωδικοποιημένης ακολουθίας. Σε γενικές γραμμές εκτελεί την αντίστροφη διαδικασία από αυτήν του RTP Packetizer, όμως υπάρχουν όμως κάποιες επιπλέον «δυσκολίες» στις οποίες θα πρέπει να είναι σε θέση να ανταπεξέλθει:

- Έλεγχος της δομής του ωφέλιμου φορτίου που έχει χρησιμοποιηθεί κατά την πακετοποίηση (“*Search the payload structure ”*). Αυτήν η διαδικασία γίνεται ελέγχοντας την τιμή του type (5 τελευταία bit) του πρώτου byte έπειτα από τον RTP Header. Εάν έχει γίνει χρήση του SNU τότε ένα Prefix NalU Header θα πρέπει να υπάρχει σε αυτήν την θέση έχοντας σαν τιμή στο πεδίο type τον αριθμό 14 (DEC). Εάν έχει γίνει χρήση του STAP-A τότε θα πρέπει να υπάρχει ένας STAP-A Header με τιμή 24 (DEC) , ενώ τέλος εάν έχει γίνει χρήση του FU-A τότε ένας FU-A Header θα πρέπει να υπάρχει με την τιμή 28(DEC).
- Ανεκτικότητα στις απώλειες πακέτων.
- Εύρεση του αριθμού των RTP πακέτων και του μεγέθους του κάθε RTP πακέτου (“*Find the number and the length of all the RTP packets*”).
- Για την διαδικασία του FU-A Depacketization:
    - Υπάρχουν δύο διαθέσιμες συναρτήσεις ξεπακεταρίσματος. Η συνάρτηση που θα εκτελεστεί επιλέγεται σύμφωνα με τον τρόπο με τον οποίο έχει γίνει η πακετοποίηση. Ο Πρώτος τρόπος διαθέτει σταθερό αριθμό τεμαχίων ανά πλαίσιο (FUs/NALUs) και μεταβλητό μήκος RTP πακέτων ενώ ο δεύτερος τρόπος διαθέτει σταθερό μέγεθος RTP πακέτων (Bytes/FU).
    - Πρέπει να γίνεται αφαίρεση όλης της άχρηστης πληροφορίας(redundant Bytes).
    - Ο H.264/AVC NALU Header θα πρέπει να ανακατασκευάζεται μέσω των FU indicator και FU Header. Εάν ο ένας ή και οι δύο Headers έχουν χαθεί κατά την μετάδοση τότε ο Depacketizer είναι σε θέση να ανακατασκευάσει τον H.264/AVC NALU Header χρησιμοποιώντας τους Headers των επόμενων FUs του ίδιου NAL Unit. Αυτό έχει σαν αποτέλεσμα ο αποκωδικοποιητής να είναι σε θέση να αναγνωρίζει και να αποκωδικοποιεί **<u>όλα</u>** τα πλαίσια.

➢ Κατά την διαδικασία ξεπακεταρίσματος RTP πακέτων που κάνουν χρήση της SNU και STAP-A δομής δεν είναι σε θέση ο RTP Depacketizer να ανακατασκευάσει τους χαμένους H.264/AVC NALU Headers, οπότε είναι αδύνατη η διάσωση των χαμένων πλαισίων. Αντί αυτού, είναι δυνατή η ανεύρεση του αριθμού και της θέσης των χαμένων πλαισίων έτσι ώστε να είναι δυνατή ή χρήση του μηχανισμού αντιγραφής πλαισίων.

Στην εικόνα 20 γίνεται αναπαράσταση του “Flow Chart” διαγράμματος το οποίο περιέχει τα βασικά βήματα που εκτελεί ο RTP De-Packetizer (μέθοδος main).



Εικόνα 20. MVC MST RTP Depacketizer

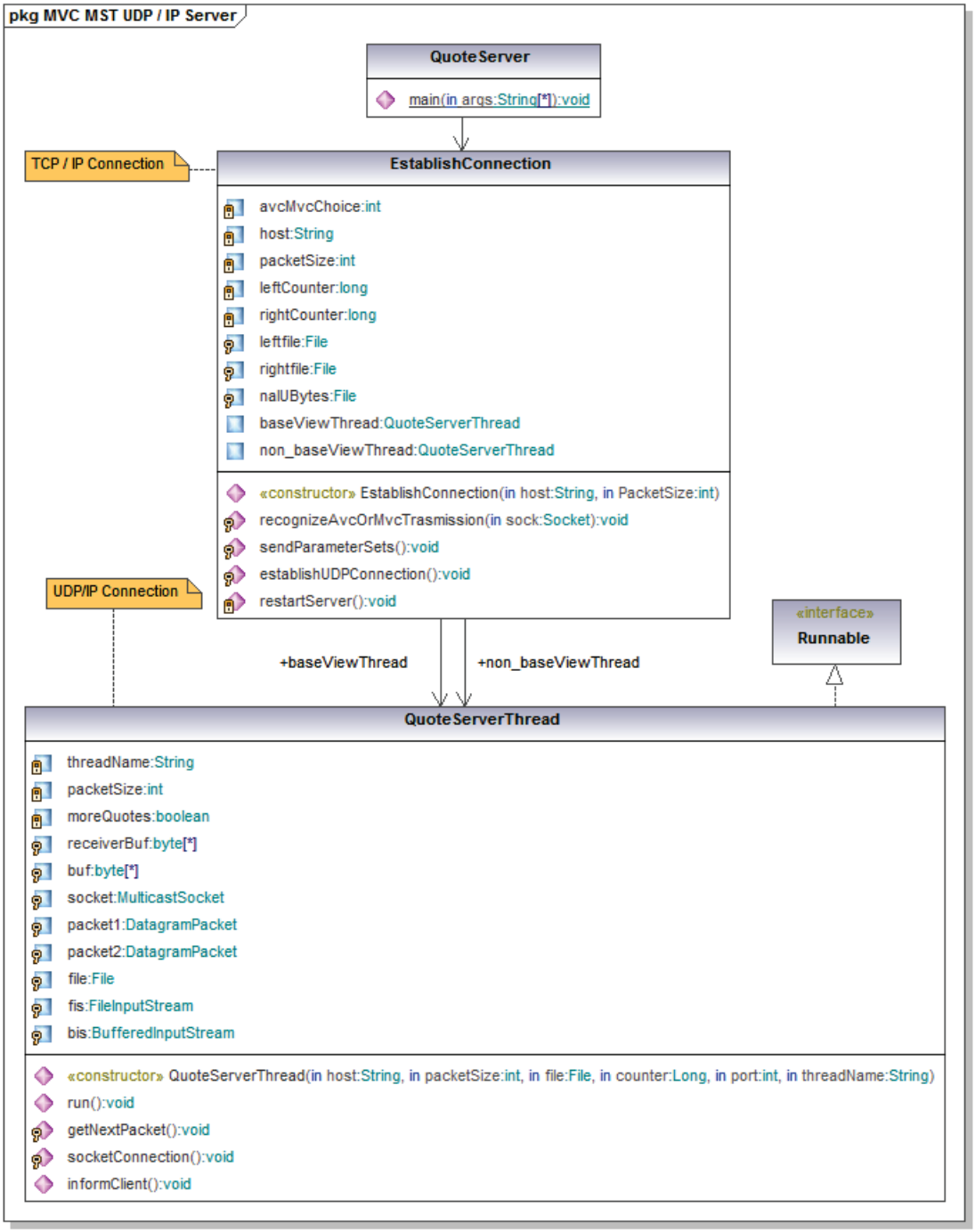
4.1.3. MVC MST UDP/ IP Streamer

Σε αυτήν την ενότητα αναλύεται το εργαλείο το οποίο κατασκευάστηκε για την μετάδοση των MVC ακολουθιών. Όπως απεικονίζεται στην Εικόνα 21, ο MVC MST UDP/IP Streamer αποτελείται από τρείς κλάσεις:

1. Η κλάση ”*QuoteServer*” περιέχει την μέθοδο “*main*” η οποία δέχεται από τον χρήστη την IP και το μέγεθος του πακέτου .
2. Η κλάση ”*EstablishConnection*” μέσω της μεθόδου “*sendParameterSets()*” (καλείτε από την “*main*”) δημιουργεί ένα κανάλι σηματοδοσίας (TCP / IP) για ασφαλή μετάδοση του SPS και PPS όταν κάποιο τερματικό ζητήσει την λήψη κάποιας 3D ή 2D ακολουθίας μέσω της “*recognizeAvcOrMvcTrasmission()*”. ***Αυτά τα δεδομένα σύμφωνα με το πρότυπο πρέπει να στέλνονται πριν την MVC ακολουθία κάνοντας χρήση ενός αξιόπιστου καναλιού.*** Έπειτα από την αποστολή του SPS και του PPS, μέσω της μεθόδου “execute()” της κλάσης “*ExecutorService*”,

καλείτε η μέθοδος "*establishUDPConnection()*" δύο φορές (MVC) ή μία (AVC) έτσι ώστε να εκτελεστούν τόσα νήματα όσες είναι και οι προς αποστολή ακολουθίες.

3. Η κλάση "*QuoteServerThread*" λειτουργεί κάνοντας χρήση νημάτων εφόσον υλοποίει την διεπαφή "*Runnable*". Ο κατασκευαστής της, εκτός των άλλων, δέχεται τον αριθμό της πόρτας αφού το κάθε νήμα θα δημιουργεί το δικό του κανάλι για την αποστολή της κάθε ακολουθίας (MST). Έπειτα, με την χρήση της μεθόδου 'SocketConnection()', αποστέλλονται όλα τα πακέτα ("*getNextPacket()"*) μέσω UDP/IP .
4. Όταν όλες οι ακολουθίες μεταδοθούν ή η σύνδεση με τον χρήστη έχει χαθεί τότε ο Server ενημερώνει τον χρήστη στέλνοντας ένα μήνυμα «ειδικού σκοπού» μέσω της "*informClient()*" και κάνει επανεκκίνηση μέσω της "*restartServer()*".



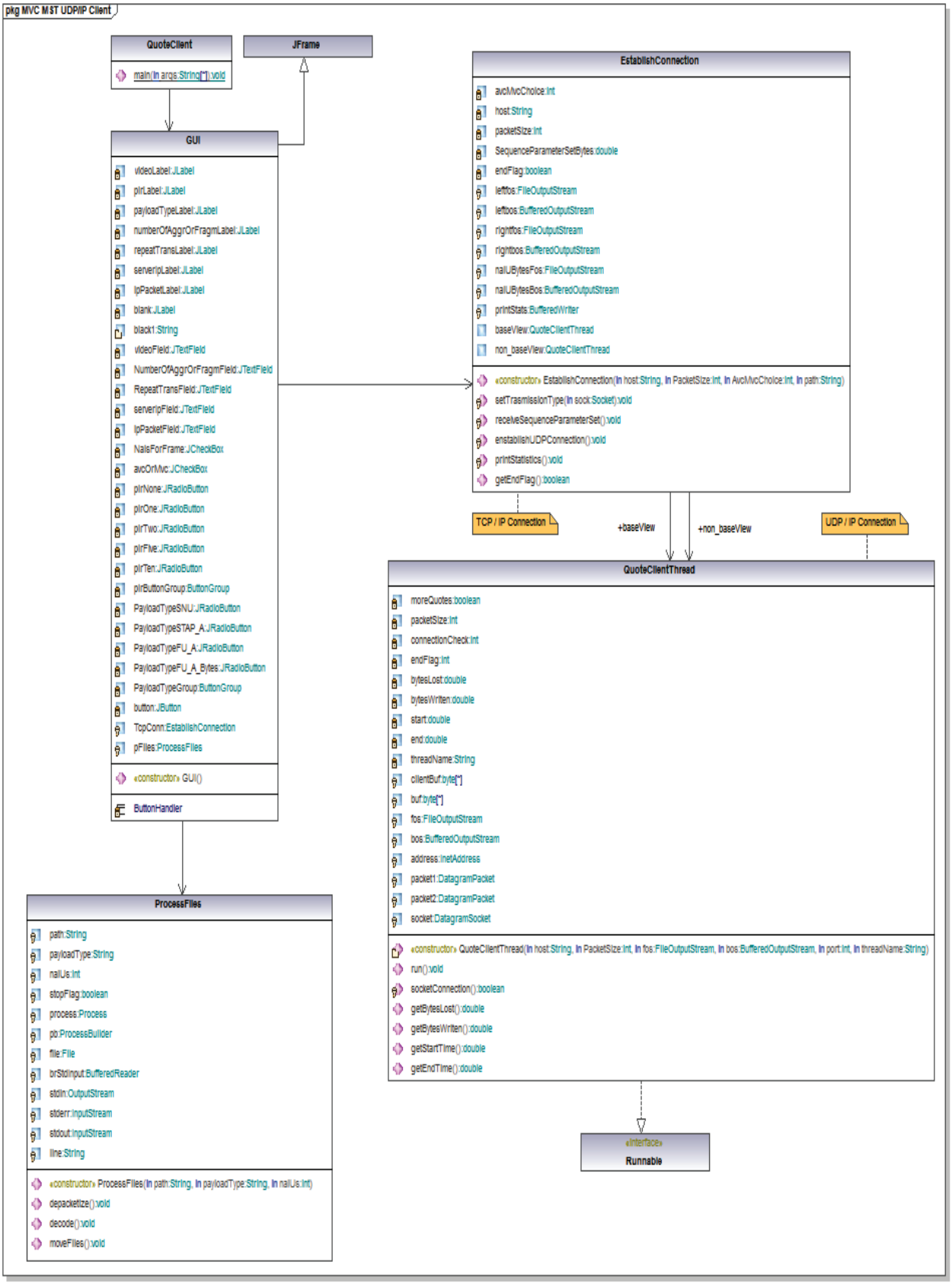
Εικόνα 21. MVC MST UDP/IP Streamer

4.1.4. MVC MST UDP/ IP Client

Σε αυτήν την ενότητα αναλύεται το εργαλείο το οποίο κατασκευάστηκε για την λήψη των MVC ακολουθιών. Στην Εικόνα 22 απεικονίζεται ο MVC MST UDP/IP Client ο οποίος σε γενικές γραμμές κάνει την αντίστροφη διαδικασία από αυτήν του Streamer.

Η βασική διαφοροποίηση του είναι η χρήση μιας επιπλέον κλάσης με το όνομα "*GUI*" η οποία εκτελείται από την 'main' όταν εκτελεστεί το πρόγραμμα. Αυτή η κλάση έχει ως σκοπό την εμφάνιση κατά την εκτέλεση ενός γραφικού παραθύρου για την επιλογή των παραμέτρων μετάδοσης (MVC/AVC , Payload Type , Number of NALU's per Frame , IP packet Size , Server IP,Number of Re-Transmissions) ενώ επίσης παρέχει τον συγχρονισμό της λήψης και αποκωδικοποίησης των πακέτων. Όταν το κουμπί της γραφικής εφαρμογής πατηθεί, τότε η εσωτερική κλάση "*ButtonHandler*" καλείται για να επεξεργαστεί τα δεδομένα του χρήστη και να εκτελέσει την κλάση "*EnstablishConnection*" τόσες φορές όσες του όρισε ο χρήστης μέσω του πεδίου 'Number of Re-Transmissions'. Οι ακολουθίες ξεπακετάρονται και αποκωδικοποιούνται μέσω των μεθόδων " *depacketize()*" και " *decode()*" της κλάσης "*ProcessFiles*" οι οποίες καλούν τα εργαλεία του RTP Depacketizer και του nokia MVC Decoder. Πρέπει να τονιστεί ότι το συγκεκριμένο γραφικό περιβάλλον δεν κατασκευάστηκε για την χρήση του από πραγματικούς χρήστες αλλά για την γρηγορότερη εξαγωγή των πειραματικών αποτελεσμάτων της συγκεκριμένης πτυχιακής.

Η μέθοδος "*PrintStatistic*s()" δίνει την δυνατότητα της προβολής και της αποθήκευσης των στατιστικών στοιχείων της εκάστοτε μετάδοσης. Τα στατιστικά στοιχεία περιέχουν τον χρόνο μετάδοσης των ακολουθιών "getStartTime() - getEndTime()", τον αριθμό των bytes που λήφθηκαν "getBytesWriten()" , τον αριθμό των bytes που χάθηκαν "getBytesLost()" καθώς και τον ρυθμό απώλειας πακέτων που είχε το κανάλι.



Εικόνα 22. MVC MST UDP/IP Client

4.2. Αποτελέσματα

Η πρώτη ενότητα αυτού του κεφαλαίου υπολογίζει και αναλύει τον πλεονασμό τον οποίο εισάγεται κατά την κωδικοποίηση και την πακετοποίηση των MVC ακολουθιών. Οι επόμενες δύο ενότητες αναλύουν τις επιπτώσεις της μετάδοσης MVC ακολουθιών σε κανάλια με απώλειες. Οι συνέπειες αυτές είναι είτε η απώλεια ολόκληρων πλαισίων είτε η μείωση της ποιότητας του 3D βίντεο (PSNR).

Τα χαρακτηριστικά των ακολουθιών βίντεο [20] και των παραμέτρων κωδικοποίησης που χρησιμοποιήθηκαν κατά την διάρκεια των πειραμάτων δίδονται στον πίνακα xv, οι παράμετροι RTP πακετοποίησης περιέχονται στον πίνακα xvi, ενώ στον πίνακα xvii απεικονίζονται οι παράμετροι δικτύωσης. Κάθε μετάδοση επαναλήφθηκε 15 φορές για την απόκτηση του μέσου όρου του PSNR και των χαμένων πλαισίων. Όταν γίνεται χρήση του 1 NALU ανά πλαίσιο τότε δεν γίνεται χρήση της STAP-A δομής αφού αυτό θα προκαλούσε την ενθυλάκωση πολλαπλών πλαισίων σε ένα RTP πακέτο. Επιπρόσθετα, η FU-A δομή δεν εφαρμόζεται στην περίπτωση των πολλαπλών NALUs ανά πλαίσιο αφού ο περαιτέρω κατακερματισμός των NALUs σε RTP πακέτα είναι περιττός.

Πίνακας xv. Παράμετροι Κωδικοποίησης

Ακολουθία βίντεο	Flamenco	Objects
Αριθμός πλαισίων	1000	624
Intra περίοδος	5 πλαίσια	5 πλαίσια
Ρυθμός πλαισίων	25 fps	25 fps
Ανάλυση	640x480 pixels	640x480 pixels
Παράμετρος κβαντίσου	24	24
Αριθμός όψεων	2	2

Πίνακας xvi. Παράμετροι πακετοποίησης

Επιλογές πακετοποίησης του βίντεο	1 NALU / πλαίσιο		Πολλαπλά NALUs/πλαίσιο	
Επιλογές RTP πακετοποίησης	SNU	FU-A	SNU	STAP-A

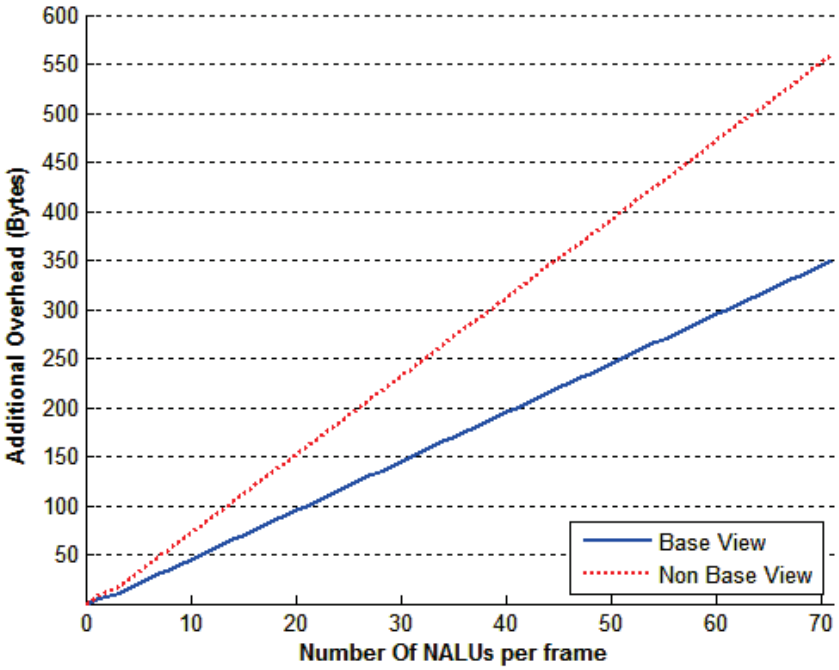
Πίνακας xvii. Παράμετροι δικτύωσης

MTU μέγεθος (Bytes)	1024	512
Ποσοστό απώλειας πακέτων	0% , 1% , 2% , 5%	
Μοντέλος σφάλματος	Όλες οι όψεις	Βασική Όψη

4.2.1. Επιπρόσθετος πλεονασμός

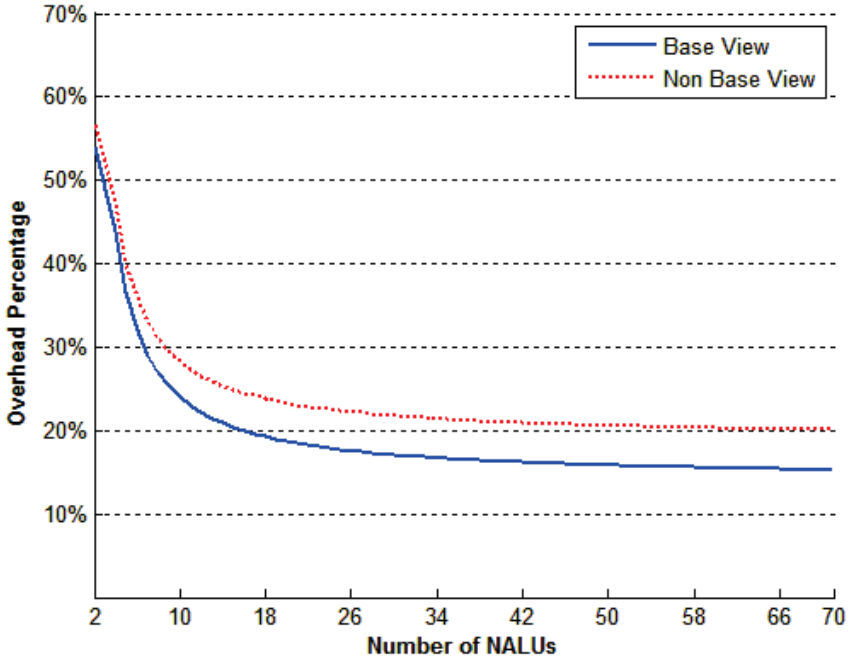
Σε αυτήν την ενότητα αρχικά αναλύεται ο πλεονασμός ο οποίος εισάγεται κατά την κωδικοποίηση και την ενθυλάκωση του ωφέλιμου φορτίου σε NALUs. Έπειτα συγκρίνονται οι τρείς δομές ωφέλιμου φορτίου βάση του πλεονασμού του οποίου εισάγουν κατά την ενθυλάκωση των NALUs σε RTP πακέτα.

Όταν το κάθε πλαίσιο ενθυλακώνεται σε περισσότερα του ενός NAL Units τότε ένας επιπρόσθετος πλεονασμός προστίθεται στην κωδικοποιημένη ροή από bit. Στην Εικόνα 23 απεικονίζεται ο αυξημένος πλεονασμός που εισάγεται όταν ο αριθμός των NALUs ανά πλαίσιο αυξάνεται.



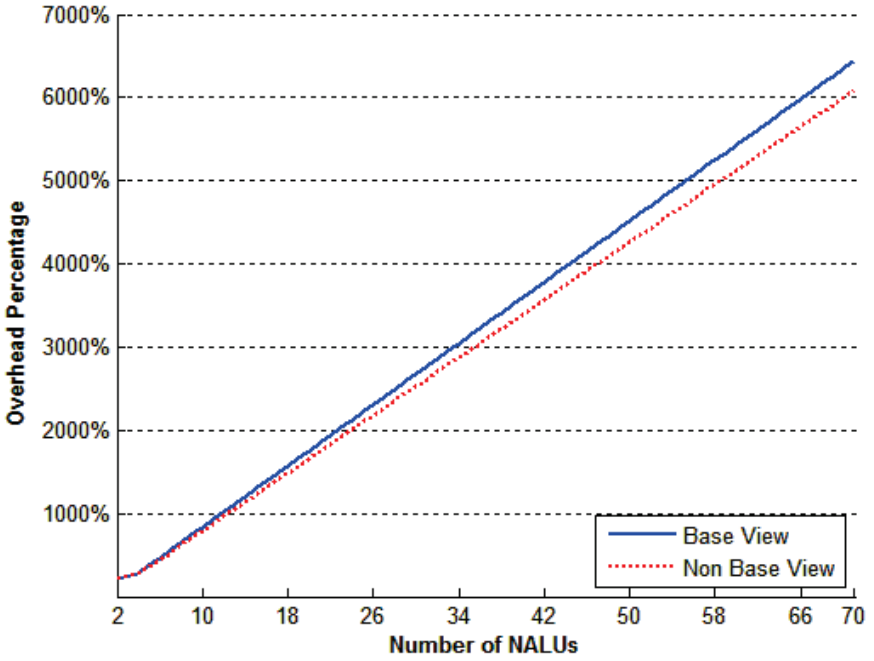
Εικόνα 23. Αύξηση του πλεονασμού λόγω της ενθυλάκωσης ενός πλαισίου σε πολλαπλά NALUs

Η Εικόνα 24 απεικονίζει την μείωση του πλεονασμού (σε σύγκριση με την SNU δομή) καθώς αυξάνεται ο αριθμός των συσσωρευμένων NALUs ανά RTP πακέτο. Το κάθε aggregation πακέτο μπορεί να ενθυλακώσει από 2 έως 70 aggregation units (NALUs) κάνοντας χρήση της STAP-A δομής.



Εικόνα 24. Σύγκριση του πλεονασμού του STAP-A σε σχέση με τον πλεονασμό του SNU.

Η Εικόνα 25 αποδεικνύει ότι όσο αυξάνεται ο αριθμός των τεμαχίων ανά NALU τόσο αυξάνεται ο συνολικός πλεονασμός. Το κάθε NALU μπορεί να κατακερματιστεί σε 2 έως 70 fragmentation units (FU-A).



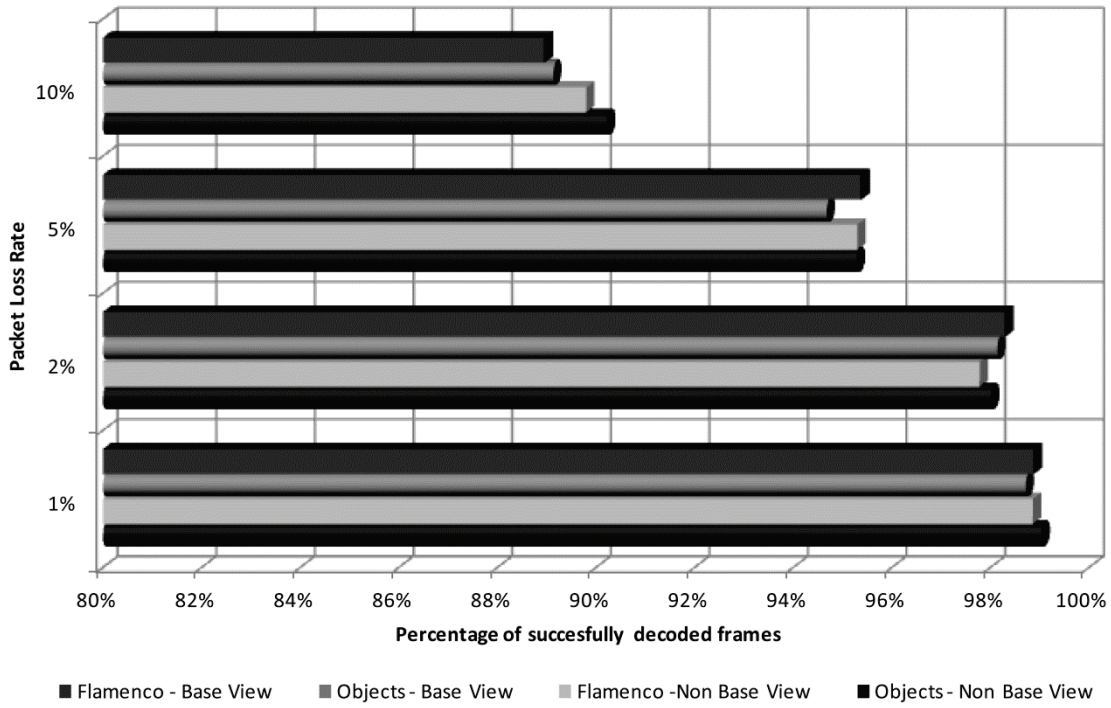
Εικόνα 25. Σύγκριση του πλεονασμού του FU-A σε σχέση με τον πλεονασμό του SNU.

