ΤΕΙ ΜΕΣΟΛΟΓΓΙΟΥ
ΤΜΗΜΑ ΤΗΛΕΠ/ΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΣΑΡΜΟΓΗ ΥΠΗΡΕΣΙΩΝ VIDEO ΜΕ ΒΑΣΗ ΤΟ ΠΡΩΤΟΚΟΛΛΟ HTTP

ΠΑΠΑΔΟΓΙΑΝΝΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ Α.Μ. 0817

Επιβλέπων: Τάσος Νταγιούκλας, Ph.D.
Ναύπακτος 2012

ΠΕΡΙΛΗΨΗ

Το HTTP χρησιμοποιείται από το τελικό χρήστη για υπηρεσίες video streaming από ένα server. Ο ρυθμός μετάδοσης εξαρτάται από τα χαρακτηριστικά του τελικού χρήστη και το διαθέσιμο εύρος ζώνης μεταξύ του χρήστη και του server. Στα πλαίσια της πτυχιακής εργασίας αναπτύσσεται και εξετάζεται ένα σύστημα video streaming χρησιμοποιώντας το πρωτόκολλο HTTP. Στην περίπτωση αυτή ο χρήστης έχει τη δυνατότητα να κατεβάζει διαφορετικές ποιότητες του video από το server με βάση το feedback που λαμβάνει από τον τελικό χρήστη. Για την handshake επικοινωνία μεταξύ του χρήστη και του server χρησιμοποιείται το πρωτόκολλο HTTP.

ABSTRACT

HTTP protocol is used by the final user for video streaming services provided by a server. The bit-rate depends on the final user's features and the available bandwidth between the user and the server. As part of the thesis a video streaming system has been developed and tested using HTTP protocol. In this case the user has the ability to download different video qualities from the server based on the feedback received by the final user. For handshake communication between the user and the server HTTP protocol is used.

Λέξεις Κλειδιά

DASH, HTTP, PSNR, packet loss, Adobe streaming

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΙΣΑΓΩΓΗ. 4

1. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ. 6

1.1 ΣΤΟΙΒΑ ΠΡΩΤΟΚΟΛΛΩΝ ΤΟΥ INTERNET- TCP/IP. 6
1.1.1 TCP/IP 6
1.1.2 Επίπεδα της Σουίτας TCP/IP 7
1.1.3 Ανάλυση των επιπέδων 7
1.1.3.1 Εφαρμογής 7
1.1.3.2 Μεταφοράς 7
1.1.3.3 Δικτύου 9
1.1.3.4 Συνδέσμου 10
1.2 ΤΡΟΠΟΙ ΜΕΤΑΔΟΣΗΣ ΔΕΔΟΜΕΝΩΝ 11
1.2.1 Unicast τρόπος μετάδοσης δεδομένων 11
1.2.2 Broadcast τρόπος μετάδοσης δεδομένων 12
1.2.3 Multicast τρόπος μετάδοσης δεδομένων 13
1.3 ΜΕΘΟΔΟΙ ONLINE ΑΝΑΠΑΡΑΓΩΓΗΣ ΒΙΝΤΕΟ 15
1.3.1 Progressive Download 15
1.3.2 Stateful streaming 16
1.3.3 Adaptive bitrate streaming 17
1.3.3.1 Εισαγωγή 17
1.3.3.2 Τρέχουσα χρήση του adaptive bitrate streaming 18
1.3.3.3 Πλεονεκτήματα του adaptive bitrate streaming 18
1.4 ΠΡΟΚΛΗΣΕΙΣ ΚΑΙ QUALITY OF EXPERIENCE (QOE) 19
1.4.1 Δίκτυο σύνδεσης και τρέχουσα ασφάλεια 19
1.4.2 Έλεγχος εύρους ζώνης 20
1.4.3 Πολλαπλοί χρήστες και πολλαπλές αναλύσεις 20
1.4.4 Ασφάλεια περιεχομένου 20