Μελετώντας τις τρείς εικόνες, αποδεικνύεται ότι ο τεμαχισμός ενός πλαισίου σε πολλαπλά NALUs με την χρήση της SNU ή της STAP-A δομής έχει ως αποτέλεσμα την μεγάλη μείωση του πλεονασμού συγκριτικά με τον τεμαχισμό του στο RTP επίπεδο.

4.2.2. Χαμένα πλαίσια

Κατά την διάρκεια της μετάδοσης βίντεο σε δίκτυα με απώλειες μερικά από τα πλαίσια χάνονται πλήρως. Η απώλεια ενός ολόκληρου πλαισίου προκαλείται κυρίως λόγο της απώλειας ενός IP πακέτου το οποίο περιέχει το NAL Unit header αυτού του NALU (όταν το κάθε NALU περιέχει ένα ολόκληρο πλαίσιο). Εάν ο αποκωδικοποιητής δεν είναι σε θέση να αναγνωρίσει τα δεδομένα που περιέχονται σε ένα NAL Unit τότε αυτά απορρίπτονται.

Στην Εικόνα 26 απεικονίζεται ο αριθμός των πλαισίων ο οποίος κατάφερε να αποκωδικοποιηθεί κάτω από συγκεκριμένες συνθήκες ενός δικτύου. Το μέγεθος του κάθε IP πακέτου για το συγκεκριμένο πείραμα είναι 1024 bytes ενώ η δομή ωφέλιμου φορτίου που χρησιμοποιείται είναι η Single NAL Units. Στην οριζόντια στήλη αναγράφεται το ποσοστό των πλαισίων τα οποία αποκωδικοποιήθηκαν ενώ στην κάθετη ο ρυθμός απώλειας πακέτων.



Εικόνα 26. Ποσοστό αποκωδικοποιημένων πλαισίων της βασικής και της μη βασικής όψης κάνοντας χρήση της SNU δομής σε διαφορετικούς ρυθμούς απώλειας πακέτων ("Flamenco","Objects").

4.2.3. PSNR αποτελέσματα

Το **P**eak **S**ignal-To-**N**oise **R**atio υπολογίζει την αναλογία μεταξύ της μέγιστης δυνατής ισχύς ενός σήματος σε σχέση με την ισχύ του θορύβου του (λογαριθμική κλίμακα). Το σήμα είναι τα αρχικά δεδομένα ενώ ο θόρυβος είναι το σφάλμα το οποίο εισάγεται στην ακολουθία κατά την κωδικοποίηση και μετάδοση της. Οι τυπικές τιμές ενός απωλέστηκα κωδικοποιημένου πλαισίου είναι από 30 έως 50 dB ενώ οι αποδεκτές τιμές έπεται από ασύρματη μετάδοση του είναι από 20 έως 25 dB. Όσο υψηλότερη είναι η τιμή του PSNR τόσο χαμηλότερη είναι η τιμή του MSE και τόσο καλύτερη είναι η ποιότητα του βίντεο. Το PSNR υπολογίζεται μέσω του MSE χρησιμοποιώντας τους παρακάτω δύο τύπους.

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Εικόνα 27. Υπολογισμός του MSE και του PSNR

4.2.3.1. Απώλεια Πακέτων και στις δύο όψεις

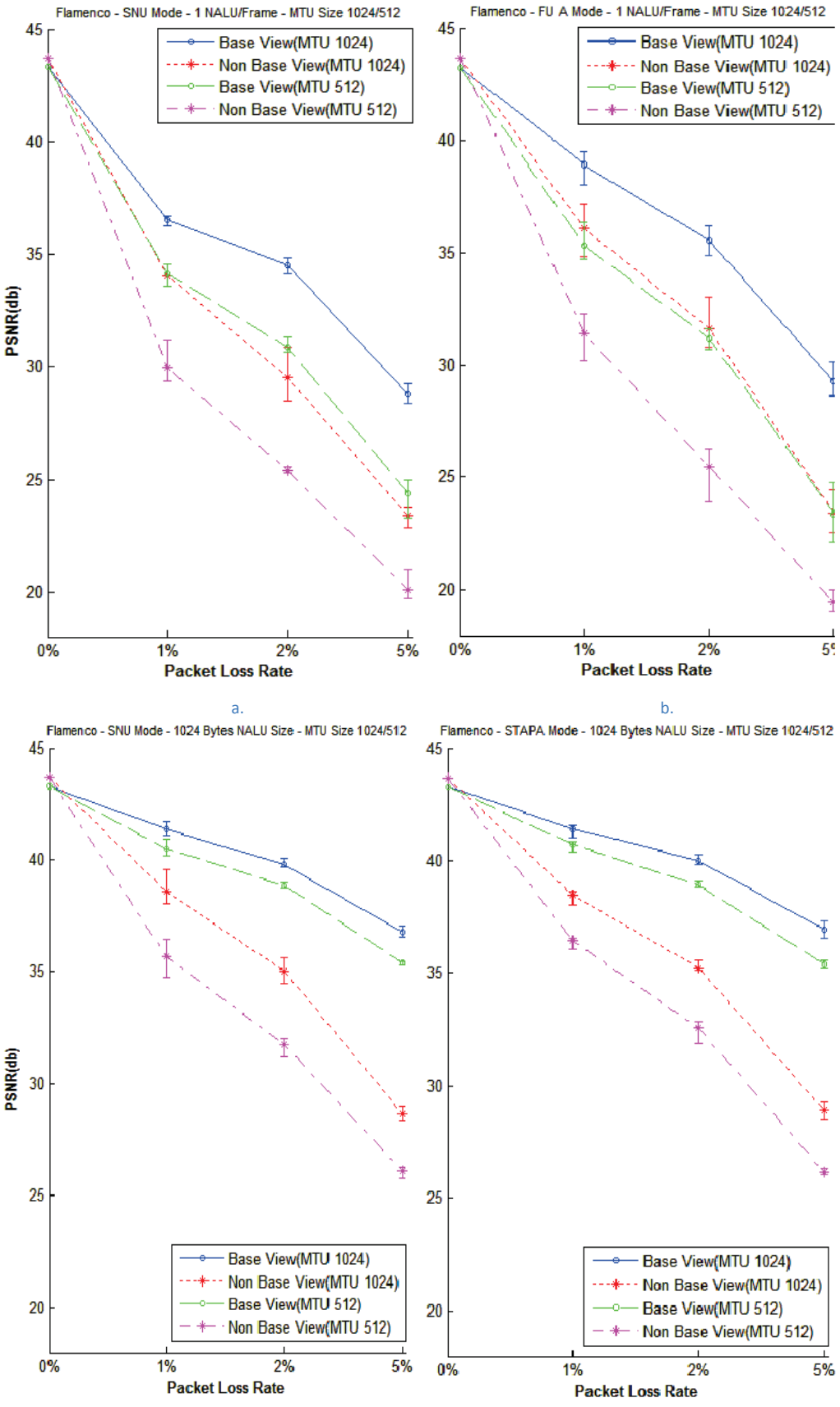
Οι Εικόνες 28 και 29 παρουσιάζουν το αντιληπτό PSNR δύο ακολουθιών πολλαπλών όψεων οι οποίες μεταδόθηκαν υπό διαφορετικές συνθήκες πακετοποίησης (a,b,c,d) και υπό διαφορετικές συνθήκες δικτύου. Είναι εμφανές ότι το αντιληπτό PSNR της βασικής όψης σε όλες τις περιπτώσεις είναι σημαντικά υψηλότερο από το αντίστοιχο της μη βασικής όψης. Αυτό συμβαίνει λόγω της επιπρόσθετης διάδοσης λάθους πού υπάρχει στην μη-βασική όψη λόγω της εξάρτησης της από την βασική όψη. Αυτή η διαφορά του PSNR θα μπορούσε να αποφευχθεί μόνο εάν η μη βασική όψη είχε κωδικοποιηθεί ανεξάρτητα από την βασική όψη, κάτι το οποίο όμως θα είχε επιπτώσεις στην απόδοση της κωδικοποίησης (μεγαλύτερος όγκος δεδομένων). Επιπλέον, όσο το μέγεθος του MTU αυξάνεται τόσο το PSNR αυξάνεται λόγω της εξάλειψης των “error bursts”.

Ειδικότερα, οι Εικόνες 28a και 29a απεικονίζουν τα PSNR αποτελέσματα όταν η κωδικοποιημένη ακολουθία πακετάρεται με την χρήση της SNU δομής, οπότε κάθε RTP πακέτο περιέχει ένα NALU το οποίο περιέχει ένα πλαίσιο. Είναι εμφανές ότι η ενθυλάκωση ενός πλαισίου ανά RTP πακέτο παράγει το χαμηλότερο βίντεο QoS σε σύγκριση με όλους τους άλλους τύπους πακετοποίησης. Σε αυτήν την περίπτωση, η απώλεια ενός IP πακέτου που περιέχει το H.264/AVC NALU header έχει ως αποτέλεσμα την απώλεια του πλαισίου. Οι απώλειες ολόκληρων πλαισίων έχουν ως αποτέλεσμα την αύξηση της συνολικής αλλοίωσης του βίντεο. Για να είναι δυνατός ο υπολογισμός του PSNR, ο αποκωδικοποιητής υλοποιεί την τεχνική της αντιγραφής πλαισίου στην περίπτωση απώλειας ενός πλαισίου.

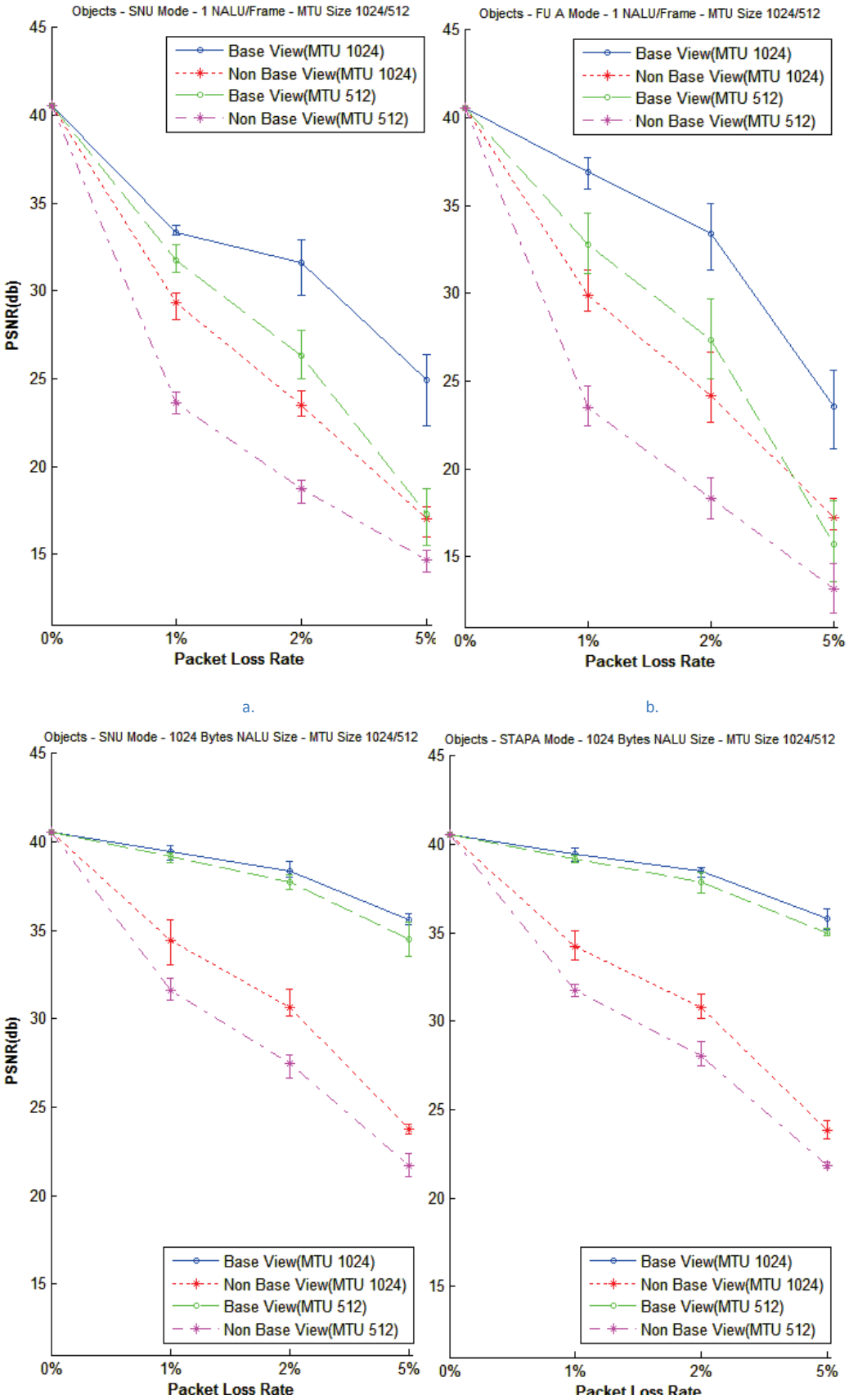
Στις Εικόνες 28b και 29b εμφανίζεται το PSNR της FU-A δομής. Ο τεμαχισμός ενός NALU με την παράλληλη χρήση του προτεινόμενου μηχανισμού διαχείρισης λαθών σε RTP επίπεδο (3.1.4) έχει ως αποτέλεσμα την αύξηση του PSNR, ειδικά στις περιπτώσεις του 1% και 2% plr. Σε αυτήν την περίπτωση όλα τα πλαίσια αποκωδικοποιούνται επιτυχώς, έχοντας ως αποτέλεσμα την σημαντική αύξηση του αντιληπτού PSNR.

Ωστόσο, τα πειράματα απέδειξαν ότι η ενθυλάκωση του κάθε πλαισίου σε πολλαπλά NALUs είχε ως αποτέλεσμα την δραματική αύξηση του video QoS συγκριτικά με τις δύο προηγούμενες περιπτώσεις. Σε αυτήν περίπτωση, το PSNR μετράται για τις περιπτώσεις χρήσης SNU(1 NALU ανά RTP πακέτο) και STAP-A(όλα τα NALUs ενός πλαισίου ανά RTP πακέτο) και απεικονίζεται στις εικόνες 28c,28d,29c και 29d αντίστοιχα. Ο τεμαχισμός ενός πλαισίου σε πολλαπλά NALUS μειώνει την παραμόρφωση στο βίντεο ενώ όπως αποδεικνύεται μειώνει τις επιπτώσεις της αλλαγής του μέγεθος MTU στο PSNR.

Τελικά, συγκρίνοντας τα PSNR αποτελέσματα των ακολουθιών “flamenco” και “objects” είναι εμφανές ότι η “flamenco” επιτυγχάνει υψηλότερο PSNR. Αυτό συμβαίνει λόγω του διαφορετικού περιεχομένου του βίντεο (λιγότερη κίνηση) που περιέχεται στην ακολουθία “flamenco”.



Εικόνα 28. PSNR αποτελέσματα: διαφορετικό ποσοστό απώλειας πακέτων - διαφορετικοί τύποι πακετοποίησης - διαφορετικά MTU μεγέθη – απώλεια σε δύο όψεις - "flamenco"

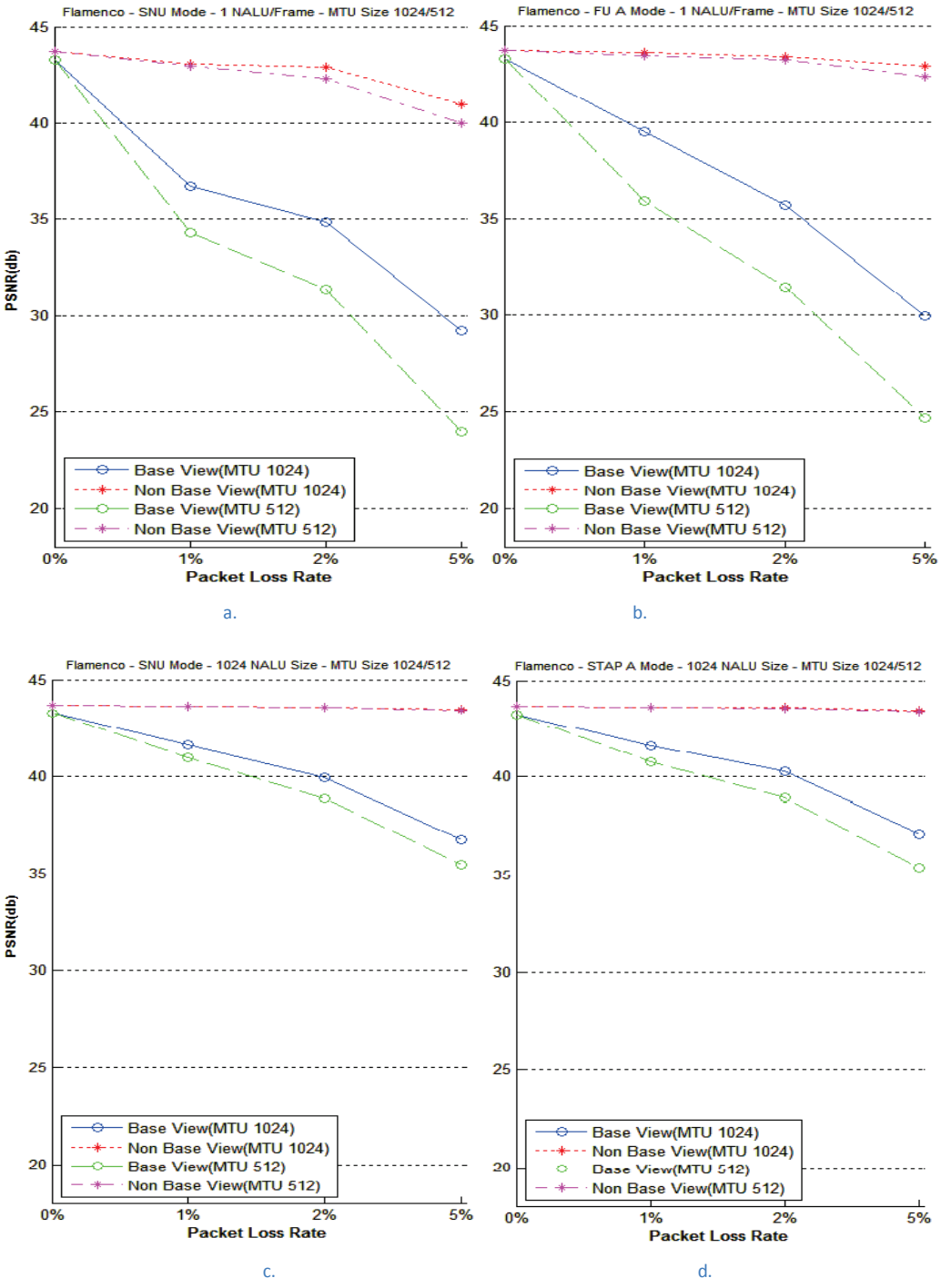


Εικόνα 29. PSNR αποτελέσματα: διαφορετικό ποσοστό απώλειας πακέτων - διαφορετικοί τύποι πακετοποίησης - διαφορετικά MTU μεγέθη - απώλεια σε δύο όψεις - "Objects"

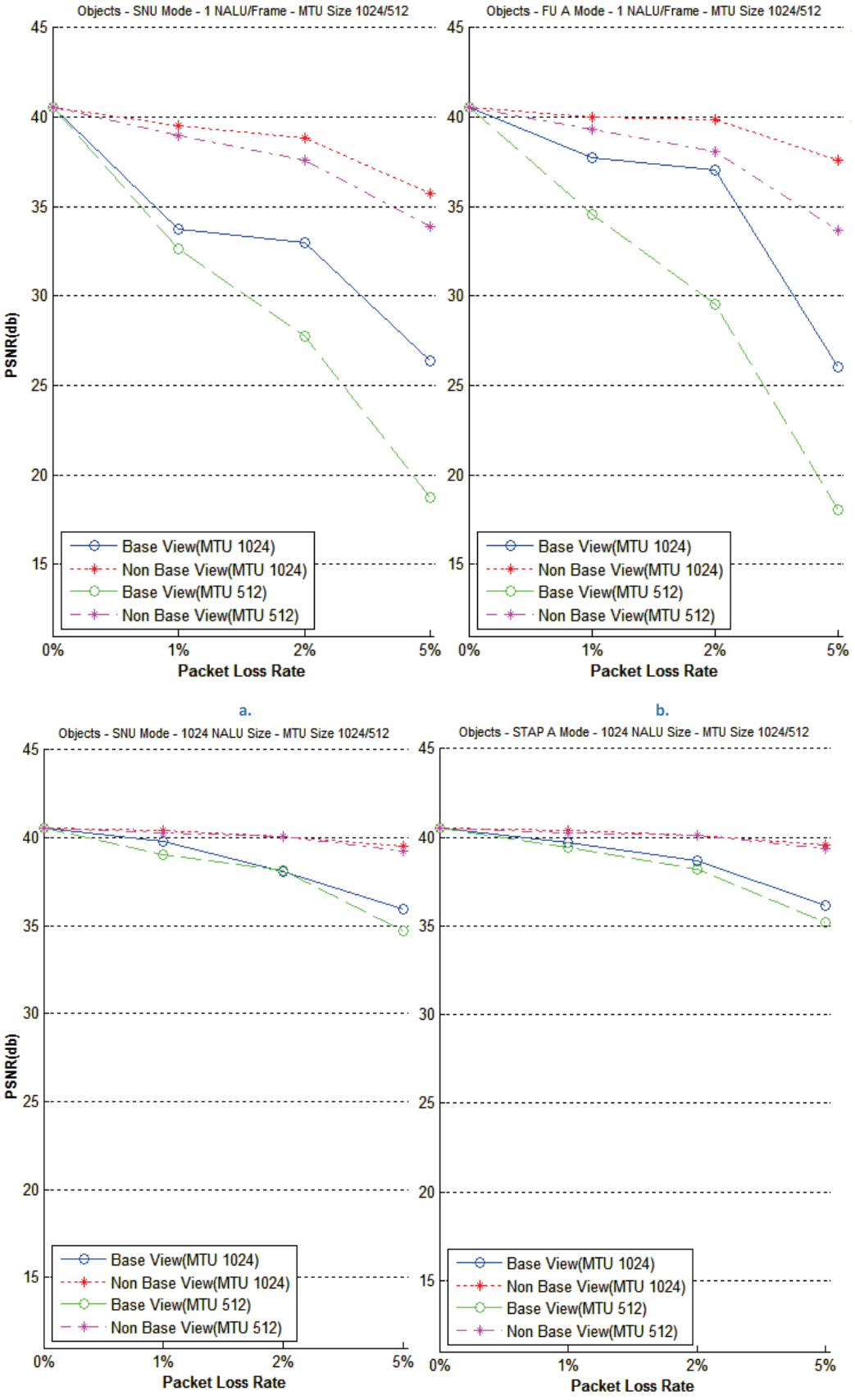
4.2.3.2. Απώλεια Πακέτων μόνο στην βασική όψη – Διάδοση λάθους

Στην προηγούμενη ενότητα αναφέρεται ότι η μη βασική όψη επηρεάζεται από την παραμόρφωση της βασικής όψης. Αυτήν η ενότητα αποδεικνύει την ύπαρξη και το μέγεθος της εξάρτησης μεταξύ των όψεων.

Οι Εικόνες 30(a, b, c, d) και 31(a, b, c, d) απεικονίζουν το PSNR όταν κατά την μετάδοση των ακολουθιών χάνονται πακέτα μόνο της βασικής όψης. Παρόμοια με τα προηγούμενα πειράματα, το QoS του βίντεο αυξάνεται όσο αυξάνεται το μέγεθος του MTU. Συγκρίνοντας αυτά τα αποτελέσματα με τα προηγούμενα είναι εμφανής η εξάρτηση μεταξύ των όψεων ενώ επίσης δίνεται η δυνατότητα υπολογισμού της διάδοσης λάθους.



Εικόνα 30. PSNR αποτελέσματα: διαφορετικό ποσοστό απώλειας πακέτων - διαφορετικοί τύποι πακετοποίησης - διαφορετικά MTU μεγέθη – απώλεια στην βασική όψη - "flamenco"



Εικόνα 31. PSNR αποτελέσματα: διαφορετικό ποσοστό απώλειας πακέτων - διαφορετικοί τύποι πακετοποίησης - διαφορετικά MTU μεγέθη – απώλεια στην βασική όψη - "Objects".

Κεφάλαιο 5ο. Συμπεράσματα – Μελλοντική εργασία

Στόχος αυτής της πτυχιακής εργασίας ήταν η αξιολόγηση των επιδόσεων του 3D βίντεο streaming σε IP δίκτυα κάνοντας χρήση διαφορετικών τρόπων πακετοποίησης σύμφωνα με τα H.264/MVC και RTP πρότυπα. Ιδιαίτερο ενδιαφέρον δόθηκε στον τεμαχισμό του πλαισίου σε πολλαπλά NALUs και Fragmentation Units. Για την διάσωση των χαμένων headers, προτάθηκε ένας μηχανισμός διαχείρισης λαθών σε επίπεδο RTP ο οποίος χρησιμοποιεί τους headers των τεμαχίων που λήφθηκαν σωστά για την ανακατασκευή του H.264/AVC NALU Header. Για την αξιολόγηση των επιδόσεων λήφθηκε υπόψη ο πλεονασμός που προκαλείται από τις διαφορετικού τύπου δομές ωφέλιμου φορτίου, το Maximum Transmission Unit μέγεθος, ο αριθμός των αποκωδικοποιημένων πλαισίων και η ποιότητα του βίντεο (PSNR) τόσο της βασικής όσο για της μη βασικής όψης, κάτω από διαφορετικές συνθήκες δικτύου και διαφορετικούς τρόπους πακετοποίησης. Τελικά, κατασκευάστηκε μία πλατφόρμα δοκιμών, αποτελούμενη από έναν RTP packetizer και de-packetizer και έναν UDP/IP streamer και client.

Οι μελλοντικές επεκτάσεις της συγκεκριμένης πτυχιακής χωρίζονται σε δύο κατηγορίες. Καταρχήν, θα μπορούσε να κατασκευαστεί ένα MANE-based σύστημα και να συγκριθεί με το SST και το MST. Η λογική κατασκευής ενός τέτοιου συστήματος δεν διαφέρει πολύ από την λογική κατασκευής ενός MST ή SST συστήματος, αφού ουσιαστικά χρησιμοποιεί παράλληλα και τις δύο προαναφερθείσες μεθόδους. Η δεύτερη εργασία η οποία θα μπορούσε να γίνει είναι η δημιουργία ενός RTP /DCCP/IP Server/Client και να συγκριθεί με τον είδη υπάρχων RTP/UDP/IP. Συνοψίζοντας, ένα συνολικό μοντέλο μελλοντικής εργασίας θα μπορούσε να είναι η κατασκευή ενός MVC MANE-Based συστήματος το οποίο θα λειτουργούσε πάνω από RTP/DCCP/IP για να συγκριθεί με την πλατφόρμα αυτής της πτυχιακής.

Βιβλιογραφία

[1] Y. Chen et al, , “The Emerging MVC Standard for 3D Video Services”, Eurasip Journal on Advances in Signal Processing, No. 8, 2009, DOI: 10.1155/2009/786015

[2] G. B. Akar, A. M. Tekalp, C. Fehn and M. R, Civanlar, ‘Transport Methods in 3DTV—A Survey’, IEEE Trans. On Circuits and Systems for Video Technology, Vol. 17, 2007, DOI: 10.1109/ TCSVT.2007. 905365

[3] Y. Chen, P. Pandit, S. Yea, “WD 4 Reference software for MVC,” *JVT-AD207*, JVT of ISO/IEC MPEG & ITU-T VCEG, Geneva, Jan-Feb. 2009

[4] IETF Draft, “RTP Payload Format for MVC Video”, draft-ietf-wang-avt-rtp-mvc, October 2010

[5] IETF Draft, ”RTP Payload Format for H.264 Video”, draft-ietf-avt-rtp-rfc3984, February 2005

[6] IETF Draft, ”RTP Payload Format for SVC Video”, draft-ietf-avt-rtp-svc, October 2010

[7] IETF Draft, “RTP: A Transport Protocol for Real-Time Applications”, RFC3550, July 2003

[8] Kwang-deok Seo, Jin-soo Kim, Soon-heung Jung, and Jeong-ju Yoo, ”A Practical RTP Packetization Scheme for SVC Video Transport over IP Networks”, ETRI JourNAL, No. 2, April 2010, DOI: 10.4218/etrij.10.1409.0031

[9] S. Liu, Y. Chen, Y. Wang, M. Gabbouj, M. M. Hannuksela, H. Li, “Frame Loss Error Concealment For Multiview Video Coding”, June 2008,DOI: 10.1109/ISCAS.2008.4542206

[10] A. Hallapuro, M. Hannuksela, J. Lainema, M. Salmimaa, K. Ugur, K. Willner, Y. Wang, Y.Chen, “Nokia 3D Video & The MVC Standard”, nokia research center

[11] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G. Akar, G. Triantafyllidis, A. Koz, ”Coding Algorithms for 3DTV: A Survey”, IEEE Trans. Ob Circuits and Systems for Video Technology, Vol. 17, November 2007

[12] ITU-T H.241, “Extended video procedures and control signals for H.300-series terminals”, December 2009

[13] L., Rizzo, “Dummynet: a simple approach to the evaluation of network protocols,” ACM SIGCOMM Computer Communication Review, 27 (1), 31-41, 1997

[14] Renshou Dai, “Sage’s Video Quality Monitor”, February 13, 2008

[15] ITU-T , “Advanced video coding for generic audiovisual services” , May 2003

[16] A. Vetro, T. Wiegand, G. J. Sullivan, " Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard “,April 4, 2011

[17] B.W.Micallef, C.J.Debono, "An analysis on the effect of transmission errors in real-time H.264-MVC Bit-streams", 15th IEEE Mediterranean Electrotechnical Conference (MELECON), Apr 2010

[18] H. Schwarz, D. Marpe and T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”, IEEE Circuits and Systems for Video Technology, Vo. 17, No. 9, September 2007

[19] Tasos Dagiuklas, “3D Media over Future Internet: Current Status and Future Research Directions”, IJCSI International Journal of Computer Science Issues, January 2012

[20] ftp://ftp.ne.jp/KDDI/multiview

Συντομογραφίες

PSNR	Peak Signal-To-Noise Ratio
MVC	Multiple Video Coding
SVC	Scalable Video Coding
NAL	Network Abstraction Layer
JVT	Joint Video Team
IDR	Instantaneous Decoding Refresh
GOP	Group Of Pictures
FPS	Frames per Second
MB	Macro Block
CMB	Corresponding Macro Block
RTP	Real Time Transport Protocol
QoS	Quality of Service
SST	Single-Session Transmission
MST	Multi-Session Transmission
MANE	Media-Aware network element
ADTE	Adaptation Decision Taking Engine
DON	Decoding Order Number
UDP	User Datagram Protocol
NAT	Network Address Translation
MTU	Maximum Transmission Unit
SNU	Single NAL Unit
FU	Fragmentation Unit
MSE	Mean squared error

Παράρτημα Α

Διευθύνσεις internet

1. http://research.nokia.com/page/4988
2. http://www.3dtv-research.org/publicSwLibrary.php
3. http://www.acceptv.com/page/products_vqa
4. http://compression.ru/video/quality_measure/video_measurement_tool_en.html
5. http://info.iet.unipi.it/~luigi/dummynet/
6. http://www.3dtv.at/Index_en.aspx
7. http://en.wikipedia.org/wiki/Network_Abstraction_Layer#VCL_and_Non-VCL_NAL_Units
8. http://en.wikipedia.org/wiki/Real-time_Transport_Protocol
9. http://www.aero.org/publications/crosslink/winter2002/04.html
10. http://en.wikipedia.org/wiki/Peak_sigNAL-to-noise_ratio
11. http://www.lmt.ei.tum.de/pv2012/index.php
12. http://www.tridentcom.org/2012/

Παράρτημα Β

Πηγαίος κώδικας

- A. H.264/MVC MST RTP Packetizer

```
#include <stdlib.h>
#include<stdio.h>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

/*Declaration  of input and output file names*/
typedef struct userParams userParam_s;
typedef struct byteStreams byteStream_s;
typedef struct infosinfo_s;

typedef struct paddingBytespaddingByte_s;

/*Parse command line arguments*/
static int parseArg(int argc, char **argv, userParams *param)
/*Print packetizer usage information*/
static void usage(void)
/*Open files*/
static int openFiles(userParam_s *param,byteStreams *strByte)
/*Print info at the end of packetization*/
static void Info(infos info,int seq,int PacketizationMode)

/*Create and store bytes per NALU-metadata*/
int* NALUnitsBytes(FILE *file)

/*RTP MST MVC(AVC) Packetization*/

static unsigned char *GenerateRtpHeader(unsigned char seq)
/*Create NalUBytes file*/
static void CreateNalUBytes(byteStreams *strByte,int number,int view_id,int FUCounter,paddingBytes *
    paddingByte)
/*Packetize Parameter Sets(SNU Mode)*/
static int PacketizeSPS(int counter,unsigned char *Bitbuf,unsigned char *RtpBitbuf,unsigned char seq)
/*Create RTP Packets --> Single Nal Unit Payload Structure*/
static int PacketizeSingleNalUnits(int counter,unsigned char *Bitbuf,unsigned char *RtpBitbuf,unsigned
    char seq)

/*Create RTP Packets --> STAP-A Payload Structure*/
static int PacketizeAggregationNalUnits(int Counter,unsigned char *Bitbuf,unsigned char *RtpBitbuf,
    unsigned char seq,int StapFlag)
/*Create RTP Packets --> FU-A Payload Structure*/
static int PacketizeFragmentationNalUnits(int counter,unsigned char *Bitbuf,unsigned char *RtpBitbuf,
                                          unsigned char seq,int Fragments,int FuMode,paddingBytes *
    paddingByte)
```

```
653 int main(int argc, char **argv)
654 {
655
656     userParams param;
657     byteStreams strByte;
658     infos info;
659     paddingBytes paddingByte;
660     paddingByte.Bytes=0;
661     info.NumberOfFrames=0;
662     info.RtpOverHeadBytes=0;
663     info.NalOverHeadBytes=0;
664
665     unsigned char seq=0;//field in rtp header
666     int value;
667     int FirstBytesNumber;
668     int VideoMode;//AVC || MVC
669     int counter=-1;
670     int tempcounter=0;
671     int BitbufCounter=0;
672     int RtpBitbufCounter=0;
673     int n,j=0;
674     int PacketizationMode;//SNU || STAP-A || FU-A
675     int* BaseViewRtpPackets;//store Base's View bytes/NalU
676     int* NonBaseViewRtpPackets;//store non Base's View bytes/Nalu
677     int NalUCounter;
678     int FuMode=0;//If 1 then bytes/FU mode
679     int StapMode=0;//If 1 then 1 RTP packet / Frame
680     int StapCounter=0;
681     int NumberOfFragmentsPerNalU=0;
682     int NumberOfAggregationNals=0;
683     int NumberOfBytesPerNalU=0;
684     int MaxView;
685
686     int *viewid;//Viewid tree
687     int *NumberOfBytes;//Number Of Bytes per NALU
688     int *temp;
689
690     viewid=(int*)malloc(100000*sizeof(int));
691     NumberOfBytes=(int*)malloc(100000*sizeof(int));
692     temp=(int*)malloc(100000*sizeof(int));
693
694     unsigned char *Bitbuf; //MVC Payload
695     unsigned char *RtpBitbuf; //RTP payload
696     Bitbuf=(unsigned char*)malloc(100000*sizeof(int));
697     RtpBitbuf=(unsigned char*)malloc(100000*sizeof(int));
698
699     for(int i=0;i<100000;i++)
700     {
701         viewid[i]=9999999;
702         NumberOfBytes[i]=9999999;
703     }
704
705
706 printf("\n\t\t\t*********************\n\t\t\t* MVC/AVC RTP PACKETIZER"
707                                 " *\n\t\t\t*********************\n\n");
708     /* Parse command line arguments  */
709     if (!parseArg(argc, argv, &param))
710     {///Print packetizer usage information
711         usage();
```

```
712         exit(1);
713     }
714     /*Open coded files*/
715     VideoMode=openFiles(&param, &strByte);
716
717     /*Find the Number and the length of the NALU's (base view)*/
718     BaseViewRtpPackets=NALUnitsBytes(strByte.LeftView);
719
720     /*Find the length(in bytes) of SPS*/
721     FirstBytesNumber=BaseViewRtpPackets[0]+BaseViewRtpPackets[1]+BaseViewRtpPackets[2]+
        BaseViewRtpPackets[3]-9;
722
723     /*Remove SPS length from counter*/
724     NalUCounter=BaseViewRtpPackets[29999]-4;
725
726     printf("\nVideoMode=%d \nSPS Bytes =%d\n",VideoMode,FirstBytesNumber);
727
728     /*Create NumberOfBytes per NALU buffer (base view)*/
729     for(int i=0;i<=29995;i++)
730     {
731       NumberOfBytes[j]=BaseViewRtpPackets[i+4];
732       if(VideoMode==1)
733       j+=2;
734       else if(VideoMode==0)
735       j++;
736       else
737       printf("Undefined VideoMode");
738     }
739     fseek(strByte.LeftView,0,SEEK_SET);
740
741     /*If MVC*/
742     if (VideoMode==1)
743     {
744        /*Find the Number and the length of the NALU's (non base view )*/
745        NonBaseViewRtpPackets=NALUnitsBytes(strByte.RightView);
746
747        /*Create NumberOfBytes per NALU buffer (non base view)*/
748        j=1;
749        for(int i=0;i<=29995;i++)
750         {
751         NumberOfBytes[j]=NonBaseViewRtpPackets[i+1];
752         j+=2;
753         }
754
755       if(NalUCounter>NonBaseViewRtpPackets[29999]-1)
756           MaxView=0;//If base view > non base view(bytes)
757       else if(NalUCounter<NonBaseViewRtpPackets[29999]-1)
758           MaxView=1;//If base view < non base view
759       else
760           MaxView=2;//If base view == non base view
761
762       NalUCounter+=(NonBaseViewRtpPackets[29999]-1);
763       fseek(strByte.RightView,0,SEEK_SET); //go to beg
764     }
765
766       /*create ViewId tree*/
767       for(int i=0;i<NalUCounter;i++)
768       {
769         if (VideoMode==0)
770          {
771           viewid[i]=0;
772          }
773         else if(VideoMode==1 && MaxView==0)
774          {//non base view < base view (bytes)
775             viewid[i]=0;
776             if(NumberOfBytes[i+1]!=9999999)
```

```
777             {
778             viewid[i+1]=1;
779             }
780             else
781             {
782             NalUCounter++;
783             }
784
785            i++;
786          }
787        else if(VideoMode==1 && MaxView==1)
788        {//non base view > base view (bytes)
789           if(i==0)
790           {
791              viewid[0]=0;
792              i++;
793           }
794
795           viewid[i]=1;
796           if(NumberOfBytes[i+1]!=9999999)
797            {
798             viewid[i+1]=0;
799            }
800           else
801            {
802             NalUCounter++;
803            }
804            i++;
805          }
806        else if(VideoMode==1 && MaxView==2)
807        {//if non base view == base view (bytes)
808          viewid[i]=0;
809           i++;
810          viewid[i]=1;
811        }
812      }
813
814    int FrameCounter=0;
815    int NalUCounterArray[59999][2];
816    int newBaseViewFrame=0;
817    int newNALU=0;
818
819    for(int i=0;i<59999;i++)
820    {
821       NalUCounterArray[i][0]=0;
822       NalUCounterArray[i][1]=0;
823    }
824
825 /*If base view != non base view (No of NALU's)*/
826     if (VideoMode==1)
827       {
828       for(int i=0;i<NalUCounter;i++)
829       {
830         if(NumberOfBytes[i]==9999999 && viewid[i]==9999999 && viewid[i+1]!=9999999)
831         {//Erase Null row
832            NumberOfBytes[i]=NumberOfBytes[i+1];
833            viewid[i]=viewid[i+1];
834
835            //re-sort the array
836            for(int j=i+1;j<NalUCounter;j++)
837            {
838              NumberOfBytes[j]=NumberOfBytes[j+1];
839              viewid[j]=viewid[j+1];
840            }
841            NalUCounter--;
842         }
```

```
843     }
844
845 /*Find the Number of the Frames and the Number of the NALU's/frame*/
846
847     unsigned int *LeftArrayValues;
848     LeftArrayValues=(unsigned int*)malloc(15*sizeof(int));
849     LeftArrayValues[0]=0xb8;
850     LeftArrayValues[1]=0xe2;
851     LeftArrayValues[2]=0xe4;
852     LeftArrayValues[3]=0xe6;
853     LeftArrayValues[4]=0xe8;
854
855     LeftArrayValues[5]=0xba;
856     LeftArrayValues[6]=0xec;
857     LeftArrayValues[7]=0xee;
858     LeftArrayValues[8]=0xf0;
859     LeftArrayValues[9]=0xf2;
860
861     LeftArrayValues[10]=0xbd;
862     LeftArrayValues[11]=0xf6;
863     LeftArrayValues[12]=0xf8;
864     LeftArrayValues[13]=0xfa;
865     LeftArrayValues[14]=0xfc;
866
867     unsigned int *RightArrayValues;
868     RightArrayValues=(unsigned int*)malloc(15*sizeof(int));
869     RightArrayValues[0]=0xb4;
870     RightArrayValues[1]=0xd0;
871     RightArrayValues[2]=0xd1;
872     RightArrayValues[3]=0xd1;
873     RightArrayValues[4]=0xd2;
874
875     RightArrayValues[5]=0xb4;
876     RightArrayValues[6]=0xd3;
877     RightArrayValues[7]=0xd3;
878     RightArrayValues[8]=0xd4;
879     RightArrayValues[9]=0xd4;
880
881     RightArrayValues[10]=0xb5;
882     RightArrayValues[11]=0xd5;
883     RightArrayValues[12]=0xd6;
884     RightArrayValues[13]=0xd6;
885     RightArrayValues[14]=0xd7;
886
887      /*Base View*/
888     fseek(strByte.LeftView,FirstBytesNumber,SEEK_SET);
889
890     for(int j=0;j<NalUCounter;j++)
891       {
892         /*Read one NALU*/
893         n = fread(Bitbuf,NumberOfBytes[j],1,strByte.LeftView);
894
895         if(Bitbuf[14]==LeftArrayValues[newNALU])
896         {//Normally only first time enters
897             NalUCounterArray[newBaseViewFrame][0]++;
898             FrameCounter++;
899         }
900         else if(Bitbuf[14]==LeftArrayValues[newNALU+1])
901         {//If new frame
902             newBaseViewFrame++;
903             NalUCounterArray[newBaseViewFrame][0]++;
904             FrameCounter++;
905             if(newNALU>=14)
906             newNALU=0;
907             else
908             newNALU++;
```

```
909         }
910         else if(Bitbuf[14]==LeftArrayValues[0])
911         {//If new GOP
912             newBaseViewFrame++;
913             FrameCounter++;
914             NalUCounterArray[newBaseViewFrame][0]++;
915             newNALU=0;
916         }
917         else
918         {//If more NALUs for the same frame
919            NalUCounterArray[newBaseViewFrame][0]++;
920         }
921         /*Skip non Base's View NALU's*/
922          while(viewid[j+1]!=0)
923          j++;
924     }
925
926     int newNonBaseViewFrame=0;
927     newNALU=0;
928
929     NalUCounterArray[newNonBaseViewFrame][1]++;
930     FrameCounter++;
931     fseek(strByte.RightView,NumberOfBytes[1],SEEK_SET);
932
933     /*Non base view*/
934     for(int j=3;j<NalUCounter;j++)
935       {
936         /*Read one NALU*/
937         n = fread(Bitbuf,NumberOfBytes[j],1,strByte.RightView);
938
939         if(Bitbuf[17]==RightArrayValues[newNALU+1])
940         {//If new frame
941             newNonBaseViewFrame++;
942             NalUCounterArray[newNonBaseViewFrame][1]++;
943             FrameCounter++;
944             if(newNALU>=14)
945             newNALU=0;
946             else
947             newNALU++;
948         }
949         else if(Bitbuf[17]==RightArrayValues[0])
950         {//If new GOP
951             newNonBaseViewFrame++;
952             FrameCounter++;
953             NalUCounterArray[newNonBaseViewFrame][1]++;
954             newNALU=0;
955         }
956         else
957         {//If more NALUs in the same frame
958            NalUCounterArray[newNonBaseViewFrame][1]++;
959         }
960         /*Skip Base's View NALU's*/
961         while(viewid[j+1]!=1)
962             j++;
963
964     }
965     printf("Number Of Frames In Each View:%d\n",FrameCounter/2);
966
967     /*Set Packetization Order*/
968     int *tempViewId;
969     tempViewId=(int*)malloc(NalUCounter*sizeof(int));
970     int *tempNALuBytes;
971     tempNALuBytes=(int*)malloc(NalUCounter*sizeof(int));
972
973     int tempViewCounter=0;
974     int tempNALUCounter=0;
```

```
975     int tempNALUCounterNonBase=1;
976
977     for(int i=0;i<(FrameCounter/2);i++)
978     {//Execute for all frames
979       for(int j=0;j<NalUCounterArray[i][0];j++)
980       {
981         tempViewId[tempViewCounter]=viewid[tempNALUCounter];
982
983         tempNALuBytes[tempViewCounter]=NumberOfBytes[tempNALUCounter];
984         tempViewCounter++;
985
986         if(viewid[tempNALUCounter+1]==1)
987         tempNALUCounter+=2;
988         else if(viewid[tempNALUCounter+1]!=9999999)
989         tempNALUCounter++;
990
991       }
992
993        for(int j=0;j<NalUCounterArray[i][1];j++)
994         {
995         tempViewId[tempViewCounter]=viewid[tempNALUCounterNonBase];
996         tempNALuBytes[tempViewCounter]=NumberOfBytes[tempNALUCounterNonBase];
997
998         tempViewCounter++;
999
1000        if(viewid[tempNALUCounterNonBase+1]==0)
1001        tempNALUCounterNonBase+=2;
1002        else if(viewid[tempNALUCounterNonBase+1]!=9999999)
1003        tempNALUCounterNonBase++;
1004        }
1005
1006    }
1007
1008
1009    for(int i=0;i<NalUCounter-1;i++)
1010    {
1011        viewid[i]=tempViewId[i];
1012        NumberOfBytes[i]=tempNALuBytes[i];
1013    }
1014
1015    fseek(strByte.LeftView,0,SEEK_SET);
1016    fseek(strByte.RightView,0,SEEK_SET);
1017
1018
1019 }//If VideoMode == 1
1020    /*Create NALU info file*/
1021    if(VideoMode==1)
1022    {
1023        FILE *NaluInfo = NULL;
1024        char *NalUcharInfo="NaluInfo.txt";
1025
1026        if(fopen(NalUcharInfo, "r"))
1027        {
1028         remove(NalUcharInfo);
1029        }
1030        NaluInfo=fopen(NalUcharInfo, "w");
1031
1032        int i=0;
1033         fprintf(NaluInfo,"------ Number Of NALU's Per Frame -------\n");
1034         fprintf(NaluInfo,"No Base Nbase's\n");
1035        while(NalUCounterArray[i][0]!=0 && NalUCounterArray[i][1]!=0)
1036        {
1037         fprintf(NaluInfo,"%d  %d   %d \n",i,NalUCounterArray[i][0],NalUCounterArray[i][1]);
1038        i++;
1039        }
1040        i=0;
```

```
1041          fprintf(NaluInfo,"--------Number Of Bytes per NALU--------\n");
1042         fprintf(NaluInfo,"No View Length's\n");
1043         while(NumberOfBytes[i]!=9999999)
1044         {
1045         fprintf(NaluInfo,"%d  %d   %d\n",i,viewid[i],NumberOfBytes[i]);
1046         i++;
1047         }
1048         fclose(NaluInfo);
1049     }
1050
1051     /*Read parameter sets from file*/
1052     n = fread(Bitbuf,FirstBytesNumber,1,strByte.LeftView);
1053
1054     /*Packetize SPS--> always SNU payload structure*/
1055     RtpBitbufCounter=PacketizeSPS(FirstBytesNumber,Bitbuf,RtpBitbuf,seq);
1056
1057     /*Store NALU info in NalUBytes*/
1058     CreateNalUBytes(&strByte,RtpBitbufCounter,0,0,&paddingByte);
1059     /*Store SPS*/
1060     fwrite (RtpBitbuf,RtpBitbufCounter,1,strByte.OutLeft);
1061     info.RtpOverHeadBytes=RtpBitbufCounter-FirstBytesNumber;
1062     info.NalOverHeadBytes+=32;//4 NalU's Headers in SPS(32 bytes)
1063
1064    /*check for packetization mode*/
1065     printf("\n\n-------Press 0 for Signle Nal Units----------\n-------"
1066                       "Press 1 for Aggregation Packets-------\n-------"
1067                       "Press 2 for Fragmentation Units-------\nChoice:");
1068     scanf("%d",&PacketizationMode);
1069     if (PacketizationMode<0 || PacketizationMode>2)
1070     {
1071        printf("exiting.....");
1072        system("pause");
1073        exit(1);
1074     }
1075
1076    /*Create RTP Packets -> output 3 files(base view.264,non-base view.264,NalUBytes.txt)*/
1077
1078     counter=0;
1079     /*Single NAL Units Mode*/
1080     if (PacketizationMode==0)
1081      {
1082         while(viewid[counter]==0 || viewid[counter]==1)
1083          {//execute for all NALU's
1084            if(viewid[counter]==0)
1085             {//read one Nal Unit from LeftView
1086               n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1087               /*Create a RTP Packet*/
1088               RtpBitbufCounter=PacketizeSingleNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,seq);
1089               /*Increase sequence number*/
1090               seq++;
1091               /*Store NALU info to NalUBytes*/
1092               CreateNalUBytes(&strByte,RtpBitbufCounter,0,0,&paddingByte);
1093               /*Store RTP packet to file*/
1094               n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutLeft);
1095               info.NalOverHeadBytes+=6;//6 Nal Header Bytes(2 Headers(5,1));
1096
1097             }
1098            else if(viewid[counter]==1 &&  param.RightView!=NULL)
1099             { //read one NAL Unit from RightView
1100               n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.RightView);
1101               /*Create a RTP Packet*/
1102               RtpBitbufCounter=PacketizeSingleNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,seq);
1103               /*Increase sequence number*/
1104               seq++;
1105               /*Store NALU info to NalUBytes*/
1106               CreateNalUBytes(&strByte,RtpBitbufCounter,1,0,&paddingByte);
```

```
1107                /*Store RTP packet ot file*/
1108                n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutRight);
1109                info.NalOverHeadBytes+=9;//9 Nal Header Bytes(2 Headers(5,4))
1110
1111               }
1112                 info.NumberOfFrames+=1;
1113                 info.RtpOverHeadBytes=info.RtpOverHeadBytes+(RtpBitbufCounter-NumberOfBytes[counter]);

1114                 counter++;
1115          }
1116     }
1117
1118     /*Aggregation packets (STAP-A Mode)*/
1119    else if(PacketizationMode==1)
1120     {
1121       if(viewid[0]== 0 && viewid[1]==1 || VideoMode==0)
1122       {//For 1 NALU/Frame OR For AVC
1123         StapMode=0;
1124         printf("\nSet the Number of the  NALU's/RTP packet to aggregate[>1]:");
1125         scanf("%d",&NumberOfAggregationNals);
1126
1127         if (NumberOfAggregationNals<2)
1128         {
1129           printf("exiting.....");
1130           system("pause");
1131           exit(1);
1132         }
1133       }
1134      else if(VideoMode==1)
1135      {//For More NALU's/Frame
1136        StapMode=1;
1137      }
1138
1139      if(StapMode==0)
1140      {/*Each NALU contain a whole frame*/
1141       int NumberOfAggregationNalsArray[10000][2];
1142       int AggrNalsArrayCounterBase=0;
1143       int AggrNalsArrayCounterNonBase=0;
1144
1145        for(int i=0;i<10000;i++)
1146        {
1147          NumberOfAggregationNalsArray[i][0]=0;
1148          NumberOfAggregationNalsArray[i][1]=0;
1149        }
1150
1151        /*Create Packetization Order*/
1152        for(int i=0;i<NalUCounter;i++)
1153        {//for base view
1154           while(viewid[i]==0 && NumberOfAggregationNalsArray[AggrNalsArrayCounterBase][0]
     <NumberOfAggregationNals)
1155           {
1156             NumberOfAggregationNalsArray[AggrNalsArrayCounterBase][0]++;
1157             i++;
1158             if(VideoMode==1)
1159             i++;
1160           }
1161             i--;
1162             AggrNalsArrayCounterBase++;
1163        }
1164
1165        if(VideoMode==1)
1166        {
1167          for(int i=1;i<NalUCounter;i++)
1168          {//for non base view
1169            while(viewid[i]==1 && NumberOfAggregationNalsArray[AggrNalsArrayCounterNonBase][1]
     <NumberOfAggregationNals && VideoMode==1)
```

```
1170          {
1171           NumberOfAggregationNalsArray[AggrNalsArrayCounterNonBase][1]++;
1172           i+=2;
1173          }
1174           i--;
1175           AggrNalsArrayCounterNonBase++;
1176        }
1177      }
1178
1179      /*Create RTP packets*/
1180      int NonBaseCounter=1;
1181      for(int i=0;i<AggrNalsArrayCounterBase;i++)
1182      {
1183        for(int j=0;j<NumberOfAggregationNalsArray[i][0];j++)
1184          {//base view
1185            n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1186            if(j==0)
1187            {//If first NALU --> create RTP Header + STAP-A header + NALU Size header
1188               RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,
      seq,0);
1189            }
1190            else
1191            {//else --> create only NALU Size header
1192               RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,
      seq,1);
1193            }
1194
1195             CreateNalUBytes(&strByte,RtpBitbufCounter,0,0,&paddingByte);
1196             n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutLeft);
1197
1198             info.RtpOverHeadBytes=info.RtpOverHeadBytes+(RtpBitbufCounter-NumberOfBytes[counter]);
1199             info.NumberOfFrames++;
1200             info.NalOverHeadBytes+=6;//6 Nal Header Bytes((5(4+1),1) in every Nal
1201
1202             counter++;
1203             if(VideoMode==1)
1204             counter++;
1205          }
1206
1207          if(VideoMode==1)
1208          {//non base view
1209            for(int j=0;j<NumberOfAggregationNalsArray[i][1];j++)
1210            {
1211               n = fread(Bitbuf,1,NumberOfBytes[NonBaseCounter],strByte.RightView);
1212               if(j==0)
1213               {//If first NALU--> create RTP Header + STAP-A header + NALU Size header
1214                 RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[NonBaseCounter],Bitbuf,
      RtpBitbuf,seq,0);
1215               }
1216               else
1217               {//else --> create only NALU Size header
1218                 RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[NonBaseCounter],Bitbuf,
      RtpBitbuf,seq,1);
1219               }
1220
1221                 CreateNalUBytes(&strByte,RtpBitbufCounter,1,0,&paddingByte);
1222                 n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutRight);
1223
1224                 info.RtpOverHeadBytes=info.RtpOverHeadBytes+(RtpBitbufCounter-NumberOfBytes
      [NonBaseCounter]);
1225                 info.NumberOfFrames++;
1226                 info.NalOverHeadBytes+=9;//9 Nal Header Bytes((5(4+1),4)) in every Nal)
1227
1228                 NonBaseCounter+=2;
1229            }
1230          }//if(VideoMode==1)
```

```
1231          }//for(int i=0;i<AggrNalsArrayCounterBase;i++)
1232        }//if(StapMode==0)
1233
1234        else if(StapMode==1)
1235        {/*Each NALU contain part of a frame*/
1236          int AggrNalsArrayCounter=0;
1237          while(viewid[counter]==0 || viewid[counter]==1)
1238           {//execute for all NALU's
1239             if(viewid[counter]==0)
1240             {//base view
1241               n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1242             }
1243             else if(viewid[counter]==1)
1244             {//non base view
1245               n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.RightView);
1246             }
1247
1248             if(AggrNalsArrayCounter==0)
1249             {//If first NALU --> create RTP Header + STAP-A header + NALU Size header
1250               RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,seq,
      0);
1251             }
1252             else
1253             {//else 1--> create only NALU Size header
1254               RtpBitbufCounter=PacketizeAggregationNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf,seq,
      1);
1255             }
1256
1257             if(viewid[counter]==0)
1258              {//base view
1259                CreateNalUBytes(&strByte,RtpBitbufCounter,0,0,&paddingByte);
1260                n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutLeft);
1261
1262                infc.NalOverHeadBytes+=6;//6 Nal Header Bytes((5(4+1),1) in every Nal
1263              }
1264             else if(viewid[counter]==1)
1265              {//non base view
1266                CreateNalUBytes(&strByte,RtpBitbufCounter,1,0,&paddingByte);
1267                n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutRight);
1268
1269                infc.NalOverHeadBytes+=9;//9 Nal Header Bytes((5(4+1),4) in every Nal)
1270              }
1271
1272                infc.RtpOverHeadBytes=infc.RtpOverHeadBytes+(RtpBitbufCounter-NumberOfBytes[counter]);
1273                infc.NumberOfFrames++;
1274
1275                counter++;
1276                AggrNalsArrayCounter++;
1277                if((viewid[counter-1]==0 && viewid[counter]==1) || (viewid[counter-1]==1 && viewid
      [counter]==0))
1278                AggrNalsArrayCounter=0;
1279             }// while(viewid[counter]==0 || viewid[counter]==1)
1280        }// else if(StapMode==1)
1281
1282      }
1283      /*Fragmentation Units (FU-A Mode)*/
1284      else if(PacketizationMode==2)
1285        {
1286           if(((NumberOfBytes[0]+100)>1024 && (NumberOfBytes[0]-100)<1024)  ||
1287                ((NumberOfBytes[0]+100)>512 && (NumberOfBytes[0]-100)<512)  ||
1288                ((NumberOfBytes[0]+100)>1470 && (NumberOfBytes[0]-100)<1470)||
1289                ((NumberOfBytes[0]+100)>1470 && (NumberOfBytes[0]-100)<2048))
1290             {
1291               printf("No Need Of Fragmentation.Each NALU contain part of a frame."
1292                      "Choose Another Payload Structure\nexiting.....");
1293               system("pause");
```

```
1294            exit(1);
1295          }
1296       /*Fragments/NALU:
1297         All NALU's have the same number of FU's && all the FU's have different number of bytes*/
1298       /* Bytes/FU-A:
1299        All NALU's have different number of FU's && all the FU's have the same number of bytes*/
1300       printf("\n Select 0 to Set  Fragments/NALU mode OR 1 to set Bytes/FU-A mode:");
1301       scanf("%d",&FuMode);
1302
1303       if(FuMode==0)
1304       {
1305         printf("\nSet the Number of fragments per NALU [2-10]:");
1306         scanf("%d",&NumberOfFragmentsPerNalU);
1307
1308         if (NumberOfFragmentsPerNalU<2 || NumberOfFragmentsPerNalU>10)
1309             {
1310               printf("exiting.....");
1311               system("pause");
1312               exit(1);
1313             }
1314        }
1315        else if(FuMode==1)
1316        {
1317             printf("\nSet the Number of bytes per NALU:\n\n Press 0-->512bytes/RTP packet \t"
1318                       "Press 2-->1470bytes/RTP packet\n\n Press 1-->1024 Bytes/RTP Packet \t"
1319                     "Press 3-->2048bytes/RTP packet \n |---------------------------------|"
1320        "\n |For UDP/IP Packet Size(1024 bytes) |\n |---------------------------------|\n |"
1321             "    UDP/IP headers      (20 bytes) |\n |       Rtp Header        (46 bytes)|\n |"
1322           "-H.264/AVC NALU Header   (-1 byte) |\n |      Reserved Bytes       (4 byte)|\n |   "
1323        "Mvc Payload bytes    (955 bytes)|\n | --------------------------------|\n CHOICE:");
1324
1325             scanf("%d",&NumberOfBytesPerNalU);
1326             if(NumberOfBytesPerNalU==0)
1327             {
1328                 NumberOfBytesPerNalU=512;
1329             }
1330             else if(NumberOfBytesPerNalU==1)
1331                 NumberOfBytesPerNalU=1024;
1332             else if(NumberOfBytesPerNalU==2)
1333                 NumberOfBytesPerNalU=1470;
1334             else if(NumberOfBytesPerNalU==3)
1335                  NumberOfBytesPerNalU=2048;
1336             else
1337             {
1338               printf("exiting.....");
1339               system("pause");
1340               exit(1);
1341             }
1342        }
1343         else
1344             {
1345               printf("exiting.....");
1346               system("pause");
1347               exit(1);
1348            }
1349
1350        /*RTP Packetization*/
1351         while(viewid[counter]==0 || viewid[counter]==1)//check for view_id
1352         {//execute for all NALU's
1353          if(viewid[counter]==0)
1354           {//base view
1355             n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1356               if(FuMode==0)
1357                {
1358                 RtpBitbufCounter=PacketizeFragmentationNalUnits(NumberOfBytes[counter],Bitbuf,
RtpBitbuf,seq,NumberOfFragmentsPerNalU,FuMode,&paddingByte);
```

```
1359                seq+=NumberOfFragmentsPerNalU;
1360               }
1361              else if(FuMode==1)
1362               {
1363               RtpBitbufCounter=PacketizeFragmentationNalUnits(NumberOfBytes[counter],Bitbuf,RtpBitbuf
,   seq,NumberOfBytesPerNalU,FuMode,&paddingByte);
1364                seq+=RtpBitbuf[RtpBitbufCounter];
1365               }
1366
1367              /*Calculate the Number Of Fragments Per NalU */
1368              if(FuMode==1)
1369              {
1370                NumberOfFragmentsPerNalU=NumberOfBytes[counter]/(NumberOfBytesPerNalU-(46/*RTP Headers
*/+20/*Reserved for UDP header*/+4/*Reserved Bytes*/));
1371
1372                if ((((NumberOfFragmentsPerNalU * (NumberOfBytesPerNalU-(46/*RTP Headers*/+20/*
Reserved for UDP header*/+4/*Reserved Bytes*/)))+9)-NumberOfBytes[counter])<0)
1373                {/// there is one more last fragment needed to store all payload
1374                   NumberOfFragmentsPerNalU++;
1375                }
1376
1377              }
1378           CreateNalUBytes(&strByte,RtpBitbufCounter,0,NumberOfFragmentsPerNalU,&paddingByte);
1379           n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutLeft);
1380
1381           info.NalOverHeadBytes+=5;//5 Nal Header Bytes(2 Headers(5,0))
1382                                    /*note:H.264/AVC have been erased*/
1383
1384          }
1385          else if(viewid[counter]==1 &&  param.RightView!=NULL)
1386          {//non base view
1387           n = fread(Bitbuf,1,NumberOfBytes[counter],strByte.RightView);
1388             if(FuMode==0)
1389              {
1390                RtpBitbufCounter=PacketizeFragmentationNalUnits(NumberOfBytes[counter],Bitbuf,
RtpBitbuf,seq,NumberOfFragmentsPerNalU,FuMode,&paddingByte);
1391                seq+=NumberOfFragmentsPerNalU;
1392              }
1393             else if(FuMode==1)
1394              {
1395                RtpBitbufCounter=PacketizeFragmentationNalUnits(NumberOfBytes[counter],Bitbuf,
RtpBitbuf,seq,NumberOfBytesPerNalU,FuMode,&paddingByte);
1396                seq+=RtpBitbuf[RtpBitbufCounter];
1397              }
1398              if(FuMode==1)
1399              {
1400
1401                NumberOfFragmentsPerNalU=NumberOfBytes[counter]/(NumberOfBytesPerNalU-(46/*RTP Headers
*/+20/*Reserved for UDP header*/+4/*Reserved Bytes*/));
1402
1403                if ((((NumberOfFragmentsPerNalU * (NumberOfBytesPerNalU-(46/*RTP Headers*/+20/*
Reserved for UDP header*/+4/*Reserved Bytes*/)))+9)-NumberOfBytes[counter])<0)
1404                 {/// there is one more last fragment needed to store all payload
1405                     NumberOfFragmentsPerNalU++;
1406                 }
1407
1408              }
1409           CreateNalUBytes(&strByte,RtpBitbufCounter,1,NumberOfFragmentsPerNalU,&paddingByte);
1410           n=fwrite (RtpBitbuf,1,RtpBitbufCounter,strByte.OutRight);
1411
1412           info.NalOverHeadBytes+=8;//8 Nal Header Bytes(2 Headers(5,3))
1413                                    /*note:H.264/AVC have been erased*/
1414          }
1415
1416           info.RtpOverHeadBytes=info.RtpOverHeadBytes+(RtpBitbufCounter-NumberOfBytes[counter]);
1417           info.NumberOfFrames+=1;
```

```
1418            counter++;
1419
1420         }// while(viewid[counter]==0 || viewid[counter]==1)
1421
1422
1423     }//else if(PacketizationMode==2)
1424
1425       fclose(strByte.LeftView);
1426       fclose(strByte.OutLeft);
1427       fclose(strByte.NalUBytes);
1428     if(param.RightView!=NULL)
1429     {
1430       fclose(strByte.RightView);
1431       fclose(strByte.OutRight);
1432     }
1433
1434     Info(info,seq+3,PacketizationMode);//printf info
1435
1436    system("pause");
1437 }
1438
```