2. DYNAMIC ADAPTIVE STREAMING OVER HTTP (DASH) 22

2.1 ΕΙΣΑΓΩΓΗ. 22
2.2 ΛΕΙΤΟΥΡΓΙΑ. 22
2.3 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΚΑΙ ΣΥΓΚΡΙΣΗ ΤΟΥ DASH 24

3. ΓΝΩΣΤΕΣ ΥΛΟΠΟΙΗΣΕΙΣ ΤΟΥ DASH. 29

3.1 ADOBE SYSTEMS'S HTTP DYNAMIC STREAMING 29
3.2 APPLE HTTP LIVE STREAMING 31
3.3 MICROSOFT'S SMOOTH STREAMING 33
3.4 OCTOSHAPE MULTI-BITRATE 35
3.5 DASH VLC PLUGIN 36
3.6 GPAC 38
3.7 GSTREAMER 39

4. ΕΠΙΣΚΟΠΗΣΗ ΤΟΥ DASH ΜΕ ΤΟ "ADOBE HTTP DYNAMIC STREAMING" 40

4.1 ΕΙΣΑΓΩΓΗ 40
4.2 ΠΕΡΙΒΑΛΛΟΝ 42
4.3 ΠΕΙΡΑΜΑ 43
4.4 ΣΥΜΠΕΡΑΣΜΑΤΑ ΑΠΟ ΤΑ ΠΕΙΡΑΜΑΤΑ 49

5. ΒΙΒΛΙΟΓΡΑΦΙΑ. 50

6. ΠΑΡΑΡΤΗΜΑ: ΚΩΔΙΚΑΣ ADOBE DASH PLAYER 51

6.1 ΚΩΔΙΚΑΣ 51

Εισαγωγή.

Το Internet και η παγκόσμια δικτύωση βρίσκονται συνεχώς σε εξέλιξη. Στις αρχές οι ιστοσελίδες ήταν ένα καθαρό κείμενο στο οποίο αρκετά γρήγορα ενσωματώθηκαν απλές εικόνες αλλά και κινούμενες εικόνες τύπου "Animated GIF". Η εμφάνιση του βίντεο έγινε δυνατή μερικά χρόνια αργότερα.

Σήμερα η δυνατότητα για αναπαραγωγή βίντεο βρίσκεται παντού στο διαδίκτυο, αλλά η εμπειρία μιας ομαλής αναπαραγωγής δεν μπορεί να εγγυηθεί: μεγάλοι χρόνοι εκκίνησης, η αδυναμία για αναζήτηση ενός συγκεκριμένου σημείου-στιγμής πάνω στο βίντεο αλλά και οι διακοπές στο buffering δεν αποτελούν εξαιρέσεις. Ωστόσο τα τελευταία χρόνια νέες τεχνικές μετάδοσης έχουν υλοποιηθεί για να λύσουν αυτά τα προβλήματα όπως και το "dynamic adaptive streaming over http" που θα συζητηθεί παρακάτω.

Για την επικοινωνία πάνω από το ίντερνετ χρησιμοποιείται το πρωτόκολλο IP συνήθως σε συνεργασία με το πρωτόκολλο TCP. Το IP είναι υπεύθυνο για την μεταφορά του πακέτου διαμέσου του δικτύου στον κατάλληλο αποδέκτη. Το TCP εγγυάται ότι όλα τα πακέτα θα φτάσουν χωρίς πρόβλημα με βάση τις σωστές οδηγίες και αν χρειαστεί επανατοποθετεί τα λανθασμένα ή χαμένα πακέτα στη διαδικασία της μεταφοράς και άλλες φορές. Αυτό είναι απαραίτητο για την αξιόπιστη μεταφορά αρχείων μέσα από το ίντερνετ, αλλά αυτές οι τεχνικές διόρθωσης λαθών έχουν και ένα κόστος. Το TCP χρειάζεται να περιμένει και να επιβεβαιώνει την μεταφορά κάθε πακέτου κάτι το οποίο αποτελεί την επιθυμητή συμπεριφορά για το κατέβασμα ενός εγγράφου αλλά δεν ισχύει το ίδιο και για την online αναπαραγωγή ενός βίντεο καθώς θα πάγωνε την αναπαραγωγή λόγω αναμονής πακέτων και για αυτό θα ήταν

προτιμότερο να παραληφθεί κάποιο πακέτο και να παρουσιαστεί κάποια μικρή αλλοίωση στην εικόνα ή τον ήχο.

Το πρωτόκολλο RTP δεν χρησιμοποιεί τις τεχνικές διόρθωσης του TCP αντιθέτως συνεργάζεται με το UDP πρωτόκολλο. Το UDP πρωτόκολλο φιλτράρει μόνο τα κατεστραμμένα πακέτα αλλά δεν κάνει καμία διόρθωση. Το RTP είναι ακόμα μία από τις πιο δημοφιλείς μορφές για streaming ήχου και βίντεο, ειδικά στο VoIP και στην συμβολή του βίντεο γενικά. Λόγω της ελλιπούς τεχνικής ελέγχου λαθών χρησιμοποιείται πιο πολύ σε εσωτερικά ελεγχόμενα δίκτυα. Επιπλέον τα περισσότερα τείχη προστασίας είναι ρυθμισμένα να αποτρέπουν την κίνηση UDP και παράλληλα την κίνηση RTP.

Το πρωτόκολλο HTTP χρησιμοποιείται για να εξυπηρετήσει σχεδόν κάθε ιστοσελίδα στο διαδίκτυο. Η πλειοψηφία των τειχών προστασίας επιτρέπει την κίνηση του. Σαν αποτέλεσμα αποτελεί ένα ελκυστικό πρωτόκολλο για την μεταφορά αρχείων συμπεριλαμβανομένων και βίντεο στο ευρύ κοινό. Η νέα μέθοδος για την προβολή βίντεο πάνω από HTTP ονομάζεται “dynamic adaptive streaming over http”. Με αυτή την μέθοδο επιτυγχάνεται μια ομαλή βέλτιστη αναπαραγωγή βίντεο προσαρμοσμένη στις δυνατότητες του εκάστοτε δικτύου αλλά και της εκάστοτε τερματικής συσκευής.

Στα πλαίσια της πτυχιακής εργασίας θα ασχοληθούμε με αυτή την μέθοδο χρησιμοποιώντας το adobe http dynamic streaming σε ειδικό περιβάλλον που το bandwidth θα αλλάζει δυναμικά και τυχαία με βάση την Gaussian κατανομή και το packet loss θα αλλάζει επιλεκτικά σε διαφορετικές τιμές επίσης.

1. Βασικές έννοιες.

1.1 Στοίβα πρωτοκόλλων του Internet- TCP/IP.

1.1.1 TCP/IP

Το '"TCP/IP" (Transmission Control Program/Internet Protocol=Πρόγραμμα Ελέγχου Μετάδοσης και πρωτόκολλο του Internet) είναι μια συλλογή πρωτοκόλλων επικοινωνίας στα οποία βασίζεται το Διαδίκτυο αλλά και μεγάλο ποσοστό των εμπορικών δικτύων. Η ονομασία TCP/IP προέρχεται από τις συντομογραφίες των δυο κυριότερων πρωτοκόλλων που περιέχει : το TCP ή Transmission Control Protocol (Πρωτόκολλο Ελέγχου Μετάδοσης) και το IP ή Internet Protocol (Πρωτόκολλο Διαδικτύου).