- B. MVC MST RTP Depacketizer

```
#include <stdlib.h>
#include<stdio.h>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;


/*Declaration  of input and output file names*/
typedef struct userParams userParam_s;
/*Declaration of Files*/
typedef struct byteStreams byteStream_s;
/*Recognition of the position of the frame in GOP */
typedef struct reconstructCounters reconstructCounters;
/*Reconstruction of lost NALU headers (FU-A Bytes)*/
typedef struct storeMvcHeadersstore_MvcHeaders;
/*Decodable frames counters*/
typedef struct numberOfDecodedFramesnumberOfDecoded_Frames;


/* Parse command line arguments  */
static int parseArg(int argc, char **argv, userParams *param)
/*Print usage information.*/
static void usage(void)
/*Open Files*/
static int openFiles(userParam_s *param,byteStreams *strByte)


/*Packetization mode*/
static int ModeCheck(int counter,unsigned char *RtpBitbuf,byteStreams *strByte)


/*DEPACKETIZATION*/
static int DepacketizeSPS(int FirstBytesNumber,unsigned char *RtpBitbuf,unsigned char *PayloadBitbuf)

static int DepacketizeSingleNalUnit(int counter,unsigned char *RtpBitbuf,unsigned char *PayloadBitbuf,
    FILE *p,int FrameCounter,numberOfDecodedFrames *numberOfDecodedFrames,int viewId)

static int DepacketizeAggregationNalUnits(int counter,unsigned char *RtpBitbuf,unsigned char *
    PayloadBitbuf,int RtpFlag,FILE *p,int FrameCounter,numberOfDecodedFrames *numberOfDecodedFrames,int
     viewId,reconstructCounters *reconCounter)

static int DepacketizeFragmentationNalUnits(int counter,unsigned char *RtpBitbuf,unsigned char *
    PayloadBitbuf,int Fragments,FILE *p,int FrameCounter,numberOfDecodedFrames *numberOfDecodedFrames,
    int viewId,reconstructCounters *reconCounter)

static int DepacketizeFragmentationNalBytesvsFu(int counter,unsigned char *RtpBitbuf,unsigned char *
    PayloadBitbuf,int Fragments,int NumberOfPaddingBytes,int viewId,reconstructCounters *reconCounter,
    storeMvcHeaders *storeMvcNALUHeader,int errorConceal,FILE *p,int FrameCounter,
    numberOfDecodedFrames *numberOfDecodedFrames)

int main(int argc, char **argv) {


    userParams param;
    byteStreams strByte;

    numberOfDecodedFrames numberOfDecodedFrames;
    numberOfDecodedFrames.baseViewCounter=numberOfDecodedFrames.nonBaseViewCounter=0;
    numberOfDecodedFrames.baseViewLostCounter=numberOfDecodedFrames.nonBaseViewLostCounter=0;

    reconstructCounters reconCounter;
    reconCounter.Base_View_FU_Indicator=reconCounter.non_Base_View_FU_Indicator=0;

    storeMvcHeaders storeMvcNALUHeader;
```

```
941     storeMvcNALUHeader.idrPrefixNALUHeader=(unsigned int*)malloc(5*sizeof(int));
942     storeMvcNALUHeader.nonIdrPrefixNALUHeader=(unsigned int*)malloc(5*sizeof(int));
943
944     /*Initialize prefix NALU Headers of Base View*/
945     storeMvcNALUHeader.idrPrefixNALUHeader[0]=storeMvcNALUHeader.nonIdrPrefixNALUHeader[0]=0x2e;
946     storeMvcNALUHeader.idrPrefixNALUHeader[1]=0x01;
947     storeMvcNALUHeader.nonIdrPrefixNALUHeader[1]=0x41;
948     storeMvcNALUHeader.idrPrefixNALUHeader[2]=storeMvcNALUHeader.nonIdrPrefixNALUHeader[2]=0x00;
949     storeMvcNALUHeader.idrPrefixNALUHeader[3]=0x07;
950     storeMvcNALUHeader.nonIdrPrefixNALUHeader[3]=0x03;
951     storeMvcNALUHeader.idrPrefixNALUHeader[4]=storeMvcNALUHeader.nonIdrPrefixNALUHeader[4]=0x80;
952
953     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader=(unsigned int*)malloc(5*sizeof(int));
954
955     /*Initialize prefix NALU Headers of non-Base View*/
956     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader[0]=0x2e;
957     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader[1]=0x41;
958     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader[2]=0x00;
959     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader[3]=0x41;
960     storeMvcNALUHeader.nonBaseViewPrefixNALUHeader[4]=0x80;
961
962     storeMvcNALUHeader.codedSliceOfMvcExtension=(unsigned int*)malloc(3*sizeof(int));
963
964     /*Initialize Coded Slice Of MVC Extension*/
965     storeMvcNALUHeader.codedSliceOfMvcExtension[0]=0x41;
966     storeMvcNALUHeader.codedSliceOfMvcExtension[1]=0x00;
967     storeMvcNALUHeader.codedSliceOfMvcExtension[2]=0x41;
968
969
970     int value;
971     int FirstBytesNumber;
972     int counter=-1;
973     int tempcounter=0;
974     int AggregationNalsCounter=0;
975     int RtpBitbufCounter=0;
976     int PayloadBitbufCounter=0;
977     int DePacketizationMode;
978     int n;
979     int Fragments=0;
980     int FuMode=0;
981     int VideoMode=1;
982
983     int * temp;
984     int * viewId;
985     int * NumberOfBytes;
986     int * NumberOfFu;
987     int * NumberOfPaddingBytes;
988
989     viewId=(int*)malloc(100000*sizeof(int));
990     NumberOfBytes=(int*)malloc(100000*sizeof(int));
991     temp=(int*)malloc(300000*sizeof(int));
992     NumberOfFu=(int*)malloc(100000*sizeof(int));
993     NumberOfPaddingBytes=(int*)malloc(100000*sizeof(int));
994
995     unsigned char *RtpBitbuf;
996     unsigned char *PayloadBitbuf;
997     RtpBitbuf=(unsigned char*)malloc(100000*sizeof(int));
998     PayloadBitbuf=(unsigned char*)malloc(100000*sizeof(int));
999
1000    printf("\n\t\t\t**************************\n\t\t\t* MVC/AVC RTP DEPACKETIZER *\n\t\t\t**********
        ****************\n\n");
1001
1002    /* Parse command line arguments  */
1003    if (!parseArg(argc, argv, &param))
1004    {/*Print depacketizer usage information*/
1005        usage();
```

```
        exit(1);
    }

    /*Open coded packetized files*/
    VideoMode=openFiles(&param, &strByte);

    /*Skip first rtp header*/
    strByte.NalUBytes.seekg(88);
    /*Temporarily store all values of NalUBytes on temp[]*/
    while (strByte.NalUBytes >> value)
    {
         if(counter==-1)
        {//Store parameter sets NoOfbytes
            FirstBytesNumber=value;
        }
        else
        {
            temp[counter]=value;
        }

       counter++;
     }

    /*Seperate Parameter Sets*/
    for(int j=0;j<counter-1;j+=4)
    {
        if(VideoMode==0 && temp[j]==1)
        {
          /*backwards compatibility*/
         /*If an MVC coded sequence -> If only  base view inserted
          ->erase the bytes of the non base view NALU's*/
        }
        else
        {
        viewId[tempcounter]= temp[j];
        NumberOfBytes[tempcounter]=temp[j+1];
        NumberOfFu[tempcounter]=temp[j+2];
        NumberOfPaddingBytes[tempcounter]=temp[j+3];
        tempcounter++;
        }
    }

    strByte.NalUBytes.close();

    /*Read Parameter Sets from LeftView.264*/
    n = fread(RtpBitbuf,FirstBytesNumber,1,strByte.LeftView);

    /*Depacketize Parameter Sets*/
    PayloadBitbufCounter=DepacketizeSPS(FirstBytesNumber,RtpBitbuf,PayloadBitbuf);
    fwrite (PayloadBitbuf,PayloadBitbufCounter,1,strByte.out3d);//Store to file

    printf("\nSearching for the payload structure of the bitstream:");
    /*Check for packetization mode*/
    DePacketizationMode=ModeCheck(NumberOfBytes[0],RtpBitbuf,&strByte);

    printf("\nNumber Of NALU's=%d\n",tempcounter);

    /*Create two files for two views -> number of the decodable frames */
    char *file1 = "Base_View_Frame_Information.txt";
    char *file2 = "Non_Base_View_Frame_Information.txt";
    FILE *base = NULL;
    FILE *nonbase = NULL;

    int baseViewFrameCounter=0;
    int nonBaseViewFrameCounter=0;
```

```
1072    if(fopen(file1, "r"))
1073    {
1074       remove(file1);
1075    }
1076
1077    base=fopen(file1, "w");
1078
1079    if(fopen(file2, "r"))
1080    {
1081      remove(file2);
1082    }
1083    nonbase=fopen(file2, "w");
1084
1085
1086    /*REMOVE RTP HEADERS AND JOIN 2 FILES*/
1087
1088    /*Single Nal Units Mode*/
1089     counter=0;
1090     if(DePacketizationMode==0)
1091      {
1092         /*SNU loop*/
1093         while(viewid[counter]==0 || viewid[counter]==1 )
1094         {
1095          /*Base View*/
1096          if(viewid[counter]==0)
1097          {//Read one packet
1098            if(counter!=0)//first nalU buffered at mode ckeck
1099            n = fread(RtpBitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1100
1101            /*Depacketize*/
1102            PayloadBitbufCounter=DepacketizeSingleNalUnit(NumberOfBytes[counter],RtpBitbuf,
    PayloadBitbuf, base,baseViewFrameCounter,&numberOfDecodedFrames,0);
1103            n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1104
1105            baseViewFrameCounter++;
1106          }
1107         /*non Base View*/
1108         else if(viewid[counter]==1 && VideoMode==1)
1109          {//Read one packet
1110              n = fread(RtpBitbuf,1,NumberOfBytes[counter],strByte.RightView);
1111              PayloadBitbufCounter=DepacketizeSingleNalUnit(NumberOfBytes[counter],RtpBitbuf,
    PayloadBitbuf, nonbase,nonBaseViewFrameCounter,&numberOfDecodedFrames,1);
1112
1113              n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1114
1115              nonBaseViewFrameCounter++;
1116          }
1117              counter++;
1118       }
1119     }
1120
1121    /*STAP-A Mode(Aggregation packets)*/
1122    else if(DePacketizationMode==1)
1123     {
1124         int NonBaseCounter=0;
1125         int tempflagBase=0;
1126         int tempflagNonBase=0;
1127         int NumberOfAggregationNals[9999][2];
1128         int sequenceMode=0;
1129
1130         int *RtpHeaderFlag;
1131         RtpHeaderFlag=(int *)malloc(100000*sizeof(int));
1132
1133         RtpHeaderFlag[0]=1;
1134         for(int i=1;i<100000;i++)
1135         {
```

```
1136              if((viewid[1]==0 && viewid[1-1]==1) || (viewid[1]==1 && viewid[1-1]==0))
1137              {
1138                RtpHeaderFlag[1]=1;
1139              }
1140              else
1141                RtpHeaderFlag[1]=0;
1142          }
1143
1144          printf("\n\nSearching for STAP-A Mode :\n");
1145
1146          /*For Different NALU Size -> change the bellow values*/
1147          if(((NumberOfBytes[0]+100)>1024 && (NumberOfBytes[0]-100)<1024) || ((NumberOfBytes[0]+100)>
      512 && (NumberOfBytes[0]-100)<512) ||  ((NumberOfBytes[0]+100)>1470 && (NumberOfBytes[0]-100)
      <1470))
1148           {
1149             sequenceMode=1;
1150             printf("Each Frame is encapsulated in more than one NALUs.\n\n\n");
1151           }
1152          else
1153          printf("Each Frame is encapsulated in one NALUs\n\n\n");
1154
1155          /*1st Case - create the Decoding Order Of the NALU's*/
1156          if(VideoMode==1 && sequenceMode==0)
1157          {
1158            AggregationNalsCounter=0;
1159
1160            for(int i=0;i<10000;i++)
1161            {
1162             NumberOfAggregationNals[1][0]=0;
1163             NumberOfAggregationNals[1][1]=0;
1164            }
1165
1166           for(int i=0;i<tempcounter;i++)
1167           {//tempcounter=number of NALS
1168             while(viewid[1]==0)
1169              {
1170              NumberOfAggregationNals[AggregationNalsCounter][0]++;
1171              i++;
1172              }
1173
1174             while(viewid[1]==1)
1175              {
1176               NumberOfAggregationNals[AggregationNalsCounter][1]++;
1177               i++;
1178              }
1179
1180             if(AggregationNalsCounter!=0)
1181             NumberOfAggregationNals[AggregationNalsCounter][0]+=NumberOfAggregationNals
      [AggregationNalsCounter-1][1];
1182
1183             NumberOfAggregationNals[AggregationNalsCounter][1]+=NumberOfAggregationNals
      [AggregationNalsCounter][0];
1184
1185             i--;
1186             AggregationNalsCounter++;
1187          }
1188
1189           NonBaseCounter=NumberOfAggregationNals[0][0];
1190        }
1191
1192        /*Depacketize*/
1193        /*STAP-A loop*/
1194        while(viewid[counter]==0 || viewid[NonBaseCounter]==1)
1195        {
1196            /*if base view*/
1197           if(viewid[counter]==0)
```

```
1198            {
1199              if(counter!=0)
1200              n = fread(RtpBitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1201
1202              /*Depacketize the aggregation unit*/
1203              if(RtpHeaderFlag[counter]==1)
1204              PayloadBitbufCounter=DepacketizeAggregationNalUnits(NumberOfBytes[counter],RtpBitbuf,
    PayloadBitbuf,1, base,baseViewFrameCounter,&numberOfDecodedFrames,0,&reconCounter);
1205
1206              else
1207              PayloadBitbufCounter=DepacketizeAggregationNalUnits(NumberOfBytes[counter],RtpBitbuf,
    PayloadBitbuf,0, base,baseViewFrameCounter,&numberOfDecodedFrames,0,&reconCounter);
1208
1209              /*Store the NALU to file*/
1210              n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1211
1212              baseViewFrameCounter++;
1213            }
1214
1215           /*non Base View*/
1216           if(VideoMode==1 && viewId[NonBaseCounter]==1)
1217           {
1218             n = fread(RtpBitbuf,1,NumberOfBytes[NonBaseCounter],strByte.RightView);
1219
1220             /*Depacketize the aggregation unit*/
1221             if(RtpHeaderFlag[NonBaseCounter]==1)
1222             PayloadBitbufCounter=DepacketizeAggregationNalUnits(NumberOfBytes[NonBaseCounter],
    RtpBitbuf,PayloadBitbuf,1,nonbase,nonBaseViewFrameCounter,&numberOfDecodedFrames,1,&reconCounter);
1223
1224             else
1225             PayloadBitbufCounter=DepacketizeAggregationNalUnits(NumberOfBytes[NonBaseCounter],
    RtpBitbuf,PayloadBitbuf,0,nonbase,nonBaseViewFrameCounter,&numberOfDecodedFrames,1,&reconCounter);
1226
1227             /*Store the NALU to file*/
1228             n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1229
1230             nonBaseViewFrameCounter++;
1231
1232             /*If all Frames have one NALU --> find the next aggregation unit */
1233             if(sequenceMode==0)
1234             {
1235                if(counter<(NumberOfAggregationNals[tempflagBase][0]-1))
1236                {
1237                  counter++;
1238                }
1239                else
1240                {
1241                  counter=NumberOfAggregationNals[tempflagBase][1];
1242                  tempflagBase++;
1243                }
1244
1245                if(NonBaseCounter<(NumberOfAggregationNals[tempflagNonBase][1])-1)
1246                {
1247                  NonBaseCounter++;
1248                }
1249                else
1250                {
1251                  tempflagNonBase++;
1252                  NonBaseCounter=NumberOfAggregationNals[tempflagNonBase][0];
1253                }
1254               }
1255
1256               /*If each frame have more than 1 NALU's */
1257               else
1258               {
1259                  counter++;
```

```
1260                    NonBaseCounter++;
1261                  }
1262           }
1263             else if(sequenceMode)
1264             {
1265             counter++;
1266             NonBaseCounter++;
1267             }
1268             else
1269             counter++;
1270
1271           }
1272    }
1273
1274    /*FU-A Mode(Fragmentation Units)*/
1275    else if(DePacketizationMode==2)
1276     {
1277       printf("\n\nSearching for FuMode:\n");
1278
1279       Fragments=NumberOfFu[0];
1280
1281       for(int i=1;i<30000;i++)
1282         {
1283           if(NumberOfFu[i]!=Fragments && NumberOfFu[i]!=(-842150451))
1284              FuMode=1;
1285         }
1286
1287       int errorConceal=0;
1288       if(FuMode==1)
1289       {
1290        printf("Bytes/FU Mode-->All RTP packets have the same number of bytes\n");
1291        printf("Error Resilient Algoritm (0-false / 1-true):");
1292        scanf("%d",&errorConceal);
1293        printf("\n\n\n");
1294       }
1295       else
1296       printf("FU/NALU Mode-->All NALS have the same number of fragments:%d\n\n\n",NumberOfFu[0]);
1297
1298       /*FU loop*/
1299       while(viewId[counter]==0 || viewId[counter]==1)
1300        {
1301          /*base view*/
1302          if(viewId[counter]==0)
1303            {
1304              if(counter!=0)//first nal buffered at mode ckeck
1305                n = fread(RtpBitbuf,1,NumberOfBytes[counter],strByte.LeftView);
1306
1307              /*Depacketize the NALU's of one frame*/
1308              if(FuMode==0)//if Fu/NALU mode
1309                PayloadBitbufCounter=DepacketizeFragmentationNalUnits(NumberOfBytes[counter],
RtpBitbuf,
1310                PayloadBitbuf,NumberOfFu[counter],base,baseViewFrameCounter,&numberOfDecodedFrames,0,
&reconCounter);
1311
1312             else if(FuMode==1)//if bytes/fu mode
1313               PayloadBitbufCounter=DepacketizeFragmentationNalBytesvsFu(NumberOfBytes[counter],
RtpBitbuf,PayloadBitbuf,NumberOfFu[counter],NumberOfPaddingBytes[counter],0,&reconCounter,&
storeMvcNALUHeader,errorConceal,base,baseViewFrameCounter,&numberOfDecodedFrames);
1314
1315
1316               n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1317               baseViewFrameCounter++;
1318            }
1319          /*non Base View*/
1320           else if(viewId[counter]==1 && VideoMode==1)
1321             {
```

```
1322                n = fread(RtpBitbuf,1,NumberOfBytes[counter],strByte.RightView);
1323
1324               /*Depacketize*/
1325                if(FuMode==0)//if Fu/NALU mode
1326                  PayloadBitbufCounter=DepacketizeFragmentationNalUnits(NumberOfBytes[counter],
      RtpBitbuf,PayloadBitbuf,NumberOfFu[counter],nonbase,nonBaseViewFrameCounter,&numberOfDecodedFrames,
      1,&reconCounter);
1327
1328                else if(FuMode==1)//if bytes/fu mode
1329                  PayloadBitbufCounter=DepacketizeFragmentationNalBytesvsFu(NumberOfBytes[counter],
      RtpBitbuf,PayloadBitbuf,NumberOfFu[counter],NumberOfPaddingBytes[counter],1,&reconCounter,&
      storeMvcNALUHeader,errorConceal, nonbase,nonBaseViewFrameCounter,&numberOfDecodedFrames);
1330
1331                  n=fwrite (PayloadBitbuf,1,PayloadBitbufCounter,strByte.out3d);
1332                  nonBaseViewFrameCounter++;
1333               }
1334                counter++;
1335          }//end of while
1336    }//end of if packetization mode
1337
1338
1339     fprintf(base,"\nTotal Number Of Decoded NALU's: %d\n",numberOfDecodedFrames.baseViewCounter);
1340     fprintf(nonbase,"\nTotal Number Of Decoded NALU's: %d\n",numberOfDecodedFrames.nonBaseViewCounter)
      ;
1341     fprintf(base,"\nTotal Number Of Lost NALU's : %d\n",numberOfDecodedFrames.baseViewLostCounter);
1342     fprintf(nonbase,"\nTotal Number Of Lost NALU's : %d\n",numberOfDecodedFrames.
      nonBaseViewLostCounter);
1343     fclose(base);
1344     fclose(nonbase);
1345
1346
1347      fclose(strByte.LeftView);
1348      fclose(strByte.out3d);
1349      //remove(param.LeftView);
1350      //remove(param.NalUBytes);
1351      if(((fopen(param.RightView, "rb")) != NULL) && param.RightView!=NULL)
1352        {
1353        fclose(strByte.RightView);
1354        //remove(param.RightView);
1355        }
1356
1357       printf("\nRTP DEPACKETIZATION FINISHED");
1358       if(DePacketizationMode==0)
1359         {
1360          printf("\n-----Single Nal Units Depacketized------\n");
1361         }
1362       else if(DePacketizationMode==1)
1363         {
1364          printf("\n-----STAP-A Units Depacketized------\n");
1365
1366         }
1367       else if(DePacketizationMode==2)
1368        {
1369          printf("\n-----FU-A Units Depacketized------\n");
1370        }
1371
1372    exit(1);
1373 }
1374
1375
1376
```