Αυτή η συλλογή πρωτοκόλλων, όπως και πολλές άλλες άλλωστε, είναι οργανωμένη σε στρώματα ή επίπεδα (layers). Το καθένα τους απαντά σε συγκεκριμένα προβλήματα μεταφοράς δεδομένων και παρέχει μια καθορισμένη υπηρεσία στα υψηλότερα στρώματα. Τα ανώτερα επίπεδα είναι πιο κοντά στη λογική του χρήστη και εξετάζουν πιο αφηρημένα δεδομένα, στηριζόμενα σε πρωτόκολλα χαμηλότερων στρωμάτων για να μεταφράσουν δεδομένα σε μορφές που μπορούν να διαβιβαστούν με φυσικά μέσα.

Το μοντέλο OSI, το οποίο παραμένει έως σήμερα μόνο θεωρητικό, προτείνει την κατάταξη των πρωτοκόλλων δικτύων σε έναν οργανωμένο σωρό 7 στρωμάτων. Συγκρίσεις ανάμεσα στο μοντέλο OSI και το TCP/IP δείχνουν τη σημασία των πρωτοκόλλων που περιέχονται στη σουίτα IP, από την άλλη πλευρά όμως μπορεί να προκληθεί σύγχυση, καθώς το TCP/IP αποτελείται από μόνο 4 στρώματα.

1.1.2 Επίπεδα της Σουίτας TCP/IP

Τα πρωτόκολλα Διαδικτύου κάνουν χρήση της ενθυλάκωσης (encapsulation) για να παρέχουν γενικά πρωτόκολλα και υπηρεσίες. Ένα πρωτόκολλο υψηλού στρώματος χρησιμοποιεί τα πρωτόκολλα των κατώτερων για να λειτουργήσει.

Ένα απλουστευμένο σχεδιάγραμμα της στοίβας του μοντέλου TCP/IP ακολουθεί :

4	**Εφαρμογής**	π.χ. HTTP, FTP, DNS *(Πρωτόκολλα δρομολόγησης, όπως το RIP, που βασίζονται στο πρωτόκολλο UDP μπορούν επίσης να καταχωρηθούν στο στρώμα Δικτύου)*
3	**Μεταφοράς**	π.χ. TCP, UDP, RTP *(Πρωτόκολλα δρομολόγησης, όπως το OSPF, που λειτουργούν πάνω από το IP, μπορούν επίσης να καταχωρηθούν στο στρώμα Δικτύου)*
2	**Δικτύου**	Για το TCP/IP, χρησιμοποιείται μόνο το IP *(Τα πρωτόκολλα ICMP και IGMP, παρ'όλο που βασίζονται πάνω στο IP για την λειτουργία τους, καταχωρούνται στο στρώμα Δικτύου. Το ARP αποτελεί μια από τις ολιγάριθμες εξαιρέσεις, εφ'όσον είναι ανεξάρτητο του IP)*
1	**Συνδέσμου**	π.χ. Ethernet, Token Ring, κλπ.

(Εικόνα 1: μοντέλο TCP/IP)

Αυτά τα 4 επίπεδα στην εικόνα 1, συναποτελούν το Μοντέλο Διαστρωμάτωσης του Internet ή αλλιώς, Μοντέλο αναφοράς του Internet.

1.1.3 Ανάλυση των επιπέδων

1.1.3.1 Εφαρμογής

Το στρώμα εφαρμογής χρησιμοποιείται από την πλειοψηφία των δικτυωμένων προγραμμάτων. Το πρόγραμμα παραδίδει τα δεδομένα σε μια μορφή που ορίζει το ίδιο.

Εφόσον το TCP/IP δεν παρέχει στρώματα μεταξύ των στρωμάτων εφαρμογής και μεταφοράς, όλες οι λειτουργίες παρουσίασης και συνεδρίας πρέπει να υλοποιηθούν σ' αυτό το επίπεδο. Αυτή η διαδικασία διευκολύνεται με την χρήση βιβλιοθηκών.

1.1.3.2 Μεταφοράς

Το στρώμα μεταφοράς είναι υπεύθυνο για την μεταφορά μηνυμάτων, ανεξαρτήτως του υποκείμενου δικτύου, με έλεγχο σφαλμάτων (error control), κατάτμηση (fragmentation) και ρύθμιση ροής (flow control). Η μετάδοση μηνυμάτων μεταξύ δυο οντοτήτων μπορεί να κατηγοριοποιηθεί ως εξής:

1) connection-oriented, π.χ. TCP
2) connectionless, π.χ. UDP

Η λειτουργία του στρώματος αυτού μπορεί να συγκριθεί με αυτή οποιουδήποτε μηχανισμού/μέσου μεταφοράς, π.χ. ένα όχημα που πρέπει να εξασφαλίζει την πλήρη και ασφαλή διακίνηση του φορτίου του. Το στρώμα μεταφοράς παρέχει αυτή την υπηρεσία σύνδεσης εφαρμογών μεταξύ τους, κάνοντας χρήση θυρών (ports). Καθώς το IP προσφέρει μόνο παράδοση όσο το δυνατόν καλύτερα (best effort delivery), το στρώμα μεταφοράς είναι το πρώτο επίπεδο όπου λαμβάνεται υπόψιν το θέμα της αξιοπιστίας.

Παραδείγματος χάρη, σε μια προσπάθεια αξιόπιστης μετακίνησης δεδομένων, το TCP που είναι ένα connection-oriented πρωτόκολλο, έχει τα εξής χαρακτηριστικά:

- τα δεδομένα έρχονται στην ίδια σειρά με την οποία στάλθηκαν
- ελάχιστος έλεγχος σφαλμάτων
- ανεπιθύμητα αντίγραφα απορρίπτονται
- χαμένα/απορριμμένα πακέτα ξαναστέλνονται
- έλεγχος κυκλοφοριακής συμφόρησης (congestion control)

Τα πρωτόκολλα δυναμικής δρομολόγησης (dynamic routing), που κανονικά θα έπρεπε να βρίσκονται σε αυτό το στρώμα του TCP/IP (αφού λειτουργούν πάνω από το IP) αντιμετωπίζονται συχνά ως τμήματα του επίπεδου δικτύου (π.χ. το OSPF).

Το νέο SCTP είναι επίσης ένας "αξιόπιστος", connection-oriented μηχανισμός μεταφοράς. Είναι stream-oriented, όχι byte-oriented όπως το TCP, και προσφέρει την δυνατότητα multiplexing πολλών ρευμάτων (stream) σε μια μόνο σύνδεση. Προτείνει υποστήριξη multi-homing, την δυνατότητα δηλαδή για μια οντότητα να μπορέσει, στα πλαίσια μιας συγκεκριμένης σύνδεσης, να κάνει χρήση πολλαπλών (εφόσον υπάρχουν) διευθύνσεων IP, που αντιπροσωπεύουν πολλαπλές interfaces

(διασυνδετικές διατάξεις), έτσι ώστε αν κάποια παρουσιάσει βλάβη, να μη χαθεί η σύνδεση.

Το UDP είναι ένα connectionless πρωτόκολλο διαγραμμάτων δεδομένων (datagrams). Όπως και το IP, είναι ένα best effort ή "αναξιόπιστο" πρωτόκολλο: ο έλεγχος σφαλμάτων είναι αδύναμος (απλό checksum). Χρησιμοποιείται κυρίως σε εφαρμογές streaming μέσων (ήχος, βίντεο, κλπ.) όπου η έγκαιρη άφιξη των δεδομένων είναι πιο σημαντική από την ακεραιότητα τους. Ο χρόνος που κερδίζεται σε σχέση με τα connection-oriented πρωτόκολλα, που πρέπει να καθιερώσουν μια αξιόπιστη σύνδεση, το καθιστά ιδανικό για απλές ερώτημα/απάντηση εφαρμογές (π.χ. DNS).

Το TCP και το UDP εκμεταλλεύονται από εφαρμογές που διακρίνονται (στο επίπεδο του δικτύου) από την θύρα TCP ή UDP τους. Ορισμένοι αριθμοί θυρών είναι κλειστοί και αναφέρονται σε πολύ συγκεκριμένες εφαρμογές (βλ. well known port numbers).