- MVC MST UDP/IP Server

```
1  import javax.swing.JFrame;
2
3  public class QuoteServer
4  {
5    public static void main(String[] args)
6    {
7      GUI gui=new GUI();
8      gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9      gui.setSize(210,130);
10     gui.setAlwaysOnTop(true);
11     gui.setVisible(true);
12   }
13
14 }
```

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import javax.swing.JTextField;
4  import javax.swing.JButton;
5  import java.awt.FlowLayout;
6
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9
10 import javax.swing.JOptionPane;
11
12 public class GUI extends JFrame
13 {
14     private JLabel hostLabel;
15     private JLabel packetLabel;
16
17     private JTextField hostField;
18     private JTextField packetField;
19
20     private JButton button;
21
22     public GUI()
23     {
24      /*GUI Title*/
25      super("MVC MST RTP UDP/IP Streamer");
26
27      FlowLayout layout=new FlowLayout();
28      layout.setAlignment(FlowLayout.LEADING);
29      setLayout(layout);
30
31 /*--------Initialization of the components---------*/
32
33      hostLabel  =new JLabel("Server IP:      ");
34      packetLabel=new JLabel("Packet Size:");
35
36      hostField=new JTextField(10);
37      hostField.setText("192.168.1.2");
38      hostField.setToolTipText("Add a valid Server IP");
```

```
39
40      packetField=new JTextField(10);
41      packetField.setText("1024");
42      packetField.setToolTipText
43                ("Add the desired packet size in Bytes");
44
45      button=new JButton
46              ("           Start Streamer            ");
47
48
49  /*--------add components to JFrame/*--------*/
50
51      add(hostLabel);
52      add(hostField);
53      add(packetLabel);
54      add(packetField);
55      add(button);
56
57       /*declaration of the button handler*/
58       ButtonHandler handler=new ButtonHandler();
59       /*add action listener(on click)*/
60       button.addActionListener(handler);
61
62
63      }
64
65  /*--Creation of an internal class to handle events--*/
66    private class ButtonHandler implements ActionListener
67    {
68      String host;
69      int packetSize;
70
71      public void actionPerformed(ActionEvent event)
72      {
73
74       if(hostField.getText().isEmpty() ||
75                          packetField.getText().isEmpty())
76       {
```

```
77          JOptionPane.showMessageDialog(null,
78                          "Please fill the fields","ERROR",0);
79      }
80      else
81      {
82 /*--------Store  user input data--------*/
83      host= hostField.getText();
84     packetSize=Integer.parseInt(packetField.getText());
85
86      /*Start The Server*/
87      EstablishConnection establish=
88                  new EstablishConnection(host,packetSize);
89      /*Create a TCP/IP connection and Send
90       Sequence Paramenter Set and NALU Information */
91      establish.sendParameterSets();
92       /*Create UDP/IP connections*/
93       establish.establishUDPConnection();
94       }
95     }
96   }
97 }
```

```
 1 import java.io.File;
 2
 3 import java.io.InputStream;
 4 import java.io.OutputStream;
 5 import java.io.FileInputStream;
 6 import java.io.BufferedInputStream;
 7
 8 import java.net.Socket;
 9 import java.net.ServerSocket;
10 import java.net.SocketAddress;
11 import java.net.InetSocketAddress;
12
13 import java.util.concurrent.Executors;
14 import java.util.concurrent.ExecutorService;
15
16 import java.net.UnknownHostException;
17 import java.io.IOException;
18
19 import javax.swing.JOptionPane;
20
21 public class EstablishConnection
22 {
23     private   String  host;
24     private   int     avcMvcChoice;
25     private   int     packetSize;
26
27     private   long    leftCounter;
28     private   long    rightCounter;
29
30     protected File    leftfile;
31     protected File    rightfile;
32     protected File    nalUBytes;
33
34     QuoteServerThread baseViewThread;
35     QuoteServerThread non_baseViewThread;
36
37    public EstablishConnection(String host,
38                                          int packetSize
```

```
39      {
40        nalUBytes    = new File ("NalUBytes.txt");
41        leftfile     = new File("leftfile.264");
42        rightfile    = new File("rightfile.264");
43
44        if((nalUBytes.exists() || leftfile.exists() ||
45                                  rightfile.exists())==false
46        {
47          JOptionPane.showMessageDialog(null,
48                        "No Video Files Found","ERROR",0)
49          System.exit(1);
50        }
51
52         this.avcMvcChoice=0;
53         this.host=host;
54         this.packetSize=packetSize;
55         leftCounter=0;
56         rightCounter=0;
57      }
58
59
60  protected void recognizeAvcOrMvcTrasmission
61                                                 (Socket sock
62   {
63     try
64      {
65        System.out.println("Receiving information...");
66
67        byte [] mybytearray  = new byte [1];
68        InputStream is = sock.getInputStream();
69
70        is.read(mybytearray,0,1);
71        avcMvcChoice=mybytearray[0];
72        if(avcMvcChoice==0)
73        System.out.println("MVC Transmission");
74        if(avcMvcChoice==1)
75        System.out.println("Avc Transmission");
76        is.close();
```

```
 77    }
 78   catch (UnknownHostException ex)
 79   {
 80     JOptionPane.showMessageDialog(null,
 81                         "UnknownHostException","ERROR",0)
 82     System.exit(1);
 83   }
 84   catch (IOException ex)
 85   {
 86     JOptionPane.showMessageDialog(null,"IOException-1"
 87                                                 ,"ERROR",0)
 88     System.exit(1);
 89   }
 90
 91 }
 92
 93 protected void sendParameterSets()
 94 {
 95     int     SpsLength;
 96     byte [] nalUBytesArray  =
 97                       new byte [(int)nalUBytes.length()]
 98     byte [] SpsArray;
 99     byte [] SeparateFlag={22,22};
100
101     ServerSocket servsock;
102
103     try
104     {
105       servsock = new ServerSocket();
106       SocketAddress sa=new InetSocketAddress(host,2050)
107       servsock.bind(sa);
108       System.out.println("Server Listening to:"
109                               +servsock.getInetAddress())
110
111       System.out.println("Tcp Connection---Waiting...")
112       //accept a connection
113       Socket sock = servsock.accept();
114       //reveive transmission information
```

```
115     recognizeAvcOrMvcTrasmission(sock);
116
117     sock = servsock.accept();
118     System.out.println("Accepted connection : "+sock)
119
120     /*Read Nal Unit Bytes*/
121     FileInputStream fis =
122                         new FileInputStream(nalUBytes)
123     BufferedInputStream bis =
124                          new BufferedInputStream(fis)
125
126     bis.read(nalUBytesArray,0,nalUBytesArray.length);
127     /*Read Sequence Parameter Set*/
128     String[] valueStr =
129       new String(nalUBytesArray).trim().split("\\s+")
130
131     SpsLength = Integer.parseInt(valueStr[1]);
132     SpsArray=new byte [SpsLength];
133
134     FileInputStream fis2 =
135                         new FileInputStream(leftfile);
136     BufferedInputStream bis2 =
137                         new BufferedInputStream(fis2);
138
139     bis2.read(SpsArray,0,SpsLength);
140     /*Dont read again same bytes*/
141     leftCounter+=SpsLength;
142
143     /*Send Packets*/
144     OutputStream os = sock.getOutputStream();
145
146     System.out.println
147                  ("Sending SequenceParameterSet ...");
148     os.write(SpsArray,0,SpsLength);
149     os.flush();
150
151     /*Seperate information*/
152     os.write(SeparateFlag,0,SeparateFlag.length);
```

```
153
154     os.flush();
155
156     System.out.println("Sending NalUBytes ...");
157     os.write(nalUBytesArray,0,nalUBytesArray.length);
158     os.flush();
159
160     os.close();
161     sock.close();
162     servsock.close();
163     fis.close();
164     bis.close();
165     fis2.close();
166     bis2.close();
167   }
168   catch (IOException ex)
169   {
170      JOptionPane.showMessageDialog(null,
171               "Cannot Bind to the Specified Socket"
172                     + "Address: "+host,"ERROR",0);
173      System.exit(1);
174   }
175  }
176
177  protected void establishUDPConnection()
178  {
179     if(avcMvcChoice==0)
180         {/*Start 2 Threads for MVC transmission*/
181           QuoteServerThread base=
182                              new QuoteServerThread
183                              (packetSize,leftfile,
184                    leftCounter,4445,"Base View");
185           QuoteServerThread non_base=
186                              new QuoteServerThread
187                              (packetSize ,rightfile,
188               rightCounter,4446,"Non Base View");
189           ExecutorService threadExecutor=
190                    Executors.newFixedThreadPool(2);
```

```
            /*Start Threads*/
            threadExecutor.execute(base);
            threadExecutor.execute(non_base);

           /*Shut down Threads*/
            threadExecutor.shutdown();

            while(!threadExecutor.isTerminated())
            {
              /*Wait until threads are terminated*/
            }

          }
          else
          {/*Start 1 Thread for MVC transmission*/
            QuoteServerThread base=
              new QuoteServerThread(packetSize,leftfile
                          ,leftCounter,4445,"Base View")

            ExecutorService threadExecutor=
                        Executors.newFixedThreadPool(1)
            threadExecutor.execute(base);

            /*Shut down Threads */
            threadExecutor.shutdown();

            while(!threadExecutor.isTerminated())
            {
              /*Wait until threads are terminated*/
            }

          }
          restartServer();
          System.err.println("Restarting Server");
    }

    private void restartServer()
    {
        EstablishConnection establish=
             new EstablishConnection(host,packetSize);

        establish.sendParameterSets();

        establish.establishUDPConnection();
    }

}
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.BufferedInputStream;

import java.net.DatagramPacket;
import java.net.MulticastSocket;

import java.io.IOException;
import java.net.SocketTimeoutException;

import javax.swing.JOptionPane;

public class QuoteServerThread implements Runnable
{
    private   String  threadName;
    private   int     packetSize;
    private   boolean moreQuotes;

    byte[] receiverBuf;
    byte[] buf;

    DatagramPacket packet1;
    DatagramPacket packet2;
    FileInputStream fis;
    BufferedInputStream bis;
    protected MulticastSocket socket;

    protected File    file;

public QuoteServerThread(int packetSize,File file,
          Long counter,int port,String threadName)
{
      this.packetSize=packetSize;
      this.file=file;
      this.file= file;
      this.threadName=threadName;

      receiverBuf=new byte[16];
```

```
39        buf = new byte[packetSize];
40
41        packet1 = new DatagramPacket(receiverBuf,
42                                     receiverBuf.length);
43        packet2 = new DatagramPacket(buf,
44                                           buf.length);
45
46        moreQuotes=true;
47
48        try
49        {
50          fis = new FileInputStream(file);
51          bis = new BufferedInputStream(fis);
52          bis.skip(counter);
53        } catch (IOException ex)
54        {
55          JOptionPane.showMessageDialog(null,
56                                   "File Not Found "
57                         +Thread.currentThread(),
58                                           "ERROR",0);
59          System.exit(1);
60        }
61
62        try
63          {
64            socket  = new MulticastSocket(port);
65          }
66        catch (IOException ex)
67          {
68            JOptionPane.showMessageDialog(null,
69                                 "IOException-3 "+
70                               Thread.currentThread()
71                                        ,"ERROR",0);
72             System.exit(1);
73
74          }
75}
76
```

```
77
public void run()
79 {
80      /*Set Threads name*/
81      Thread.currentThread().setName(threadName);
82
83      System.out.println("Sending Sequense of "+
84                 Thread.currentThread().getName()
85                                 +" through  Udp");
86      while (moreQuotes)
87      {
88         socketConnection();//send one packet
89      }
90
91         System.out.println("End Of Transmission in "+
92                    Thread.currentThread().getName());
93         informClient();
94         socket.close();
95         socket.disconnect();
96 }
97
98
99 protected void getNextPacket()
100 {
101     try
102     {
103       if(bis.read(buf,0,packetSize)==-1)
104        {/*If EOF stop transmission*/
105         moreQuotes = false;
106        }
107      }
108     catch (IOException ex)
109     {
110      JOptionPane.showMessageDialog(null,
111                         "IOException-4 in "+
112           Thread.currentThread().getName(),
113                                    "ERROR",0);
114     }
```

```
115 }
116
117 protected void socketConnection()
118 {
119   /*Send Bytes trough  udp*/
120   try
121   {
122     /* Receive request*/
123      try
124      {
125       /*Set SoTimeout ,
126        * If request lost->connection end->restart*/
127        socket.setSoTimeout(5000);
128        socket.receive(packet1);
129
130       /*If receiver instead of a normal request(0,0)
131                          * sent (1,1)->*Connection P*/
132        if(receiverBuf[0]==1 && receiverBuf[1]==1)
133         {
134           System.out.println("Connection Problem in "
135                     +Thread.currentThread().getName()
136                                             +" Channel");
137           moreQuotes=false;
138         }
139
140    /*create packet*/
141         getNextPacket();
142    /* figure out response->send the response
143            * to the client at "address" and "port"*/
144
145         packet2.setData(buf);
146         packet2.setLength(buf.length);
147         packet2.setAddress(packet1.getAddress());
148         packet2.setPort(packet1.getPort());
149
150
151         socket.send(packet2);
152       }
```

```
153
154      catch (SocketTimeoutException ex)
155      {
156     System.err.println("User stopped the transmission
157                + "in "+Thread.currentThread().getName(
158                                                  +" Channel")
159        moreQuotes = false;
160      }
161
162    }
163   catch (IOException e)
164   {
165     System.out.println("Packet Lost in "+
166                    Thread.currentThread().getName());
167   }
168 }
169
170 public void informClient()
171 {
172    for(int i=0;i<packetSize;i++)
173        buf[i]=0;
174    packet2.setData(buf);
175    packet2.setLength(buf.length);
176    packet2.setAddress(packet1.getAddress());
177    packet2.setPort(packet1.getPort());
178    try
179    {
180       socket.send(packet2);
181    }
182    catch (IOException ex)
183    {
184      JOptionPane.showMessageDialog(null,
185                                  "IOException","ERROR",0);
186      System.exit(1);
187    }
188  }
189
190 }
```

- Δ. MVC MST UDP/IP Client

```
1  import javax.swing.JFrame;
2
3  public class QuoteClient
4  {
5
6   public static void main(String[] args)
7   {
8      GUI gui=new GUI();
9      gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10     gui.setSize(405,290);
11     gui.setAlwaysOnTop(true);
12     gui.setVisible(true);
13  }
14
15 }
```

```java
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JCheckBox;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import java.awt.FlowLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JOptionPane;
import java.io.File;

public class GUI extends JFrame
{
    private JLabel videoLabel;
    private JLabel plrLabel;
    private JLabel payloadTypeLabel;
    private JLabel numberOfAggrOrFragmLabel;
    private JLabel repeatTransLabel;
    private JLabel serverIpLabel;
    private JLabel ipPacketLabel;
    private JLabel blank;

    private JTextField videoField;
    private JTextField numberOfAggrOrFragmField;
    private JTextField repeatTransField;
    private JTextField serverIpField;
    private JTextField ipPacketField;

    private JCheckBox nalsForFrame;
    private JCheckBox avcOrMvc;

    private JRadioButton plrNone;
    private JRadioButton plrOne;
    private JRadioButton plrTwo;
```

```
39      private JRadioButton plrFive;
40      private JRadioButton plrTen;
41
42      private ButtonGroup plrButtonGroup;
43
44      private JRadioButton payloadTypeSNU;
45      private JRadioButton payloadTypeSTAP_A;
46      private JRadioButton payloadTypeFU_A;
47      private JRadioButton payloadTypeFU_A_Bytes;
48
49      private ButtonGroup payloadTypeGroup;
50
51      private JButton button;
52
53      String black1;
54
55      public GUI()
56      {
57       super("MVC MST RTP UDP/IP Client");
58
59       FlowLayout layout=new FlowLayout();
60       layout.setAlignment(FlowLayout.LEADING);
61       setLayout(layout);
62
63  /*--------------------Initialization of the
64              * components--------------------*/
65
66       videoLabel=new JLabel
67               ("Type The Name Of The Video Sequence:");
68       plrLabel=new JLabel("Packet Loss Rate:");
69       payloadTypeLabel=new JLabel("RTP Payload Type:");
70       numberOfAggrOrFragmLabel=new JLabel
71        ("If STAP / FU Mode then "+ "Type the number:");
72       repeatTransLabel =new JLabel
73           ("Type the number of the Re-Transmissions:");
74       serverIpLabel=new JLabel("Server IP Address:");
75       ipPacketLabel=new JLabel("Size Of IP Packet:  ");
76       black1="                                        ";
```

```
77      blank=new JLabel(blackl);
78
79      videoField=new JTextField(10);
80      videoField.setToolTipText
81                      ("Insert the name of the Video");
82      videoField.setText("Uli");//Suburban_Bridge
83
84      numberOfAggrOrFragmField=new JTextField(10);
85      numberOfAggrOrFragmField.setToolTipText
86                    ("Number of Aggregation Units/NAL "
87                      + "or Fragmentation Units/NAL:");
88     // numberOfAggrOrFragmField.setText("1470");
89
90      repeatTransField=new JTextField(3);
91      repeatTransField.setToolTipText
92          ("If you want more than 1 re-Transmissions "
93                              + "type the number here");
94      repeatTransField.setText("5");
95
96      serverIpField=new JTextField(10);
97      serverIpField.setText("192.168.1.2");
98
99      ipPacketField=new JTextField(10);
100     ipPacketField.setText("1024");
101
102     nalsForFrame=new JCheckBox("More NAL Units / "
103                         + "Frame (Default: 1)",true);
104
105     avcOrMvc=new JCheckBox("2D (Default: 3D)",true);
106
107
108     plrNone=new JRadioButton("0%",true);
109     plrOne=new JRadioButton("1%",false);
110     plrTwo=new JRadioButton("2%",false);
111     plrFive=new JRadioButton("5%",false);
112     plrTen=new JRadioButton("10%",false);
113
114
```

```
115     plrButtonGroup=new ButtonGroup();
116     plrButtonGroup.add(plrNone);
117     plrButtonGroup.add(plrOne);
118     plrButtonGroup.add(plrTwo);
119     plrButtonGroup.add(plrFive);
120     plrButtonGroup.add(plrTen);
121
122
123     payloadTypeSNU =new JRadioButton("SNU",false);
124     payloadTypeSTAP_A=new JRadioButton
125                                     ("STAP-A",true);
126     payloadTypeFU_A=new JRadioButton("FU-A",false);
127     payloadTypeFU_A_Bytes=new JRadioButton
128                                 ("FU-A Bytes",false);
129
130     payloadTypeGroup=new ButtonGroup();
131     payloadTypeGroup.add(payloadTypeSNU);
132     payloadTypeGroup.add(payloadTypeSTAP_A);
133     payloadTypeGroup.add(payloadTypeFU_A);
134     payloadTypeGroup.add(payloadTypeFU_A_Bytes);
135
136     button=new JButton(blackl+""
137                 + "Start the Transmission"+blackl);
138
139/*-----------add components to JFrame------------*/
140
141     add(videoLabel);
142     add(videoField);
143     add(avcOrMvc);
144     add(repeatTransField);
145     add(nalsForFrame);
146     add(plrLabel);
147     add(plrNone);
148     add(plrOne);
149     add(plrTwo);
150     add(plrFive);
151     add(plrTen);
152     add(payloadTypeLabel);
```

```
153
154      add(payloadTypeSNU);
155      add(payloadTypeSTAP_A);
156      add(payloadTypeFU_A);
157      add(payloadTypeFU_A_Bytes);
158      add(numberOfAggrOrFragmLabel);
159      add(numberOfAggrOrFragmField);
160      add(repeatTransLabel);
161      add(repeatTransField);
162      add(serverIpLabel);
163      add(serverIpField);
164      add(blank);
165      add(ipPacketLabel);
166      add(ipPacketField);
167      add(button);
168
169       /*declaration of the button handler*/
170       ButtonHandler handler2=new ButtonHandler();
171       /*add action listener(on click)*/
172       button.addActionListener(handler2);
173
174      }
175
176 /*--Creation of an internal class to handle events--*/
177     private class ButtonHandler
178                                 implements ActionListener
179     {
180       private int packetSize=1024;
181       private String host="192.168.1.2";
182       private int repeatTrans=1;
183       private int avcMvcChoice=1;
184       private String nalUs;
185       private String plr;
186
187       private String payloadType;
188       private int    numberOfnalUs;
189
190       private String path;
```

```
192       public void actionPerformed(ActionEvent event)
193       {
194 /*--------------Store  user input data--------------*/
195
196        if(ipPacketField.getText().isEmpty())
197        {
198          JOptionPane.showMessageDialog(null,
199                     "IP Packet Field is empty.A default"
200 + " ip packet value is going to be used(1024 bytes)");
201        }
202        else
203        {
204          packetSize=Integer.parseInt
205                               (ipPacketField.getText());
206        }
207
208        if(serverIpField.getText().isEmpty())
209        {
210         JOptionPane.showMessageDialog
211           (null,"Server IP Field is empty.A default "
212        + "ip value is going to be used(192.168.1.2)");
213        }
214        else
215        {
216         host=serverIpField.getText();
217        }
218        if(repeatTransField.getText().isEmpty())
219        {
220         JOptionPane.showMessageDialog(null,
221                       "The Number Of Re-Transmission "
222                     + "Field is empty.A default value"
223                      + " is going to be used(1)");
224        }
225        else
226        {
227          repeatTrans=Integer.parseInt
228                            (repeatTransField.getText());
229        }
```

```
230
231         if(avcOrMvc.isSelected()==true )
232         {
233           avcMvcChoice=0;
234         }
235         else
236         {
237          avcMvcChoice=1;
238         }
239
240 /*------------Create folder tree-------------*/
241         if(nalsForFrame.isSelected()==true )
242         {
243          nalUs=ipPacketField.getText()+"NALUSize";
244          numberOfnalUs=2;
245         }
246         else
247         {
248          nalUs="1NalUperFrame";
249          numberOfnalUs=1;
250         }
251
252         if(plrNone.isSelected())
253          plr="0%";
254         else if(plrOne.isSelected())
255          plr="1%";
256         else if(plrTwo.isSelected())
257          plr="2%";
258         else if(plrFive.isSelected())
259          plr="5%";
260         else if(plrTen.isSelected())
261          plr="10%";
262
263
264         File f = new File(videoField.getText()+
265                           "/"+nalUs+"/IpPacket"+
266                         packetSize+"BytesTest");
267         File f2;
```

```
268
269
270         f.mkdirs();
271
272         for (int i=1;i<=repeatTrans;i++)
273         {
274             //System.gc();
275          if(payloadTypeSTAP_A.isSelected())
276          {
277            path=f.getPath()+"/STAP-A"+"/"+
278                   numberOfAggrOrFragmField.getText()
279              +"/"+plr+"/"+i+"Test"+"/decodedVideos";
280            payloadType="1";
281          }
282          else if(payloadTypeFU_A.isSelected())
283          {
284            path=f.getPath()+"/FU-A"+"/"+
285                     numberOfAggrOrFragmField.getText()
286                 +"/"+plr+"/"+i+"Test"+"/decodedVideos";
287             payloadType="2";
288          }
289          else if(payloadTypeFU_A_Bytes.isSelected())
290          {
291            path=f.getPath()+"/FU-Abytes"+"/"+
292                      numberOfAggrOrFragmField.getText()
293                 +"/"+plr+"/"+i+"Test"+"/decodedVideos";
294             payloadType="2";
295          }
296          else if(payloadTypeSNU.isSelected())
297          {
298            path=f.getPath()+"/SNU"+"/"+plr+"/"+i+"Test"
299                                          +"/decodedVideos";
300
301            payloadType="0";
302          }
303          /*Create New Folder*/
304          f2=new File(path);
305
```

```
306         if(f2.exists()==false)
307         {
308           f2.mkdirs();
309           System.out.println("Directory Created");
310
311
312   /*-------------Start Transmission--------------*/
313           path=path.replaceAll("/decodedVideos", "/");
314
315           EstablishConnection establish=
316                           new EstablishConnection(host,
317                          packetSize,avcMvcChoice,path);
318           /*Establish Signalling Channel*/
319           establish.receiveParameterSet();
320           /*Establish UDP/IP Channels*/
321           establish.enstablishUDPConnection();
322
323           while(!establish.getEndFlag())
324           {
325            //wait until the end of the  transmission
326           }
327
328           /*Depacketize*/
329           ProcessFiles depDec =new ProcessFiles
330                       (path,payloadType,numberOfnalUs)
331           depDec.depacketize();
332           /*Decode*/
333           depDec.decode();
334
335              try
336              {
337               Thread.sleep(2000);
338              }
339               catch (InterruptedException ex)
340              {
341             JOptionPane.showMessageDialog(GUI.this,
342                                 "InterruptedException");
              }
```

```
344             }
345           else
346           {
347               JOptionPane.showMessageDialog(null,
348                           "Directory allready exist.Store"
349                          + "(cut) these values to a safe "
350                          + "place and then try again");
351               break;
352           }
353         }
354
355          JOptionPane.showMessageDialog(null,
356                                  "Transmission finished");
357          System.exit(1);
358       }
359    }
360 }
```

```
1  import java.net.Socket;
2
3  import java.io.InputStream;
4  import java.io.OutputStream;
5  import java.io.BufferedOutputStream;
6  import java.io.BufferedWriter;
7
8  import java.io.File;
9  import java.io.FileOutputStream;
10 import java.io.FileWriter;
11
12 import java.io.IOException;
13 import java.net.UnknownHostException;
14
15 import java.util.concurrent.Executors;
16 import java.util.concurrent.ExecutorService;
17
18 import javax.swing.JOptionPane;
19
20 public class EstablishConnection
21 {
22   private   int     avcMvcChoice;
23   private   String  host;
24   private   int     packetSize;
25   private   double  sequenceParameterSetBytes=0;
26   private   boolean endFlag;
27
28   protected FileOutputStream leftFos;
29   protected BufferedOutputStream leftBos;
30   protected FileOutputStream rightFos;
31   protected BufferedOutputStream rightBos;
32   protected FileOutputStream nalUBytesFos;
33   protected BufferedOutputStream nalUBytesBos;
34
35   protected BufferedWriter printStats;
36
37   QuoteClientThread baseView;
38   QuoteClientThread non_baseView;
```