Το RTP είναι ένα πρωτόκολλο διαγραμμάτων δεδομένων σχεδιασμένο για στοιχεία πραγματικού χρόνου (real-time) όπως τα streaming audio και video. Αν και παρουσιάζεται στο στρώμα μεταφοράς (αντί για το επίπεδο συνεδρίας), βασίζεται στο UDP για την λειτουργία του.

1.1.3.3 Δικτύου

Ο σκοπός του στρώματος δικτύου είχε αρχικά καθοριστεί ως η μεταφορά πακέτων μέσω ενός ενιαίου δικτύου.

Με την εμφάνιση πιο σύνθετων μορφών δικτύων, προστέθηκαν επιπλέον χαρακτηριστικά στο στρώμα αυτό, έτσι ώστε ο ρόλος του να είναι πια η διακίνηση δεδομένων από το δίκτυο πηγή στο δίκτυο προορισμού. Αυτό προϋποθέτει συνήθως την δρομολόγηση πακέτων διαμέσου ενός δικτύου δικτύων (internetwork) ή διαδικτύου.

Στην σουίτα πρωτοκόλλων Διαδικτύου, το IP μεταφέρει τα πακέτα δεδομένων από την πηγή, στον προορισμό. Το IP μπορεί να εξυπηρετήσει διάφορα πρωτόκολλα ανωτέρων επιπέδων (upper layer protocols) ˙ το καθένα τους προσδιορίζεται με έναν αποκλειστικό αριθμό πρωτοκόλλου: π.χ. το ICMP και το IGMP έχουν τους αριθμούς 1 και 2 αντίστοιχα.

Μερικά πρωτόκολλα που στηρίζονται στο IP, π.χ. το ICMP (χρησιμοποιείται για την διάδοση διαγνωστικών πληροφοριών σχετικά με την μεταφορά πακέτων μέσω IP) παρουσιάζονται πάνω από το IP αλλά παρέχουν υπηρεσίες επιπέδου διαδικτύου, απεικονίζοντας έτσι την ασυμβατότητα μεταξύ του Διαδικτύου, των πρωτοκόλλων Διαδικτύου και του μοντέλου OSI. Όλα τα πρωτόκολλα δρομολόγησης (π.χ. BGP, OSPF, RIP, κλπ.) ανήκουν επίσης στο στρώμα δικτύου, αν και θα μπορούσαν να τοποθετηθούν σε ανώτερα επίπεδα.

1.1.3.4 Συνδέσμου

Το στρώμα αυτό, ρόλος του οποίου είναι η διακίνηση πακέτων του επιπέδου δικτύου μεταξύ δυο οντοτήτων, δεν είναι στην ακρίβεια μέρος της σουίτας πρωτοκόλλων Διαδικτύου, διότι το IP λειτουργεί με διάφορα στρώματα συνδέσμου. Η διαδικασία διαβίβασης (αντ. λήψης) πακέτων σε (αντ. από) ένα συγκεκριμένο επίπεδο συνδέσμου μπορεί να ελέγχεται είτε από τον οδήγο του interface, είτε το firmware ή σύνολο εξειδικευμένων κυκλωμάτων (chipsets), είτε τέλος από ένα συνδυασμό τον προ-αναφερθέντων. Αυτά θα εκτελέσουν τις λειτουργίες σύνδεσης δεδομένων (data link), όπως π.χ. την πρόσθεση επικεφαλίδας (packet header) πριν την αποστολή, την ίδια τη διαβίβαση του πλαισίου (frame) με τη χρήση ενός φυσικού μέσου.

Για συνδέσεις μέσω μόντεμ (σε γραμμή τηλεφώνου), τα πακέτα IP μεταφέρονται συνήθως χρησιμοποιώντας το PPP. Σε ευρυζωνικές συνδέσεις (π.χ. ADSL) συναντάμε το PPPoE. Σε τοπικά δίκτυα, τα πρωτόκολλα Ethernet ή IEEE 802.11 (για

ενσύρματα ή ασύρματα δίκτυα αντίστοιχα) είναι πιο κοινά. Για δίκτυα ευρείας περιοχής (WAN) χρησιμοποιούνται συχνά το PPP πάνω σε γραμμές T-carrier ή E-carrier, το Frame relay, το ATM ή το Packet over SONET/SDH (POS).

Το στρώμα συνδέσμου είναι επίσης το επίπεδο όπου τα πακέτα μπορούν να αναχαιτιστούν για να σταλθούν σ' ένα ιδεατό ιδιωτικό δίκτυο (Virtual Private Network, VPN). Σ' αυτήν την περίπτωση, τα δεδομένα του επιπέδου αυτού αντιμετωπίζονται ως δεδομένα εφαρμογής, και "ξανακατεβαίνουν" την στοίβα πρωτοκόλλων Διαδικτύου για να σταλθούν. Στη λαμβάνουσα πλευρά, τα δεδομένα ανεβαίνουν δυο φορές την στοίβα (μια για το VPN και μια δεύτερη για τη δρομολόγηση).

Το φυσικό επίπεδο, που αποτελείται από τα φυσικά στοιχεία του δικτύου (π.χ. hubs, repeaters, καλώδια δικτύου, οπτικές ίνες, ομοαξονικά καλώδια, κάρτες δικτύων) και τις προδιαγραφές χαμηλού επιπέδου των σημάτων (τάση, συχνότητα, κλπ.), θεωρείται συχνά ως μέρος του στρώματος συνδέσμου.

1.2 Τρόποι μετάδοσης δεδομένων

1.2.1 Unicast τρόπος μετάδοσης δεδομένων

Τα περισσότερα υψηλού επιπέδου πρωτόκολλα παρέχουν μόνο unicast τρόπο μετάδοσης δεδομένων. Αυτό σημαίνει ότι μια μονάδα ενός δικτύου έχει την δυνατότητα να στέλνει δεδομένα μόνο σε μια άλλη κάθε στιγμή. Όλες οι unicast μεταφορές δεδομένων είναι μεταφορές "point-to-point". Έτσι αν ένα μέλος ενός δικτύου θελήσει να στείλει την ίδια πληροφορία κατ' αυτόν τον τρόπο σε N αποδέκτες, θα πρέπει να στείλει N αντίγραφα της ίδιας πληροφορίας, αντιστοιχίζοντας το καθένα με κάθε παραλήπτη. Για να γίνει κατανοητό το πρόβλημα που προκύπτει από μια τέτοια πρακτική, ας υποθέσουμε ότι θέλουμε να χρησιμοποιήσουμε μια video εφαρμογή μεταξύ μιας ομάδας υπολογιστών που