```
39
40  public EstablishConnection(String host,
41          int packetSize,int avcMvcChoice
42                              ,String path)
43 {
44  this.host=host;
45  this.packetSize=packetSize;
46  this.endFlag=false;
47  this.avcMvcChoice=avcMvcChoice;
48
49  try
50  {
51    leftFos=new FileOutputStream(path+"LeftView.264");
52    leftBos=new BufferedOutputStream(leftFos);
53    nalUBytesFos = new FileOutputStream(path+
54                                        "NalUBytes.txt");
55    nalUBytesBos =
56              new BufferedOutputStream(nalUBytesFos);
57
58    printStats=new BufferedWriter(new FileWriter
59                            (path+"StreamerInfo.txt"));
60
61    /*If RightFile already exists -> erase it*/
62    File  right=new File(path+"RightView.264");
63    right.delete();
64
65    if(avcMvcChoice==0)
66    {//MVC
67      rightFos=new FileOutputStream(path+
68                                        "RightView.264");
69      rightBos=new BufferedOutputStream(rightFos);
70    }
71   }
72   catch (IOException ex)
73   {
74     JOptionPane.showMessageDialog(null,
75                           "File Not Found","ERROR",0);
76     System.exit(1);
```

```
77      }
78  }
79
80  protected void setTrasmissionType(Socket sock)
81  {
82    try
83    {
84      System.out.println("Sending Transmission "
85                              + "Information To Server");
86      byte [] flag  = new byte [1];
87      if(avcMvcChoice==0)
88      flag[0]=0;
89      else if(avcMvcChoice==1)
90      flag[0]=1;
91      OutputStream os = sock.getOutputStream();
92      os.write(flag,0,1);
93      os.flush();
94
95    } catch (IOException ex)
96    {
97       JOptionPane.showMessageDialog(null,
98                            "IOException-1","ERROR",0);
99       System.exit(1);
100   }
101
102 }
103
104 protected void receiveParameterSet()
105 {
106    int filesize=99999999;//temporary hardcoded
107    int bytesRead;
108    int current = 0;
109    int SpsLength = 0;
110
111    Socket sock;
112    try
113    {
114     sock = new Socket(host, 2050);
```

```
115     /*send transmission information to server*/
116     setTrasmissionType(sock);
117     sock = new Socket(host, 2050);
118     System.out.println("Connecting through Tcp...");
119
120     byte [] mybytearray  = new byte [filesize];
121     InputStream is = sock.getInputStream();
122
123     bytesRead = is.read(mybytearray,0,
124                                 mybytearray.length);
125     current = bytesRead;
126
127     do
128     {
129      bytesRead = is.read(mybytearray,current,
130                     (mybytearray.length-current));
131        if(bytesRead >= 0)
132        current += bytesRead;
133      }
134      while(bytesRead > -1);
135
136      //Calculate SPS length
137      for(int i=0;i<current;i++)
138       {
139        if(mybytearray[i]==22 && mybytearray[i+1]==22)
140        {
141           SpsLength=i;
142        }
143       }
144      /*Store length of SPS*/
145      sequenceParameterSetBytes=current;
146      /*Write SPS data to file*/
147      leftBos.write(mybytearray, 0 , SpsLength);
148      leftBos.flush();
149      System.out.println("Parameter Set Received");
150      /*Write NALU's length to file*/
151      nalUBytesBos.write(mybytearray,
152                             (SpsLength+2),current);
```

```
153
154         nalUBytesBos.flush();
155         System.out.println("NalUBytes Received");
156         sock.close();
157         is.close();
158         nalUBytesFos.close();
159         nalUBytesBos.close();
160
161       } catch (UnknownHostException ex)
162       {
163           JOptionPane.showMessageDialog(null,
164                           "Unknown Host","ERROR",0);
165           System.exit(1);
166       } catch (IOException ex)
167       {
168           JOptionPane.showMessageDialog(null,
169                           "IOException-2","ERROR",0);
170           System.exit(1);
171       }
172   }
173
174   protected void enstablishUDPConnection()
175   {
176     if(avcMvcChoice==0)
177     {//Start 2 Threads for MVC transmission
178       baseView=new QuoteClientThread(host,packetSize,
179                   leftFos,leftBos,4445,"Base View    ");
180       non_baseView=new QuoteClientThread(host,packetSize
181                 ,rightFos,rightBos,4446,"Non-Base View");
182
183       ExecutorService threadExecutor=
184                           Executors.newFixedThreadPool(2);
185       //Start Threads
186       threadExecutor.execute(baseView);
187       threadExecutor.execute(non_baseView);
188
189       threadExecutor.shutdown();
190
```

```
192     while(!threadExecutor.isTerminated())
193      {
194      /*Wait until threads are terminated*/
195      }
196
197
198   }
199   else
200   {//Start 1 Thread for AVC  transmission
201     baseView=new QuoteClientThread(host,packetSize,
202                 leftFos,leftBos, 4445,"Base View    ");
203
204     ExecutorService threadExecutor=
205                         Executors.newFixedThreadPool(1);
206     threadExecutor.execute(baseView);
207
208     threadExecutor.shutdown();
209
210     while(!threadExecutor.isTerminated())
211      {
212        /*Wait until threads are terminated*/
213      }
214    }
215
216     /*Print connection statistics*/
217     printStatistics();
218
219     endFlag=true;
220
221 }
222
223 protected void printStatistics()
224 {
225  try
226  {
227    String tempStats="";
228
229
```

```
230     tempStats="-----------------------------------"
231                               + "-----------------";
232     System.out.println(tempStats);
233      /*Write Statistics to file*/
234     printStats.write(tempStats);
235     printStats.newLine();
236
237     tempStats=" SPS Bytes="+sequenceParameterSetBytes;
238     System.out.println(tempStats);
239     printStats.write(tempStats);
240     printStats.newLine();
241
242     tempStats=" Left View Lost Bytes="+
243                                 baseView.getBytesLost();
244     System.out.println(tempStats);
245     printStats.write(tempStats);
246     printStats.newLine();
247
248     tempStats=" Left View Writen Bytes="+
249                               baseView.getBytesWriten();
250     System.out.println(tempStats);
251     printStats.write(tempStats);
252     printStats.newLine();
253
254     tempStats=" Packet Loss Rate For Left View="+
255     ((baseView.getBytesLost()/
256                   baseView.getBytesWriten())*100)+"%";
257
258     System.out.println(tempStats);
259     printStats.write(tempStats);
260     printStats.newLine();
261
262     if(avcMvcChoice==0)//MVC
263       {
264        tempStats="-----------------------------------"
265                                 + "-------------------";
266        System.out.println(tempStats);
267
```

```
268         printStats.write(tempStats);
269         printStats.newLine();
270 
271         tempStats=" Right View Lost Bytes="+
272                             non_baseView.getBytesLost();
273         System.out.println(tempStats);
274         printStats.write(tempStats);
275         printStats.newLine();
276 
277         tempStats=" Right View Writen Bytes="+
278                           non_baseView.getBytesWriten();
279         System.out.println(tempStats);
280         printStats.write(tempStats);
281         printStats.newLine();
282 
283         tempStats=" Packet Loss Rate For Left View="+
284         ((non_baseView.getBytesLost()/
285                 non_baseView.getBytesWriten())*100)+"%";
286         System.out.println(tempStats);
287         printStats.write(tempStats);
288         printStats.newLine();
289       }
290         tempStats="-----------------------------------"
291                                   + "-------------------";
292         System.out.println(tempStats);
293         printStats.write(tempStats);
294         printStats.newLine();
295 
296 
297         tempStats=" Base View Transmission Time="
298                                 +((baseView.getEndTime()-
299             baseView.getStartTime())/60000)+" minutes";
300         System.out.println(tempStats);
301         printStats.write(tempStats);
302         printStats.newLine();
303 
304         if(avcMvcChoice==0)//MVC
305          {
```

```
306         tempStats=" Non Base View Transmission Time="
307                     +((non_baseView.getEndTime()-
308                     non_baseView.getStartTime())/60000)
309                                             +" minutes";
310         System.out.println(tempStats);
311         printStats.write(tempStats);
312         printStats.newLine();
313         }
314
315         tempStats="-----------------------------------"
316                                 + "--------------------";
317         System.out.println(tempStats);
318         printStats.write(tempStats);
319         printStats.newLine();
320
321
322         printStats.close();
323         } catch (IOException ex)
324         {
325               JOptionPane.showMessageDialog(null,
326                                       "IOException-4 in "+
327                     Thread.currentThread().getName(),
328                                             "ERROR",0);
329               System.exit(1);
330         }
331
332
333 }
334
335  public boolean getEndFlag()
336  {
337    return endFlag;
338  }
339 }
```

```java
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.net.UnknownHostException;
import java.io.IOException;

import javax.swing.JOptionPane;

public class QuoteClientThread implements Runnable
{
   private boolean moreQuotes;
   private int     packetSize;
   private int     connectionCheck=0;
   private int     endFlag=0;

   private double bytesLost=0;
   private double bytesWriten=0;
   private double start;
   private double end;

   private String threadName;
   byte [] clientBuf;
   byte [] buf;

   InetAddress address;
   DatagramPacket packet1;
   DatagramPacket packet2;

   protected FileOutputStream fos = null;
   protected BufferedOutputStream bos =null;
   protected DatagramSocket socket =null;

```

```
39
40  QuoteClientThread(String host,int packetSize,
41                    FileOutputStream fos ,
42                    BufferedOutputStream bos,
43                    int port,String threadName)
44  {
45     this.fos=fos;
46     this.bos=bos;
47     this.packetSize=packetSize;
48     moreQuotes=true;
49     this.start = System.currentTimeMillis();
50
51     this.threadName=threadName;
52
53     clientBuf=new byte[16];
54     buf = new byte[packetSize];
55     try
56     {
57      try
58      {
59       address = InetAddress.getByName(host);
60       socket = new DatagramSocket();
61
62       packet1 = new DatagramPacket(clientBuf,
63               clientBuf.length,address,port);
64       packet2 = new DatagramPacket(buf,
65                     buf.length,address,port);
66      }
67      catch (UnknownHostException ex)
68      {
69       JOptionPane.showMessageDialog(null,
70                     "UnknownHostException"+
71           Thread.currentThread().getName(),"ERROR",0);
72       System.exit(1);
73      }
74      }
75     catch (SocketException ex)
76     {
```

```
77      JOptionPane.showMessageDialog(null,
78                                          "SocketException"+
79          Thread.currentThread().getName(),"ERROR",0);
80      System.exit(1);
81     }
82
83  }
84
85
86 public void run()
87 {
88    Thread.currentThread().setName(threadName);
89
90    //Start file transmission throw UDP/IP
91    while(moreQuotes)
92    {
93        SocketConnection(); //receive one packet
94    }
95
96    end=System.currentTimeMillis();
97  }
98
99
100 protected void SocketConnection()
101 {
102    moreQuotes=true;
103    try
104     {
105       /*Send request*/
106       socket.send(packet1);
107
108       /*get packet*/
109       try
110       {//Wait 70ms to receice packet else->packet lost
111
112         socket.setSoTimeout(70);
113         socket.receive(packet2);
114
```

```
115         connectionCheck=0;//connected
116         System.out.println("A Packet Of  "+
117                         Thread.currentThread().getName()
118                             +"  received through Udp.");
119       }
120       catch(SocketTimeoutException e)
121       {
122         System.err.println("A Packet Of  "+
123                         Thread.currentThread().getName()+
124     "  lost     through Udp. Bytes Lost: "+buf.length);
125         bytesLost+=buf.length;
126         for(int i=0;i<packetSize;i++)
127         {/*Since the packet is lost
128          * -> replace the lost bytes  */
129            buf[i]=(byte)0x02;
130         }
131         connectionCheck++;
132
133         /*if 10 consecutive packets lost
134          * -> channel problem*/
135         if(connectionCheck>9)
136         {
137           System.err.println("Connection Lost in "+
138                     Thread.currentThread().getName());
139
140           bytesLost-=(buf.length*10);
141           for(int i=0;i<clientBuf.length;i++)
142           {
143             clientBuf[i]=1;
144           }
145           packet1.setData(clientBuf);
146           /*Request:stop connection*/
147           socket.send(packet1);
148           for(int i=0;i<packetSize;i++)
149           {
150               buf[i]=0;
151           }
152         }
```

```
153         }
154
155         /*If packet contains no data
156                                 * -> End of Transmission*/
157
158         endFlag=0;
159         for(int i=0;i<packetSize;i++)
160         {
161           if(buf[i]==0)
162           endFlag++;
163         }
164
165         if(endFlag==packetSize)
166         {
167           System.out.println("End of Transmission in "+
168                       Thread.currentThread().getName());
169           socket.close();
170           socket.disconnect();
171           moreQuotes=false;
172           fos.close();
173           bos.close();
174         }
175         else
176         {
177           bos.write(packet2.getData(),0,
178                     packet2.getLength());
179           bos.flush();
180           bytesWriten+=packetSize;
181       }
182       } catch (IOException ex)
183       {
184          JOptionPane.showMessageDialog(null,
185                                        "IOException-3"+
186                     Thread.currentThread().getName(),
187                                        "ERROR",0);
188          System.exit(1);
189       }
190
```

```
192
193   public double getBytesLost()
194   {
195     return bytesLost;
196   }
197
198   public double getBytesWriten()
199   {
200     return bytesWriten;
201   }
202
203   public double getStartTime()
204   {
205     return start;
206   }
207
208   public double getEndTime()
209   {
210     return end;
211   }
212
213 }
```

```
1  import java.io.File;
2
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.io.IOException;
6
7  import java.io.InputStream;
8  import java.io.OutputStream;
9  import javax.swing.JOptionPane;
10
11 public class ProcessFiles
12 {
13   private String path;
14   private String payloadType;
15   private int nalUs;
16   private boolean stopFlag;
17
18   private Process process;
19   private ProcessBuilder pb;
20
21   private File file;
22
23   private BufferedReader brStdInput;
24
25   private OutputStream stdin;
26   private InputStream stderr;
27   private InputStream stdout;
28
29   private String line;
30
31   public ProcessFiles(String path,String payloadType
32                                                 ,int nalUs)
33   {
34     stopFlag=false;
35     this.path=path;
36     this.payloadType=payloadType;
37     this.nalUs=nalUs;
38   }
```

```
39
40  public void depacketize()
41  {
42
43   System.out.println("Starting Depacketization "
44                                       + "Process...");
45
46   /*Delete previous depacketized file*/
47   file=new File("Depacketizer/my.264");
48
49   if(file.exists())
50   {
51    file.delete();
52    System.err.println("my.264 deleted");
53   }
54   try
55   {
56   pb = new ProcessBuilder("Depacketizer/"
57             + "Depacketizer.exe",path+"LeftView.264",
58            path+"RightView.264",path+"NalUBytes.txt",
59                      path+"\\decodedVideos\\my.264");
60    process = pb.start();
61
62    stdin  = process.getOutputStream();
63    stderr = process.getErrorStream();
64    stdout = process.getInputStream();
65
66
67    /*If unrecognised payload type ->
68     * Send to Depacketizer the payloadType*/
69
70    line = "1"+"\n";
71    if(payloadType.equals("2"))
72     {
73      stdin.write(line.getBytes());
74      stdin.flush();
75      }
76     stdin.close();
```

```
77
78
79      // show stdout output
80     brStdInput =new BufferedReader
81                     (new InputStreamReader (stdout));
82     while ((line = brStdInput.readLine ()) != null)
83     {
84       System.out.println (line);
85     }
86     brStdInput.close();
87
88       // show stderr output
89       brStdInput =new BufferedReader
90                       (new InputStreamReader (stderr));
91       while ((line = brStdInput.readLine ()) != null)
92       {
93         System.out.println (line);
94       }
95       brStdInput.close();
96
97   process.destroy();
98
99   }
100  catch (IOException ex)
101  {
102    JOptionPane.showMessageDialog(null,"IOException"+
103         Thread.currentThread().getName(),"ERROR",0);
104    System.exit(1);
105  }
106
107 }
108
109   public void decode()
110  {
111   System.out.println("Starting Decoding Process...");
112
113     if(stopFlag==false)
114      {
```

```
115         try
116         {
117           System.out.println(path+
118                              "\\decodedVideos\\my.264");
119            pb = new ProcessBuilder
120                              ("Decoder/MVCDecoder.exe "
121                              ,"-i",path+
122                     "/decodedVideos/my.264","-recoyuv");
123
124           process = pb.start();
125
126           BufferedReader stdInput =
127               new BufferedReader(new InputStreamReader
128                              (process.getInputStream()));
129
130           BufferedReader stdError = new BufferedReader
131                                    (new InputStreamReader
132                              (process.getErrorStream()));
133
134            //Standard output of the command;
135            line = null;
136            while ((line = stdInput.readLine()) != null)
137            {
138               System.out.println(line);
139            }
140            //Standard eror of the command;
141            while ((line = stdError.readLine()) != null)
142            {
143              System.err.println(line);
144              stopFlag=true;
145            }
146
147            stdInput.close();
148            stdError.close();
149            process.destroy();
150
151            /*Mode videos at the correct place*/
152            moveFiles();
```

```
153
154         } catch (IOException ex)
155         {
156             System.err.println("IOExeption");
157         }
158
159     }
160 }
161
162   public void moveFiles()
163   {
164      try
165      {
166       try
167       {
168         if(payloadType.equals("0") && nalUs==1)
169          {
170             System.out.println("Moving Files...");
171
172             process = Runtime.getRuntime().exec
173                                    ("cmd /C MOVE "
174                 + "Base_View_Frame_Information.txt "
175                                +path+"\\decodedVideos");
176             process = Runtime.getRuntime().exec
177                                        ("cmd /C MOVE"
178              + " Non_Base_View_Frame_Information.txt "
179                                +path+"\\decodedVideos");
180          }
181         else
182         {
183           System.out.println("Deleting Files...");
184
185           process = Runtime.getRuntime().exec
186                                        ("cmd /C DEL "
187               + "Base_View_Frame_Information.txt ");
188           process = Runtime.getRuntime().exec
189                                        ("cmd /C DEL "
190            + "Non_Base_View_Frame_Information.txt");
```

```
192
193             System.out.println("Moving Videos");
194
195             process = Runtime.getRuntime().exec
196                             ("cmd /C MOVE OutLeft.yuv "
197                                +path+"\\decodedVideos");
198             process = Runtime.getRuntime().exec
199                             ("cmd /C MOVE OutRight.yuv "
200                                +path+"\\decodedVideos");
201
202             process.waitFor();
203             process.destroy();
204          }
205          catch (InterruptedException ex)
206          {
207            JOptionPane.showMessageDialog(null,
208                                "InterruptedException"+
209                        Thread.currentThread().getName(),
210                                              "ERROR",0);
211           System.exit(1);
212          }
213          }
214        catch (IOException ex)
215        {
216             JOptionPane.showMessageDialog(null,
217                                          "IOException"+
218                        Thread.currentThread().getName(),
219                                              "ERROR",0);
220        System.exit(1);
221       }
222
223     }
224
225 }
```

- E. Nokia Encoder Parameters

1 NALU ανά πλαίσιο

MVCEncoder.exe -inyuv leftview.yuv -in2 rightview.yuv -out out3d.264 -rframes 2 -frames 1000 -q 24 -qintra 24 -idrFreq 15 -intraFreq 5 -skip 0 -width 1280 -height 1024

512 bytes ανά NALU

MVCEncoder.exe -inyuv leftview.yuv -in2 rightview.yuv -out out3d.264 -rframes 2 -frames 1000 -q 24 -qintra 24 -idrFreq 15 -intraFreq 5 -skip 0 -width 1280 -height 1024 -targetbytesperslice 512 -maxbytesperslice 512

1024 bytes ανά NALU

MVCEncoder.exe -inyuv leftview.yuv -in2 rightview.yuv -out out3d.264 -rframes 2 -frames 1000 -q 24 -qintra 24 -idrFreq 15 -intraFreq 5 -skip 0 -width 1280 -height 1024 -targetbytesperslice 1024 -maxbytesperslice 1024

- Z. Dummynet Configuration

1% packet loss rate

ipfw –q –flush
ipfw –q pipe -flush

ipfw add pipe 1 ip from "Server Ip" to any out
ipfw add pipe 2 ip from any to "Server Ip"
ipfw pipe 1 config bw 100Mbit/s queue 100 plr 0.01 mask dst-ip 0x000000ff
ipfw pipe 2 config bw 100Mbit/s queue 100 mask dst-ip 0x000000ff

2% packet loss rate

ipfw –q –flush
ipfw –q pipe -flush

ipfw add pipe 1 ip from "Server Ip" to any out
ipfw add pipe 2 ip from any to "Server Ip"
ipfw pipe 1 config bw 100Mbit/s queue 100 plr 0.02 mask dst-ip 0x000000ff
ipfw pipe 2 config bw 100Mbit/s queue 100 mask dst-ip 0x000000ff

5% packet loss rate

ipfw –q –flush
ipfw –q pipe -flush

ipfw add pipe 1 ip from "Server Ip" to any out
ipfw add pipe 2 ip from any to "Server Ip"
ipfw pipe 1 config bw 100Mbit/s queue 100 plr 0.05 mask dst-ip 0x000000ff
ipfw pipe 2 config bw 100Mbit/s queue 100 mask dst-ip 0x000000ff

Παράρτημα Γ

Εργασίες οι οποίες έχουν υποβληθεί σε συνέδρια

- A. PV2012

Το PV2012 είναι το 19ος διεθνές Packet Video Workshop και θα πραγματοποιηθεί τον Μάιο 10-11,2012 στο Μόναχο, Γερμανία. Το workshop είναι αφοσιωμένο στην παρουσίαση τεχνολογικών εξελίξεων και καινοτομιών στην μετάδοση βίντεο και πολυμέσων σε ασύρματα και ενσύρματα δίκτυα.

On the performance of H.264/MVC over lossy IP-based networks

Athanasios Kordelas
Dept. of Telecommunication System and Networks
Cones Research Group
Nafpaktos 30300, Greece
athankord@tesyd.teimes.gr

Tasos Dagiuklas
Dept. of Telecommunication System and Networks
Cones Research Group
Nafpaktos 30300, Greece
ntan@tesyd.teimes.gr

Ilias Politis
Dept. of Telecommunication System and Networks
Cones Research Group
Nafpaktos 30300, Greece
ilpolitis@gmail.com

***Abstract*— this paper studies the performance of different packetization schemes for the emerging H.264/Multi-view Video Coding standard, in terms of overhead, number of decoded frames, and perceived PSNR under different network conditions (MTU size, packet loss). The experimentation test-bed platform designed for the purposes of this study, utilizes the Multi Session Transmission approach for different payload structures (single NAL Unit, aggregation packet and Fragmentation Unit) and a number of video packetization options, including one NAL Unit or several NAL Units per frame. Extensive test-bed experiments indicate that the fragmentation of frames in more than one NAL Units results in significantly higher perceived video quality in terms of PSNR, than the fragmentation of the NAL Unit in the RTP layer.**

Keywords- H.264/MVC, RTP, video quality evaluation

I. INTRODUCTION

The delivery of 3D media to individual users remains a highly challenging problem due to the large amount of data involved, diverse network characteristics, user terminal requirements, as well as, user's context such as their preferences and location . As the number of visual views increases, current systems will struggle to meet the demanding requirements in terms of delivery of consistent video quality to fixed and mobile users.

Several problems occur during the transmission of H.264 3D video sequences. The most important one is that the non – base view video quality after the transmission in an IP-based network may lead to reduced perceived video quality compared to base view due to inter-view prediction [1]. Recently, 3D video transmission over IP networks has received particular attention. In particular, [2] studies the quality reduction of multi-view coded (MVC) [3] video due to wireless losses, however the impact of these loses on the base and non-base views is not considered. Moreover, in [5] the performance of different packetization modes of the H.264/MVC standard under different network conditions is presented.

The aim of this paper is to assess the performance of 3D video streaming over IP based networks using various video packetization modes according to the H.264/MVC standard. As opposed to previous works, an error concealment scheme for recovering lost Network Abstraction Layer unit headers is implemented. The performance evaluation considers various parameters including the overhead caused by different packetization schemes, the maximum transmission unit size, the number of decoded frames and the resulted video quality in terms of Peak Signal to Noise Ratio (PSNR) for both base and non-base views, under varying network conditions and packetization modes. The MVC video transmission performance analysis is based on experiments conducted on a test-bed platform that comprises of an RTP packetizer and de-packetizer, a UDP/IP streamer and client, implemented specifically for this study.

The rest of the paper is organized as follows. Section II includes an overview of the H.264/MVC standard, while in Section III the proposed error resiliency scheme is described. The experimentation setup is presented in Section IV and Section V includes a discussion on the produced results. Finally, Section VI concludes the paper.

II. H.264/MVC OVERVIEW

H.264/MVC standard is an extension of the H.264/Advance Video Coding (AVC) and Scalable Video Coding standards [3], [4]. According to these standards the Video Coding Layer (VLC) produces a coded representation of the video, while the Network Abstraction Layer Unit (NALU) encodes the video data in a prepared way for transmission. Inter-view prediction introduced by MVC for higher compression efficiency causes dependency between the non-base views and the base view. For the encapsulation of the MVC video data in NALUs, new header extensions are introduced by H.264/MVC standard. In both views an additional header named Prefix NAL Unit (type 14) must exist while in the non-base view the H.264/AVC header is extended by three additional bytes and is named coded slice of MVC extension (type 20). The existence of the original H.264/AVC NALU header (1 octet) at the base view, allows the H.264/AVC decoder to decode a 2D representation from the H.264/MVC sequence.

There are three transmission modes for a H.264/MVC encoded video, namely, Single-Session Transmission (SST), Multi-Session Transmission (MST) and Media-Aware Network

Element (MANE) based transmission [9]. In the case of SST mode, all the MVC information is transmitted over a single RTP session, utilizing one transport address (unicast) and transmitting the base view with a number of different non-base views. Moreover, MST mode establishes more than one RTP sessions. In this case the number of the used transport addresses is equal to the number of RTP sessions (multicast). In addition, each RTP session in MST may carry only the base view, or a combination of the base view with a number of non-base views, or only non-base views. Figure 1 illustrates the MST mode, implemented within the context of this study. Furthermore, in MANE mode, although the server utilizes MST transmission, the RTP sessions are collected from an intermediate entity (MANE). According to MANE, RTP packets are de-packetized and an Adaptation Decision Taking Engine (ADTE) is utilized in order to matched client's preferences and needs and the transmission parameters. Additionally, the packets are further aggregated for SST transmission to the client through a single transport address using a single RTP session.

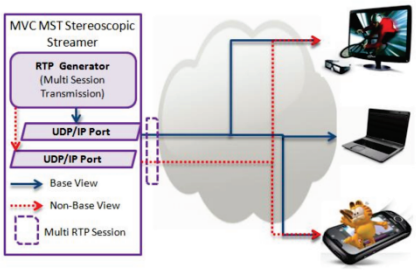


Figure 1. MVC MST Transmission Mode

For the encapsulation of NALUs in RTP packets three NALU payload structures are specified in the RTP specification of MVC [6]. The first one is the "Single NAL unit" (SNU) according to which each RTP packet encapsulates a whole NALU. The second one is the "Aggregation Packets", which specifies that multiple NALUs are encapsulated in one RTP packet. This payload structure includes five versions, STAP-A, STAP-B, MTAP-16, MTAP-24 and NI-MTAP. The latter is known as "Fragmentation Unit (FU)" and allows the fragmentation of one NALU into smaller RTP packets. In turn, FU includes two versions, FU-A and FU-B.

III. Error Resilience in RTP Layer

In order to deal with the loss of important headers, a RTP - aware error recovery algorithm based on FU payload structure is implemented at the RTP de-packetizer. This scheme is able to reconstruct in the application layer all the lost H.264/AVC NALU headers and allows the decoder to recognize and decode all the video frames (NALUs) of both views. Therefore, the proposed scheme significantly improves the perceived video quality of the decoded video.

In particular, frame losses occur more frequently when an entire encoded video frame is encapsulated in a single NALU. In this case, the H.264/AVC NALU header is encapsulated into a single IP datagram. Whenever such an IP datagram is lost, causes the decoder to discard the entire NALU (frame) as it is unable to recognize the NALU type.

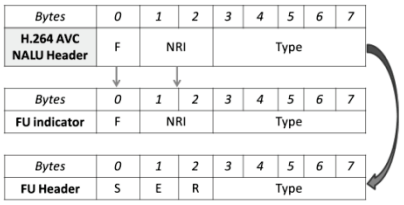


Figure 2. Headers of "Fragmentation Unit" payload structure

According to the RTP standard a FU always encapsulates part of a NALU using apart from the RTP header two additional headers, the "FU Indicator" and the "FU Header" (each one octet long), as shown in Figure 2. The first two fields (3 bits) of the "FU Indicator" denoted by *F* and *NRI* obtain their values from the corresponding fields of the H.264/AVC NALU header, while the last field, denoted by *Type* (5 bits) always has as value the type of the packet, which in the context of this study is set to *28* (FU-A). Moreover, the field *Type* of the "FU Header" obtains its value from the corresponding field of the H.264/AVC NALU header. The fields *S* and *E* indicate the first and the last fragment of a NALU, respectively. Lastly, *R* is a reserved field intended for future extensions and its values is always set to zero. Following the fragmentation of a NAL unit, the H.264/AVC NALU header is erased during packetization and re-constructed by the two FU headers during de-packetization.

Several approaches for concealing errors due to lost frames have been proposed [12]. It is possible to recover lost headers and estimate the expected NALU type based on information already known to the decoder, such as the GOP size. In particular, as long as, the frames of the same type are encapsulated with the same NALU headers, the de-packetizer is able to partially re-construct missing NALU headers based on default headers for each frame type.

A. Proposed error resilience scheme

The proposed error resilience scheme is based on the fact that the necessary fields of the H.264/AVC header are present in all FUs of a NALU. The proposed algorithm is illustrated in Figure 3 and comprises the following steps:

1. The de-packetizer discovers that the first fragment of the NALU is missing.
2. Searches in the following fragments of the same NALU to discover one received with no errors.
3. Upon discovering such a fragment, copies the needed values of the bit-fields in the "FU Indicator" and the "Fu Header" and reconstructs the necessary for the decoding H.264/AVC NALU header.
4. The de-packetizer rejects the other RTP headers and stores the NALU payload.

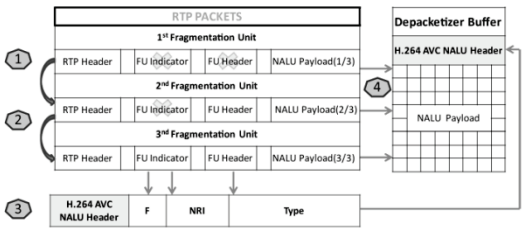


Figure 3. Proposed error-resilience algorithm

Since vital information for the decoding process, including the Sequence Parameter Set [13], is contained in the first IP datagram, it is important to ensure the reliable transmission of this information by utilizing TCP/IP transmission.

It is important to be mentioned that the RTP standard [8] specifies that: *"If a fragmentation unit is lost, the receiver SHOULD discard all following fragmentation units in transmission order corresponding to the same fragmented NAL unit"*. Since the specific proposition is not mandatory, the proposed error resilient scheme does not consider as lost all the fragmented units that follow a lost one. Thus, missing header information of the lost FUs can be recovered and the frame that contains the lost fragmented units can be successfully decoded. The missing pixel values included in lost fragmentation units are replaced by default values that the decoder interprets as distortion of the original frame and conceals the errors.

IV. Experimentation Setup

A. Test-bed platform

MVC video transmission over IP is achieved by using several tools. Nokia's MVC Encoder/Decoder for the encoding and decoding of different view video sequences [17], while Dummynet is utilized to emulated packet losses [15]. Furthermore, a number of tools have been implemented based on the MVC, RTP and UDP standards:

i. RTP packetizer – able to create RTP packets from a coded MVC sequence regardless of the fact that NALUs may contain an entire frame or a part of it. It is also able to create RTP packets using all the payload structures as defined by the standard (SNU, STAP-A, FU). In the STAP-A mode, the number of the aggregated NALUs per RTP packet can be defined for both video packetization options. In the case of multiple NALUs per frame, RTP packets can also be created automatically according to the number of the fragments per frame. Concerning the FU mode, the number of fragments per frame can be set to a specified value, or each FU's length (in bytes) is set in order to create packets equal to the length of the desirable MTU size. For both aggregation packets and FUs, the latter more intelligent choice is adopted in the rest of the paper.

ii. RTP de-packetizer – able to de-packetize RTP packets and reconstruct a MVC sequence. It is tolerant to bit errors and it is able to recognize the payload type used during the packetization process, as well as, the coding parameters used (one or multiple NALUs per frame). The de-packetizer incorporates the proposed error resilience scheme.

iii. MVC streamer – that encapsulates RTP packets into IP datagrams and creates concurrent UDP/IP connections to the client for multicast transmission of both views.

iv. MVC client – through a Graphical User Interface (GUI) transmits a request (based on transmission information including the number of views, the payload type and the MTU size) to the video streamer over TCP/IP connection. The client establishes UDP/IP connections (one connection in the case of AVC) with the streamer in order to receive IP datagrams of both views.

The test-bed platform including all the tools and functionalities designed for the purposes of this study is shown in Figure 4.

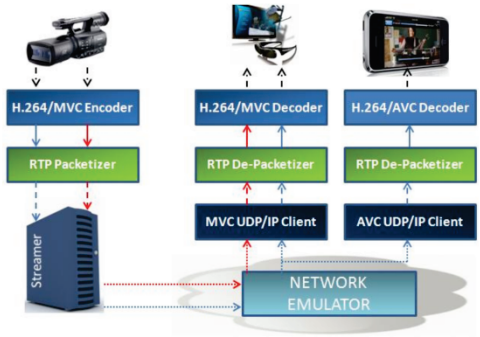


Figure 4. Experimental test-bed platform

B. Experimentation Parameters

Table 1 Coding parameters

Video Sequence	Flamenco	Objects
No of Frames	1000	624
Intra period	5 frames	5 frame
Frame rate	25 fps	25 fps
Resolution	640x480 pixels	640x480 pixels
Quantization parameter	24	24
No of Views	2	2

Table 2 Packetization parameters

Video packetization options	1 NALU / Frame		Multiple NALUs/Frame	
RTP packetization options	SNU	FU-A	SNU	STAP-A

Table 3 Networking parameters

MTU Size (Bytes)	1024	512
Packet loss rate	0% , 1% , 2% , 5%	
Error Model	Both Views	Base View

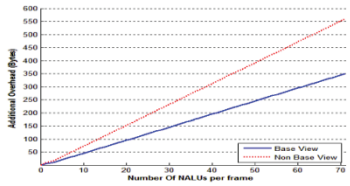


Figure 5. Overhead increase due frame encapsulation into multiple NALUs

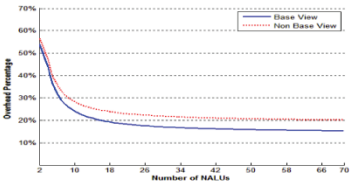


Figure 6. Overhead comparison with respect to SNU between STAP-A payload structure for base and non-base views

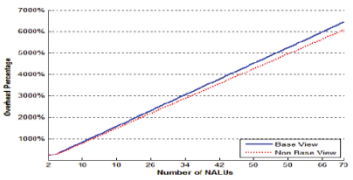


Figure 7. Overhead comparison with respect to FU-A payload structure for base and non-base views

The characteristics of the video sequences [18] and the MVC parameters used during the experiments are summarized in Table 1, RTP packetization parameters are included in Table 2 and Table 3 illustrates networking parameters studied during experiments. Each video transmission is repeated for 15 times to obtain the average PSNR values. STAP-A payload structure is not used in conjunction with 1 NALU per frame, since this would cause multiple frames to be encapsulated into a single RTP packet. Additionally, FU-A structure is not applied in the case of multiple NALUs per frame, since further fragmentation of NALUs into RTP packets is redundant.

V. EXPERIMENTAL RESULTS

A. Overhead

When each frame is encapsulated in more than one NALU, an additional overhead is added in the coded bit stream. Figure 5 illustrates the additional overhead of a coded frame as the number of NALUs increases. The aggregation of more than one NALU into one RTP packet results into less overhead.

Figure **6** illustrates the reduction of the overhead compared to the SNU mode as the number of the aggregated NALU's increases. The fragmentation of a NALU has the disadvantage of increasing overhead.

Figure **7** shows that as the number of the fragments of a NALU increases as the total overhead increases, as well. Studying all three figures, it can be proved that the frame fragmentation into multiple NALUs (either in SNU or STAP-A modes) results into much lower overhead, compaired to fragmantation at RTP layer.

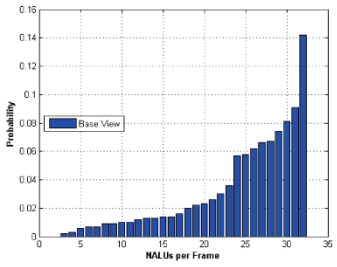


Figure 8. PMF of NALUs per frame in the base view

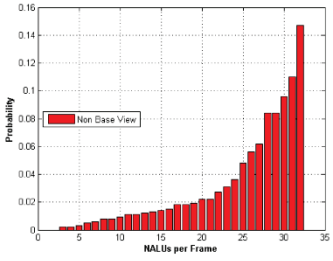


Figure 9. PMF of NALUs per frame in the non-base view

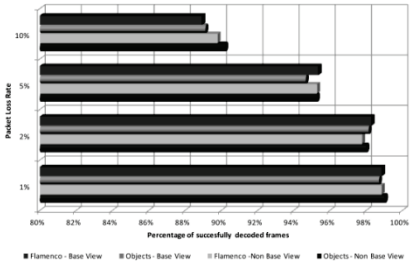


Figure 10. Percentage of decoded frames of base and non-base views in SNU mode under different packet loss rates ("flamenco" and "objects")

In addition, Figure 8 and Figure 9 show the probability mass function (pmf) of the NALUs per frame for the base and non-base views respectively. Evidently, the probability of having many NAL units per frame is significantly larger and this is due to the video content and the coding characteristics, as well as the size of the NALUs that is set to 1024 bytes.

B. Lost Frames

During video transmission over lossy networks a frame loss occurs when each RTP packet contains one NALU per frame and the de-packetization does not support error resilience. In the context of this study, this can only take place in SNU mode (with one NAL unit per frame). Figure 10 shows the number of frames that were decoded successfully after the transmission of two video sequences,

each one with two views, under different network conditions.

C. Lost Packets in both views

Figure 11 (a, b, c, d) and Figure 12 (a, b, c, d) illustrate PSNR of two multiview video sequences under different video and network conditions. During video transmission when packet loss occurs in both views, additional error propagation to the non- base view exists due to its dependency with the base view. It can be seen that for both video sequences, the perceived quality in terms of PSNR of the base view is significantly better than the quality achieved in the non-base view. This difference in the PSNR could only be eliminated if the non-base view was encoded independently, but with the cost of coding efficiency. Moreover, PSNR increases as MTU size increases, since larger MTUs eliminatre the effect of error bursts.

In particular, Figures 11a illustrates PSNR resulted by SNU mode and one NALU per frame paketization. It can be seen that one frame per RTP encapsulation produces the lowest video QoS, among all packetization schemes. In this case a lost packet causes an entire frame to be lost, increasing the received video distortion. The decoder implements frame copy concealment technique in the case of lost frames. Figure 11b shows PSNR in FU-A mode. In details, the fragmentation of a NALU in parallel with the proposed error-resiliency scheme results in increased PSNR, especially in the cases of *1%* and *2%* packet loss. In this case, all frames are succesfully decoded, thus the perceived video quality is significantly increased.

However, the experiments proved that the encapsulation of each frame into multiple NALUs resulted in higher video QoS than the previous two cases. The PSNR measured when SNU mode (1 NALU per RTP packet) and STAP-A mode (all NALUs per RTP packet) are applied, is illustrated in Figure 11c and 11d respectively. The fragmentation of a frame into multiple NALUs reduces the distortion effect due to packet loss in the received video. When frame fragmentation into multiple NALUs is applied then the MTU size has only a limited impact on the measured PSNR.

Finally, by comparing the PSNR measurements under all scenarios for the “flamenco” and “objects” sequences, it can be seen that the flamenco sequence achieves better PSNR due to the video contex.

D. Lost Packets in base view - Error Propagation

Figures 13 (a, b, c, d) and 14 (a, b, c, d) illustrate PSNR when errors occur only in the base view. Similar to previous experiments, the video QoS increases with MTU size. Comparing this set of experiments to the previous experiments it becomes clear the view dependency between base and non-base view. It is evident that as losses occur only to base view, the PSNR of the non-base view is less affected than in the previous cases (Figure 11, Figure 12).

VI. Conclussions

This papers aims to assess the performance of 3D video streaming over lossy IP based networks using various video packetization modes according to the H.264/MVC standard. Particular interest was given to frame fragmentation into one or more NAL units and Fragment Units. In order to recover missing header information, an error resilient scheme is proposed that uses the header fields of correctly received fragment units to reconstruct headers of missing fragments. The performance evaluation considers parameters like the overhead caused by different packetization schemes, the maximum transmission unit size, the number of decoded frames and the resulted video quality in terms of Peak Signal to Noise Ratio (PSNR) for both base and non-base views, under varying network conditions and packetization modes. A test-bed platform that comprises of an RTP packetizer and de-packetizer, a UDP/IP streamer and client, implemented specifically for this study.

References

[1] A.Smolic, K.Mueller, P.Merkle, C.Fehn, P.Kauff, P.Eisert, and T.Wiegand, “3D Video and Free Viewpoint Video - Technologies, Applications and MPEG Standards”, *IEEE International Conference on Multimedia and Expo (ICME)*, Jul 2006.

[2] B.W.Micallef, C.J.Debono, "An analysis on the effect of transmission errors in real-time H.264-MVC Bit-streams", 15th IEEE Mediterranean Electrotechnical Conference (MELECON), Apr 2010.

[3] ITU-T Recommendation, “H.264 - Advanced video coding for generic audiovisual services”, Mar 2010.

[4] H. Schwarz, D. Marpe and T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”, IEEE Circuits and Systems for Video Technology, Vo. 17, No. 9, September 2007.

[5] Zhao Liu, Yuansong Qiao, Brian Lee, Enda Fallon, Karunakar A. K., Chunrong Zhang, Shuaijun Zhang, “Experimental Evaluation of H.264/Multiview Video Coding over IP Networks”, ISSC Conference, Trinity College Dublin, June 23-24, 2011.

[6] IETF Draft, “RTP Payload Format for MVC Video”, draft-ietf-wang-avt-rtp-mvc, October 2010.

[7] IETF Draft, ”RTP Payload Format for SVC Video”, draft-ietf-avt-rtp-svc, October 2010.

[8] IETF Draft, ”RTP Payload Format for H.264 Video”, draft-ietf-avt-rtp-rfc3984, February 2005.

[9] K.-D. Seo, J.-S. Kim, S.-H. Jung, and J.-J. Yoo, ”A Practical RTP Packetization Scheme for SVC Video Transport over IP Networks”, ETRI Journal, No. 2, April 2010, DOI: 10.4218/etrij.10.1409.0031.

[10] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G. Akar, G. Triantafyllidis, A. Koz, ”Coding Algorithms for 3DTV: A Survey”, IEEE Trans. Ob Circuits and Systems for Video Technology, Vol. 17, November 2007.

[11] S. Liu, Y. Chen, Y. Wang, M. Gabbouj, M. M. Hannuksela, H. Li, “Frame Loss Error Concealment For Multiview Video Coding”, June 2008, DOI: 10.1109/ISCAS.2008.4542206 .

[12] Y. Chen et al, , “The Emerging MVC Standard for 3D Video Services”, Eurasip Journal on Advances in Signal Processing, No. 8, 2009, DOI: 10.1155/2009/786015.

[13] G. B. Akar, A. M. Tekalp, C. Fehn and M. R, Civanlar, ‘Transport Methods in 3DTV—A Survey’, IEEE Trans. On Circuits and Systems for Video Technology, Vol. 17, 2007, DOI: 10.1109/ TCSVT.2007. 905365.

[14] ITU-T H.241, ‘Extended video procedures and control signals for H.300-series terminals’, Dec. 2009.

[15] Y. Chen, P. Pandit, S. Yea, “WD 4 Reference software for MVC,” JVT-AD207, JVT of ISO/IEC MPEG & ITU-T VCEG, Geneva, Jan-Feb. 2009.

[16] L., Rizzo, “Dummynet: a simple approach to the evaluation of network protocols,” ACM SIGCOMM Computer Communication Review, 27 (1), 31-41, 1997.

[17] A. Hallapuro, M. Hannuksela, J. Lainema, M. Salmimaa, K. Ugur, K. Willner, Y. Wang, Y.Chen, "Nokia "3D Video & The MVC Standard", Nokia research center.

[18] ftp://ftp.ne.jp/KDDI/multiview

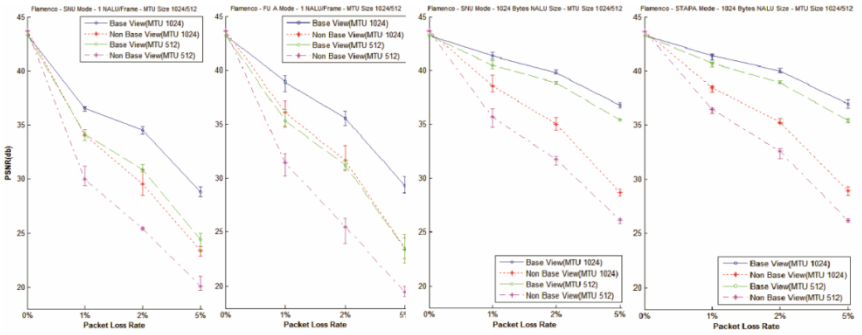


Figure 11a Figure 11b Figure 11c Figure 11d

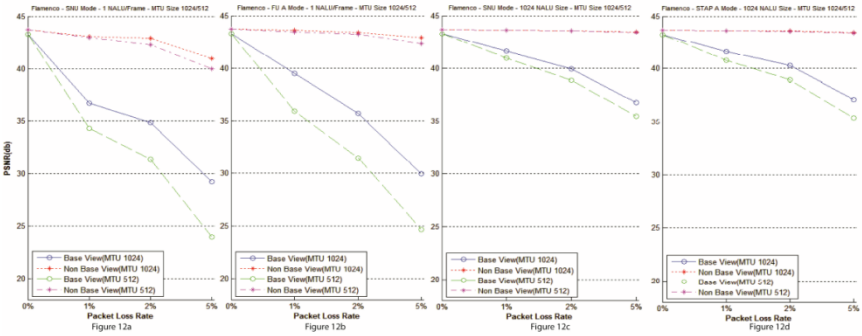


Figure 12a Figure 12b Figure 12c Figure 12d

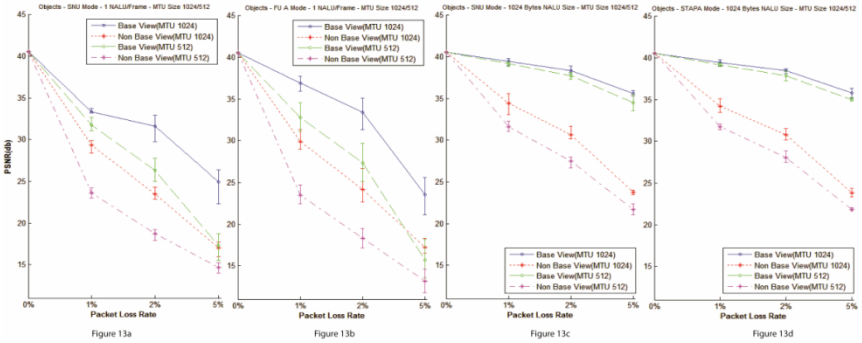


Figure 13a Figure 13b Figure 13c Figure 13d

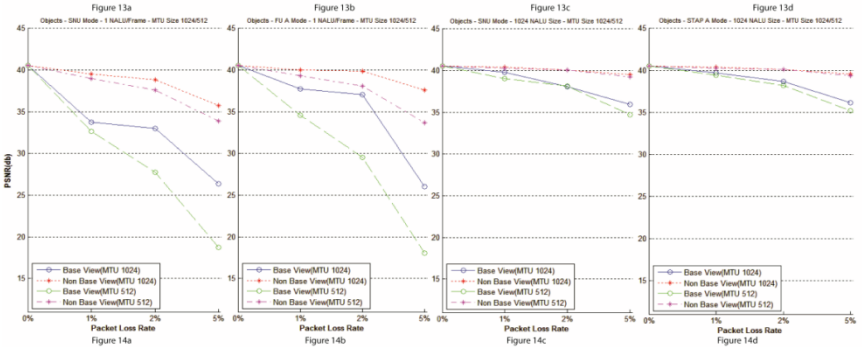


Figure 14a Figure 14b Figure 14c Figure 14d

- B. TRIDENTCOM 2012

Το 8^{ο} διεθνές ICST συνέδριο τεχνολογιών σε "testbeds" και ερευνητικές υποδομές για την ανάπτυξη των δικτύων θα διεξαχθεί στην Θεσσαλονίκη, Ελλάδα , 11-13 Ιουνίου 2012, με το όραμα να φέρει σε επαφή τεχνικούς ερευνητές από όλο τον κόσμο για να συζητήσουν για τις ερευνητικές υποδομές του μελλοντικού internet.

PERFORMANCE EVALUATION OF 3D STEREO VIDEO STREAMING OVER IP NETWORKS

Athanasios Kordelas and Tasos Dagiuklas

Dept. Of Telecommunication Systems and Networks,
TEI of Mesolonghi, Nafpaktos 30300, Greece
athankord@tesyd.teimes.gr , ntan@teimes.gr

Abstract. This paper presents the performance evaluation results of 3D Video Streaming using the H.264/MVC standard, through an experimental platform. Particular attention has been given to the evaluation of different video bit stream packetization modes, namely Single NAL Unit, Aggregation Packet and Fragmentation Units, by measuring overhead, number of decoded frames and perceived PSNR for both base and non-base view under different network conditions. Extensive test-bed experiments indicate that the non-base view is more sensitive to packet loss, due to its coding dependency from the base view.

Keywords: H.264/MVC, Performance Evaluation, 3D video streaming

1 Introduction

There are a lot of advances in 3D media technologies in terms of capturing, representation, coding, delivering, and visualizing to 3D displays. 3D media has been evolving in various areas, covering various market areas (e.g. scientific, medicine, education, training etc).

Multi-view Video Coding (MVC) standard, as specified by ISO/IEC 14496-10, supports the direct coding of multiple views and exploits inter-camera redundancy to reduce the bit rate [1]. The emerging video coding standard H.264 has been also extended in order to support MVC [2].

Apart of the coding efficiency, little work has been done regarding the performance of H.264/MVC video streaming of 3D stereo video sequences, in terms of view packetization efficiency and perceived video QoS of the different views across different network conditions. The aim of this paper is to assess the performance and the efficiency of 3D Stereo video using the H.264/MVC standard under different network conditions (i.e. packet loss), packet sizes (i.e. Maximum Transmission Unit (MTU)) and packetization options (i.e. number of Network Abstraction Layer Units (NALUs), RTP payload structures).

The paper is structured as follows: Section 2 outlines H.264/MVC overview, the MVC NAL Units Structure along with the different packetization modes, Section 3 presents the implementation of the test-bed, Section 4 analyzes the experimentation results along with performance measures in terms of transmission overhead and perceived PSNR for both views (left and right); Section 5 concludes the paper.

2 H.264/MVC Overview

H.264 Multi-View Coding supports the direct coding of multiple views and exploits inter-camera redundancy to reduce the bit rate. H.264 MVC gives very good 3D rendering capability, but the bit-rate of MVC encoded video is proportional to the number of views [2]. Furthermore, it incorporates Video Coding Layer (VLC) that produces the coded representation of the video and NAL where the video data are encoded in a prepared way for transmission over a broad variety of systems. According to H.264/MVC, the original AVC NAL header is extended in order to indicate the existence of a non base view payload used for 3D representation, as shown in Figure 1. Moreover, base view continuous to utilize the H.264/AVC NALU header, thus the decoder can still obtain a 2D representation from the MVC bit-stream.

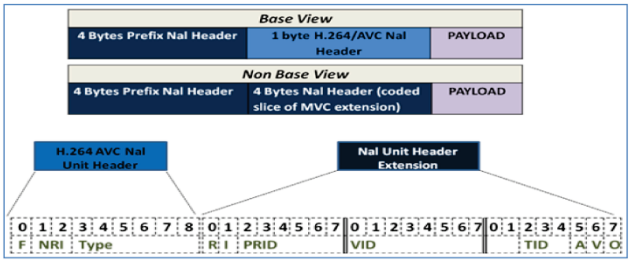


Fig. 1. MVC NALU Headers

In the NAL Unit Header, Nal_Unit_Type(Type) specifies the Type of the NAL Unit, Priority_id (PRID) specifies the importance of the NAL Unit, Temporal_id (TID) identifies the temporal layer or frame rate hierarchy and View_id (VID) specifies the view identifier that the NAL unit belong to. A detailed analysis of the bit fields used in MVC NAL Unit Header can be found in [5].

MVC exhibits packetization similarities with H.264 / Scalable Video Coding (SVC) standard [9]. There are three transmission modes for multiple-view video, as in Figure 2:

- Single Session Transmission (SST) – single RTP session,
- Multi-Session Transmission (MST) – multiple RTP sessions,
- Media-Aware Network Element (MANE) based transmission – combination of the above.

In the case of SST, each RTP session may either carry the base view bit stream or the base view with a number of different non-base views using a point-to-point unicast session. The server creates a sequence of RTP packets depending on each client's needs, by aggregating NAL units of the requested views.

Furthermore, in MST, each RTP session may carry the base view bit-stream, or more than one non-base view bit-streams, or the combination of a base view with a number of non-base views. MST is frequently applied in a multicast RTP session, where different clients may request different views of the multiple-view bit-stream. Moreover, in the case of unicast applications, each MVC video view is transmitted in different IP addresses, while in multicast applications each MVC video view is transmitted in its own IP multicast group. In terms of transmission, the number of RTP sessions may be equal to the number of different views, or less. The latter indicates encapsulation of multiple views in a single RTP session.

MANE generally resides in the path between the server and the clients. In a MANE-based system, while the server utilizes MST transmission, MANE collects all RTP sessions and depacketizes them. It customizes NALUs according to the client's needs based on an Adaptation Decision Taking Engine (ADTE) and aggregates packets for SST transmission to clients through a single transport address using in a single RTP session [9]. MANE is able to take all important information through RTP and NALU Headers. The need of such an intermediate system derives from the existence of limitations (e.g. due to firewalls and NAT protocols) that are present in real network environments.

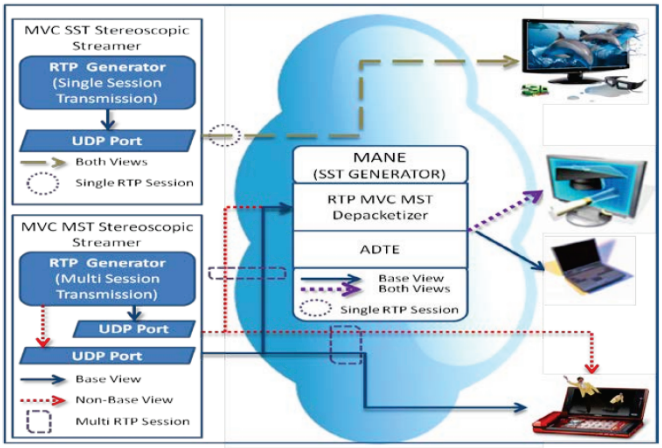


Fig. 2. MVC video Single Session Transmission (SST) & Multi-Session Transmission (MST) & MANE based Transmission.

RTP packets may use any of the three NALU payload structures as shown in Figure 3 [6]. When Single NALU mode is used, each RTP packet contains only one

NALU. Single NALU is allowed to contain NALU types from 1 to 23 and 30, 31. Aggregation packets may include more than one NALU by merging them. By aggregating NALUs the overhead due to RTP headers is reduced. Each NALU carried in an RTP packet is encapsulated in an aggregation unit first. There are five versions for aggregation units. The first four Single Time Aggregation Packets (STAPs) STAP-A, STAP-B and (Multi-TAPs) MTAP-16, and MTAP-24 are used also to packetize H.264/AVC data, while for H.264/MVC a new type of aggregation unit named Non-Interleaved MTAP (NI-MTAP) has been introduced in [5]. Every aggregation unit encapsulated in one aggregation packet contains one additional aggregation header which consists of 1 octet and has the form of a H.264/AVC NALU header. Depending on the version of the aggregation unit that has been used, some additional information is introduced regarding whether the NALs refer either to base or non-base view. In case of using STAP-A there is only one additional header (2 octet long), which indicates the size of the NALU [6].

Similarly, Fragmentation Units (FUs) contain part of a NALU. A FU allows NALU to be fragmented at the application layer, which provides advantages, for example when bit errors occur during transmission, fragmentation of a NALU allows the utilization of FEC/retransmission mechanism only at the specific fragment of the NAL Unit and not in the entire Single NAL Unit [3].

In this experimentation, an algorithm has been designed and developed at the depacketizer, which is able to reconstruct all the lost NALU Headers, so that the decoder is able to recognize and decode all Frames (NALU's). At the implementation section, there is a more detailed description of the algorithm. Each FU must be transmitted consecutively with upward sequence numbers. There are two versions of fragmentation units, namely FU-A and FU-B. The main difference between them is that the first uses timestamps, as opposed to the second, which uses the Decoding Order Number (DON) mechanism in order to retrieve the correct decoding order of the NALUs [7]. Except from the RTP header, both versions employ two additional headers that consist of one octet, named FU-indicator and FU-header [6]. Additionally, in the case of FU-B, one more additional header named DON must be used, which consists of 2 octets. It should be noted that the H.264/AVC NAL header after the fragmentation of the NALU must be deleted, while the information of the first three bits (2 bit fields) is moved to FU-indicator and the last 5 bits (Nal_Unit_Type) are moved to the last 5 bits of FU-header. It must be underlined that the above values are the same for all the fragments of a NALU. During the Depacketization procedure, the H.264/AVC NALU header is reconstructed using the FU-headers. The next 3 octets containing the NALU header extension lies at the payload.

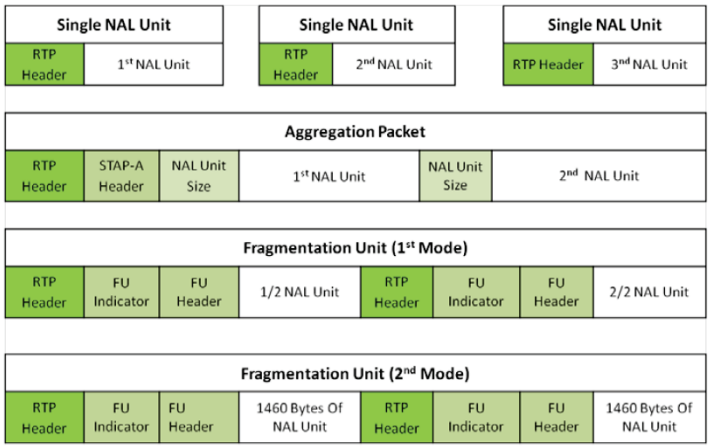


Fig. 3. RTP packet payload structures for MVC

There are three basic Packetization modes for SST namely Single NAL Unit mode, non-interleaved mode and interleaved mode. Single NAL Unit mode is being used for systems defined in accordance with the recommendation of ITU-T for H.241 [8]. Non-interleaved-mode is used for systems that are not compliant with the above recommendation and finally interleaved mode is suitable for systems that do not have latency restrictions and do not need decoding order transmission. All NAL units from type 1 up to type 29 can be used directly as packet payload for all the above modes, while for the newly introduced types, a PACSI NAL Unit (type 30) can be used directly as payload only in Single NAL Unit Mode and a NI-MTAP can be used only in non-interleaved mode.

Four new modes have been introduced for MST [4]:

- Non-interleaved timestamp based mode (NI-T)
- Non-interleaved cross-session decoding order number (CS-DON) based mode (NI-C)
- Non-interleaved combined timestamp and CS-DON mode (NI-TC)
- Interleaved CS-DON (I-C) mode.

The new modes re-use the three basic modes of SST for each of their individual RTP sessions. The two main differences of the above modes are whether or not to use decoding order transmission and what mechanisms they use in order to recover the correct decoding order through multiple sessions [5], [7]. If a system needs to transport both base and non-base views at low latency mode then one of the three first modes must be used, while for higher latency, I-C must be used for transport both base and non-base views [7].

3 Packetization Implementation

The end-to-end MVC system architecture for 3D video streaming over IP-based networks used to carry out experimental results is illustrated in Figure 4. For the encoding of the different view video sequences, a MVC Encoder / Decoder provided by Nokia have been used [11]. The network dynamics (packet delay, packet loss) have been emulated using the Network Emulator Dummynet [13].

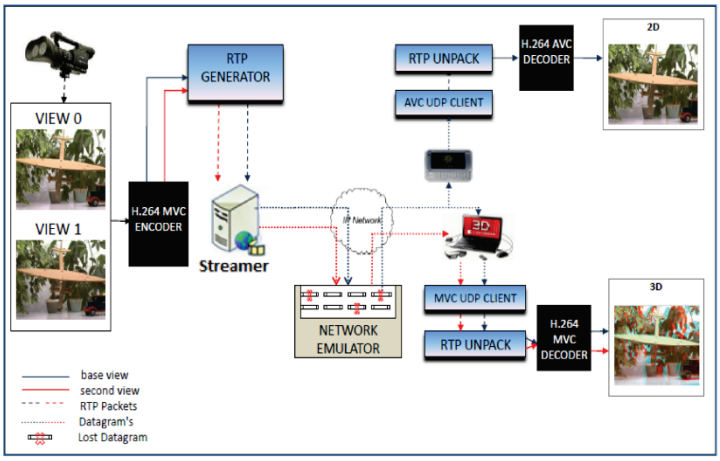


Fig. 4. E2E MVC System Architecture

3.1 RTP-Packetizer

The flow charts of the three main functions regarding MST/RTP packetization, implemented in this study, are illustrated in Figures 5, 6 and 7. The SNU & STAP-A functions are able to create one RTP packet at a time, as opposed to FU-A where one NALU is packetized in more than one RTP packets at a time.

The RTP Packetizer performs the following functions:

1. Print Packetizer Usage Information
2. Parse Command Line Arguments
3. Open MVC Coded Files
4. Find the length of each NALU for both Views
5. Packetize Sequence Parameter Set
6. Choose The Payload Type
7. Create RTP packets using the afore mentioned functions

8. Store Packet To Output Files
9. Create And Store Statistics To File
10. Close Files.

Figure 5 shows the function in which RTP packets using SNU payload type are constructed.

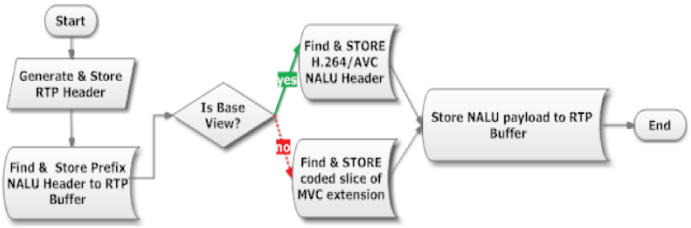


Fig. 5. Single NAL Units Function

Figure 6 illustrates the function in which RTP packets using STAP-A payload type are constructed. In order to aggregate NALUs in one RTP packet, a loop is repeated as many times as the number of the NALU's that are going to be aggregated. In each repetition one additional header consisting of 2 bytes is generated indicating the size of the aggregated NALU. Both RTP and STAP-A headers are generated only for the first aggregated NALU.

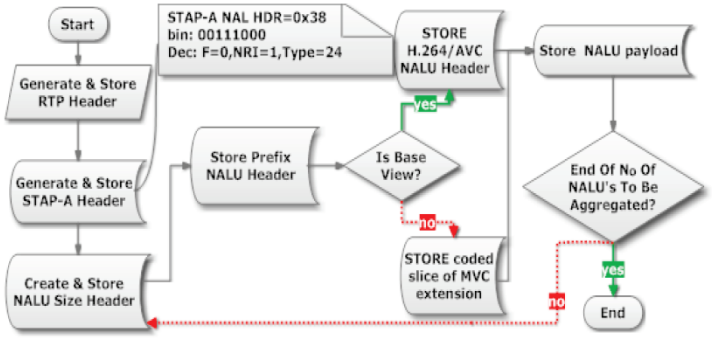


Fig. 6. Aggregation Packets Function (STAP-A)

Figure 7 shows the function in which RTP packets using FU-A payload type, are constructed. In order to create RTP packets, containing one fragment of the NALU, a

loop is repeated as many times as the number of the fragments of the NALU, resulting in as many RTP packets as the number of the fragments. Only the first FU contains the prefix NALU header while the H.264/NALU header is deleted, as described in the RTP standard [6]. An extra calculation is required at the last FU in order to determine whether or not some redundant bytes must be inserted at the end of the FU so that the sizes of all the fragments (RTP packets) are equal. In all FU's four reserved bytes for future implementations, are inserted.

The possible values of the FU-header are varying according to the value of the H.264/AVC NALU header and the fragmentation order of the FU (first, second, third, etc.) per NALU.

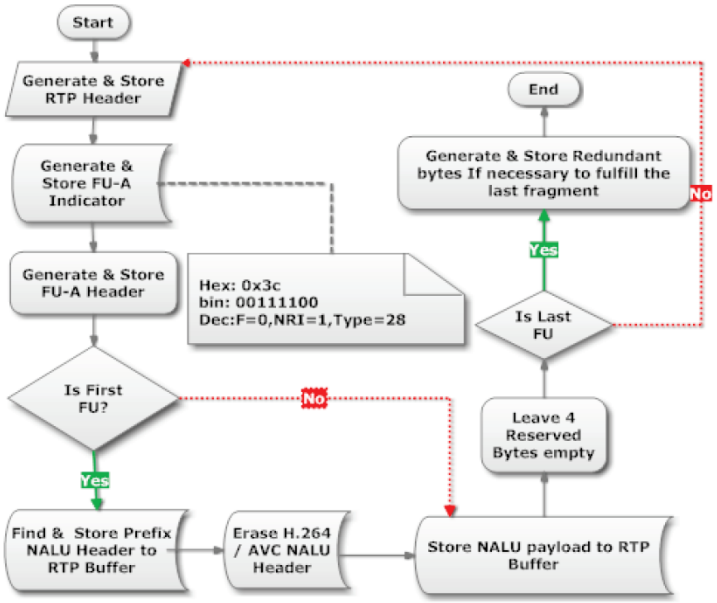


Fig. 7. Fragmentation Units Function (FU-A)

3.2 RTP-Depacketizer

The Depacketizer does almost exactly the reverse process and handles error resiliency. It is able to tolerate bit errors from all the headers. Furthermore, it is able to recognize the payload type used at the packetization process as well as the coding

parameters used (1 NALU per Frame, Several NALU's per Frame). In particular error resilience at RTP level can be achieved by recovering lost NALU headers in a number of different ways. Three approaches regarding the recovery of NALU headers include, firstly the estimation of the expected NALU type at a specific time and the partially rebuild of the lost NAL header using information such as the GOP size of the coded sequence. A second scheme is based on ARQ mechanism. According to this scheme the decoder is able to identify an IP packet containing a NALU header and to request a retransmission of the particular packet in case of packet loss. In this case the delay due to retransmission is kept to minimum, since only packets with NALU headers are retransmitted. In the context of this paper error resilience is achieved through implementing a third scheme that incorporates an RTP-aware error recovery mechanism in addition to the FU-A mode.

In details in the FU-A case the depacketizer is able to re-construct the 100% of the lost NALU Headers for both views. In order for the depacketizer to accomplish this task, it reconstructs a lost header using all the information from the remaining fragments of the NALU. All the FUs contain a FU header and a FU indicator. These headers are the same for all the fragments of a NALU. So when the depacketizer finds that the first fragment of the NALU has been lost, it then searches for the first next fragment that has been transmitted without a loss. Finally, it copies all the values needed of the bit-fields of the headers of the correct fragment and reconstructs the necessary information for the decoding H.264/AVC NALU header.

3.3 MVC MST UDP/IP Client

Figure 8 illustrates the Class Diagram representation of the MVC client tool. This tool consists of 4 classes. A *GUI* class results in a graphic interface used by the user to specify the transmission parameters (MVC/AVC, Payload Type, Number of NALUs per Frame, IP packet Size). When the user begins the transmission, the *Establish Connection* class is invoked and is establishing a TCP/IP connection in order to safely transmit all the above values and safely receive the Sequence Parameter Set. Finally, the *QuoteClientThread* class is invoked through the *EnstablishConnection* class as many times as the number of the views. This results in different threads of the same class running concurrently, hence creating different UDP/IP connections in order to receive packets of different views.

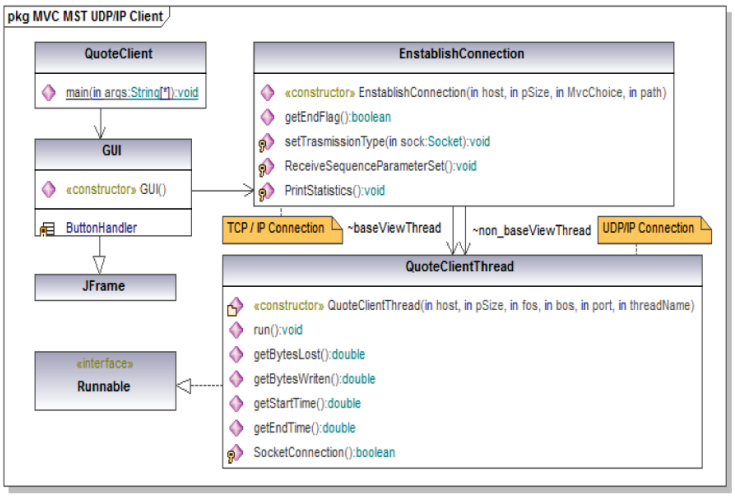


Fig. 8. MVC MST UDP/IP Client Class Diagram

4 Experimental Results

Due to the fact that MST offers certain advantages over SST, the aim of the experimentation work is to evaluate MST performance of H.264/MVC transmission for different networking conditions. For this purpose, the YUV 4.2.0 test sequence "flamenco" with 640x480 pixels resolution and 1000 frames at 25fps has been used [12]. Two different video packetization options have been considered. In the first one, each NALU contains a whole frame, while in the second one each NALU contains up to 1024 bytes, hence each frame need to be encapsulated in more than one NALUs. Each of the above options defines the type of the RTP payload structure that has to be applied. For the first option, SNU and FU modes are selected and SNU and STAP modes are selected for the second option. Although the RTP packetizer is able to use all the payload structures for both cases, there is no need of aggregating more than one frame in a RTP packet and also there is no need of fragmenting small NALUs in even smaller RTP packets. Finally, in both cases, two different MTU sizes (512 & 1024 bytes) have been used and 4 different packet loss rates (0%, 1%, 2%, and 5%). Each experiment is repeated several times (15 times), and the averaged values are presented.

Figure 9 illustrates the percentage escalation increase of STAP-A RTP MVC overhead compared to SNU (Single NAL Unit). Each STAP-A may encapsulate a

certain number of NALU's (ranging from 2 NALUs up to 10 NALUs in this particular experiment). As the number of the NALUs increases the overhead percentage decreases exponentially.

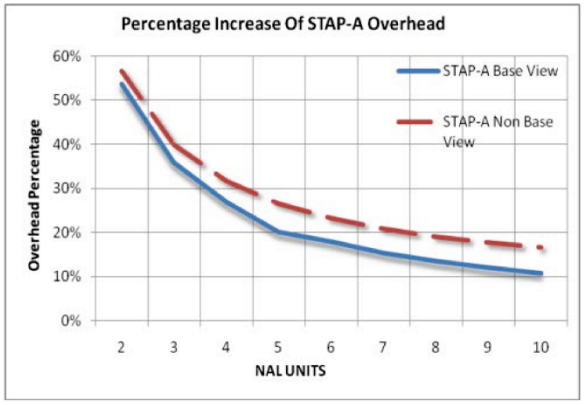


Fig. 9. Overhead comparison with respect to SNU between STAP-A mode base and non-base view

Figure 10 illustrates the corresponding results for FU-A. Unlike the STAP-A, in FU-A structure the overhead inserted in the sequence increases as the number of NALU fragments increases.

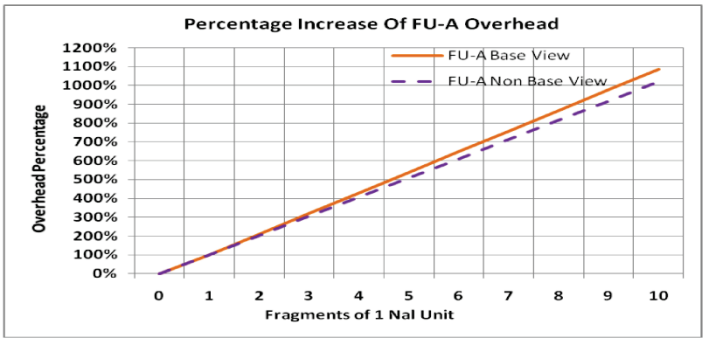


Fig. 10. Overhead comparison with respect to SNU between FU-A mode base and non-base view

4.1 One NALU per Frame

In the case of one NALU per frame, the loss of an IP packet that contains a NALU header sets the MVC decoder unable to decode the current NALU, since it cannot identify the type of data that the particular NALU contains. Therefore, the loss of such an IP packet results in the loss of a whole frame. Figure 11 illustrates the number of frames that can be decoded under specific network conditions. As expected, as packet loss increases, the number of frames from both views that can be decoded decreases.

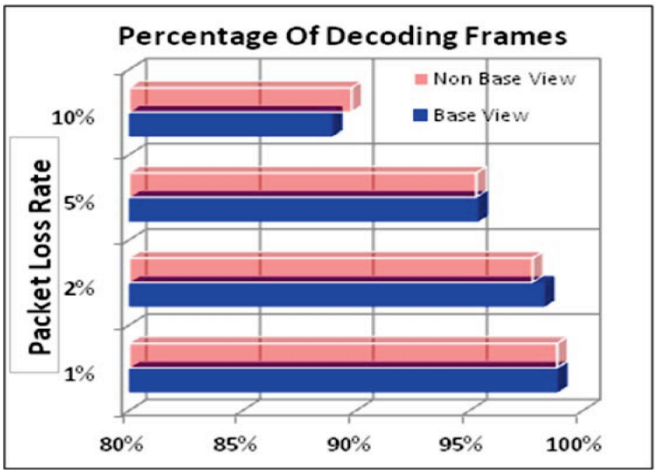
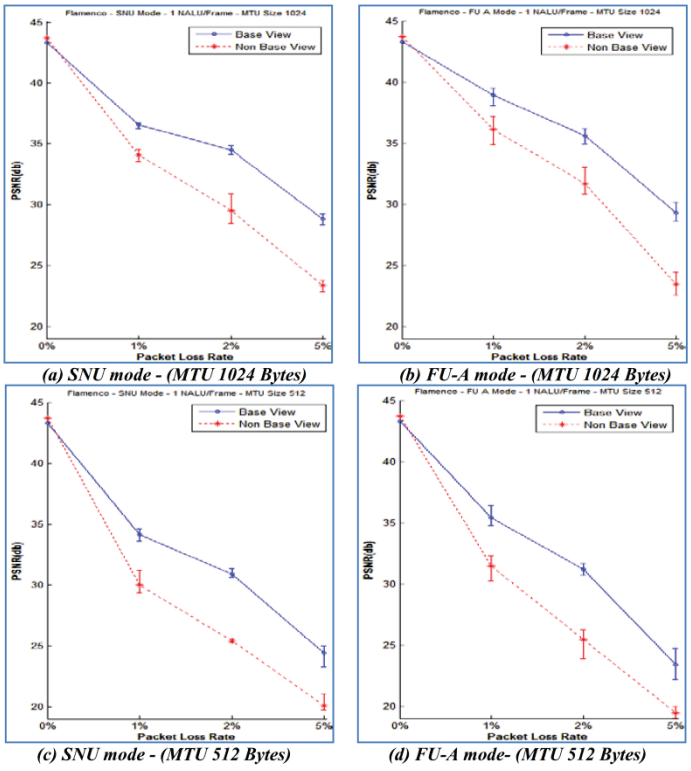


Fig. 11. Percentage of decoded frames of base and non-base views in SNU mode for different packet loss rates

Figure 12 illustrates the PSNR (Peak Signal to Noise Ratio) for different packet loss rates for both base and non-base view and MTU sizes (512 and 1024 bytes respectively), while the payload structure used is Single NAL Unit (Figure 12a, Figure 12c) and Fragmentation Unit (Figure 12b, Figure 12d). It is evident in all experiments that as packet loss increases, the perceived PSNR decreases substantially. When a packet of base view is lost, the resulting error propagates from the base view to the correlated non-base view. Evidently, a loss of base view packet causes severe deterioration to the non-base view video quality in terms of PSNR. The only way to minimize the high sensitivity in non-base view lost packets is to encode the non-base view independently, which as a result improves the perceived Quality of Service (QoS) at the cost of coding efficiency. In order to enable error resilience, frame copy technique has been used in SNU mode, while in FU-A case a proposed light reconstruction algorithm of lost NALU headers is implemented that allows all NALUs (Frames) to be decodable.

Figures 12a and 12c illustrate that in the SNU case, if the MTU size is halved then the average PSNR is reduced by almost 3,5dB and 3.8dB for the base and the non-base view, respectively. Additionally, the average PSNR resulted by the FU-A mode compared to that achieved by SNU is increased by 1,5dB and 1,4dB for base and non-base view, respectively. It must be mentioned that in fact this difference can be bigger if we consider that in SNU mode frame copy is applied in case of a frame loss. All experiments indicate that as the MTU size decreases so does the PSNR, while the fragmentation of RTP packets in transport layer (SNU) results in lower perceived QoS, compared to the fragmentation in the application layer (FU) that allows reconstruction.



(a) SNU mode - (MTU 1024 Bytes) *(b) FU-A mode - (MTU 1024 Bytes)*

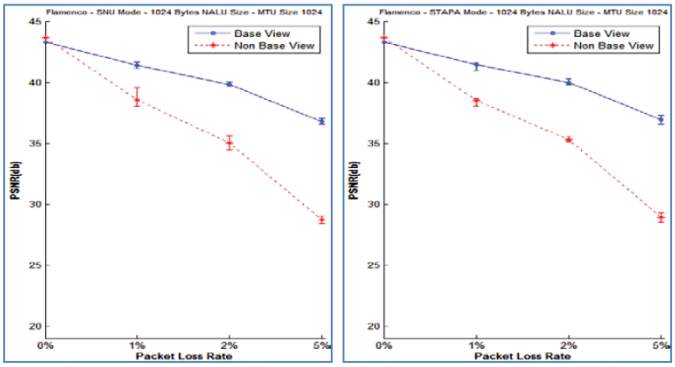
(c) SNU mode - (MTU 512 Bytes) *(d) FU-A mode- (MTU 512 Bytes)*

Fig. 12. Comparison of perceived video quality in terms of PSNR under different packet loss rates for the "flamenco" sequence (1 NALU/frame).

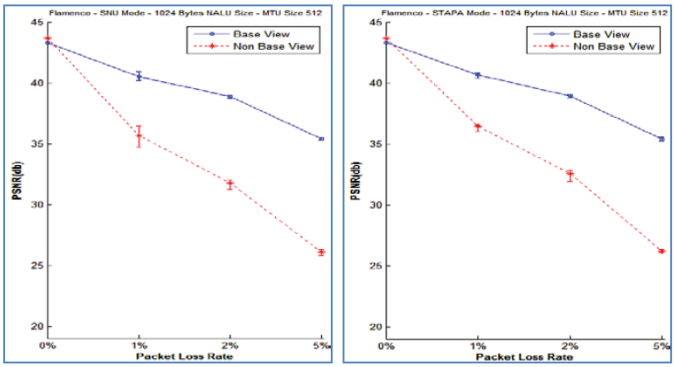
4.2 Several NALU per Frame

Although the algorithm constructed for the fragmentation of NALUs in the application layer results in better QoS than the SNU mode, this is achieved at the expense of increased overhead. A different way that results in better video quality is illustrated in Figure 13.

The NALUs of the coded sequences used in this experiment consist of 1024 bytes, thus each frame consists of more than one NALU. Two RTP payload structures are suggested for this scenario. The first one is SNU, in which each RTP packet encaptulates one NALU and each frame consists of more than one RTP packets .The second one is Aggregation Packets(STAP-A), in which each RTP packet aggregates all the NALUs of one frame, so each RPT packet contains one frame. Although the MTU size still affects PSNR ,the difference between this case and the previous one(1NALU/Frame) is smaller. The PSNR in this case is higher compared to the previous one. In case of "SNU mode NALU size of 1024 bytes and MTU 1024" at 5% packet loss the average PSNR is 36.8dB (Base View), while in case of "SNU mode one NALU per frame, MTU 1024" the average PSRN is 28.8dB (Base View). This is due to the fact that the decoder is capable of reconstructing the lost headers using the headers of the other NALUs of the same frame. Therefore, the "STAP-A mode, NALU size 1024 and MTU 1024" achieves the best PSNR compared to all ther cases (except SNU), while it achieves decreased overhead compared to the SNU mode.



a) SNU mode – (MTU 1024 Bytes) *(b) STAP-A mode – (MTU 1024 Bytes)*



(c) SNU mode – (MTU 512Bbytes) *(d) STAP-A mode – (MTU 512 Bytes).*

Fig. 13. Comparison of perceived video quality in terms of PSNR under different packet loss rates for the "flamenco" sequence (several *NALUs/frame).*

5 Conclusions

This paper presents the performance of H.264/MVC packetization schemes in terms of efficiency and perceived video quality for different network conditions (different packet loss rates) among different MVC packetization schemes. Particular attention was paid to MST due to its capabilities to support multicast and heterogeneity. MVC re-uses the packetization that has been used by SVC. It has been found that the non-base view is more sensitive to packet loss, due to its coding dependency to the base view. The best transmission mode is the one with the optimum MTU size (1024 bytes in these experiments), where each frame is encoded in more than one NALU's

References

[1] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G. Akar, G. Triantafyllidis, A. Koz, "Coding Algorithms for 3DTV: A Survey", IEEE Trans. Ob Circuits and Systems for Video Technology, Vol. 17, November 2007

[2] Y. Chen et al, , "The Emerging MVC Standard for 3D Video Services", Eurasip Journal on Advances in Signal Processing, No. 8, 2009, DOI: 10.1155/2009/786015

[3] G. B. Akar, A. M. Tekalp, C. Fehn and M. R, Civanlar, 'Transport Methods in 3DTV—A Survey', IEEE Trans. On Circuits and Systems for Video Technology, Vol. 17, 2007, DOI: 10.1109/ TCSVT.2007. 905365

[4] Y. Chen, P. Pandit, S. Yea, "WD 4 Reference software for MVC," JVT-AD207, JVT of ISO/IEC MPEG & ITU-T VCEG, Geneva, Jan-Feb. 2009

[5] IETF Draft, “RTP Payload Format for MVC Video”, draft-ietf-wang-avt-rtp-mvc, October 2010
[6] IETF Draft, ”RTP Payload Format for H.264 Video”, draft-ietf-avt-rtp-rfc3984, February 2005
[7] IETF Draft, ”RTP Payload Format for SVC Video”, draft-ietf-avt-rtp-svc, October 2010
[8] ITU-T H.241, ‘Extended video procedures and control signals for H.300-series terminals’, December 2009
[9] K.-D. Seo, J.-S. Kim, S.-H. Jung, and J.-J. Yoo, ”A Practical RTP Packetization Scheme for SVC Video Transport over IP Networks”, ETRI Journal, No. 2, April 2010, DOI: 10.4218/etrij.10.1409.0031
[10] S. Liu, Y. Chen, Y. Wang, M. Gabbouj, M. M. Hannuksela, H. Li, “Frame Loss Error Concealment For Multiview Video Coding”, June 2008, DOI: 10.1109/ISCAS.2008.4542206
[11] A. Hallapuro, M. Hannuksela, J. Lainema, M. Salmimaa, K. Ugur, K. Willner, Y. Wang, Y.Chen, “Nokia “3D Video & The MVC Standard”, Nokia research center.
[12] ftp://ftp.ne.jp/KDDI/multiview
[13] L., Rizzo, “Dummynet: a simple approach to the evaluation of network protocols,” ACM SIGCOMM Computer Communication Review, 27 (1), 31-41, 1997