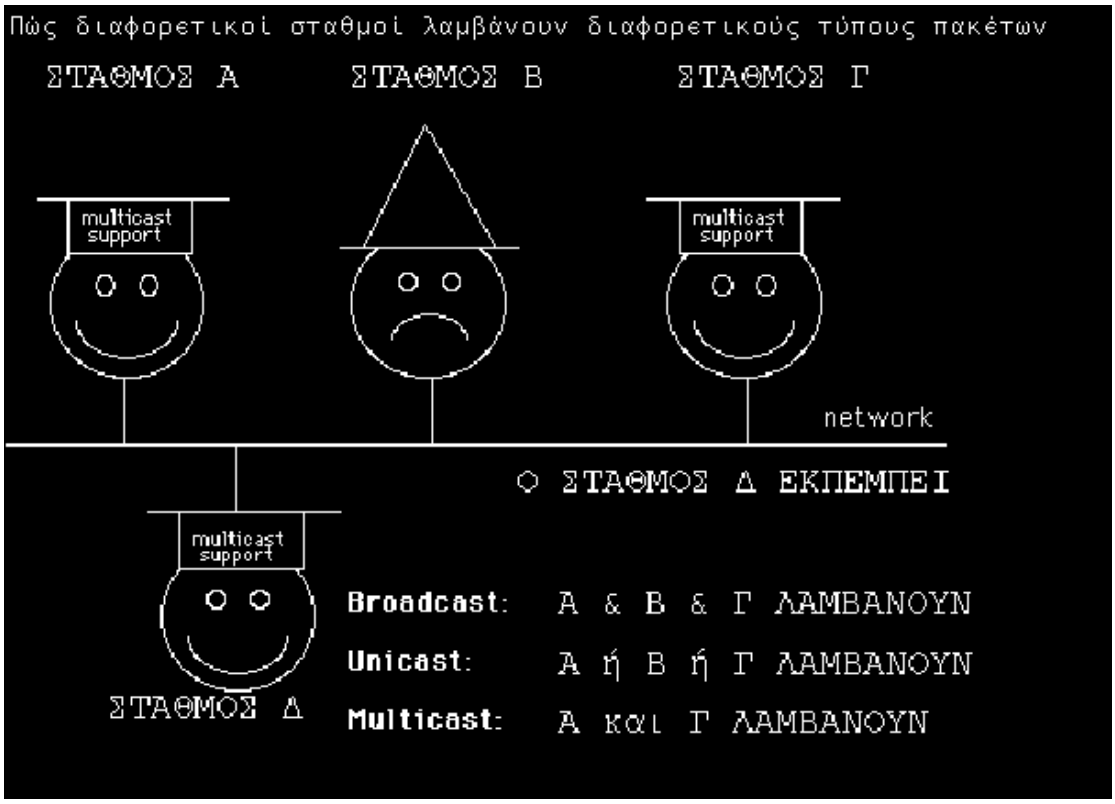
συνδέονται μέσω ενός απλού Ethernet δικτύου. Θεωρώντας ότι κάθε σταθμός στέλνει δεδομένα με ρυθμό 100 kb το δευτερόλεπτο και ότι οι υπολογιστές είναι δικτυωμένοι με σύνδεση 1000 kb το δευτερόλεπτο, συμπεραίνουμε ότι μόνο άλλοι 10 άλλοι σταθμοί μπορούν να τον παρακολουθήσουν. Αλλά και πάλι το πρόβλημα που προκύπτει είναι ότι ο 1ος σταθμός δεν μπορεί να λάβει απαντήσεις από τους υπόλοιπους. Έτσι για να επικοινωνήσουν πλήρως (unicast) 3 υπολογιστικοί σταθμοί (έτσι ώστε ο καθένας να στέλνει στους άλλους 2) απαιτούνται 600 kbps, ενώ αν είναι 4 αντίστοιχα 1200 kbps και 5 2000 kbps.

1.2.2 Broadcast τρόπος μετάδοσης δεδομένων

Ο δεύτερος σημαντικός τρόπος μετάδοσης δεδομένων είναι το broadcast. Ένα broadcast μήνυμα λαμβάνεται από κάθε μέλος ενός δικτύου, ενώ εκπέμπεται μια μόνο φορά από την πηγή του. Παρόλο που κάτι τέτοιο φαίνεται μια ικανοποιητική εναλλακτική λύση σε σχέση με το unicast, το broadcast παρουσιάζει τα δικά του προβλήματα. Έτσι για παράδειγμα στέλνοντας κατ' αυτόν τον τρόπο δεδομένα σε ένα δίκτυο υποχρεώνουμε το κάθε μέλος του δικτύου να καταναλώσει κάποιον υπολογιστικό χρόνο για να τα συλλέξει και να τα αναλύσει, για να δει αν του είναι χρήσιμα η όχι. Αυτό σημαίνει, για όσα μέλη δεν ενδιαφέρονται για την υπό μετάδοση πληροφορία, απώλεια χρόνου επεξεργασίας δεδομένων και έτσι περιττή καθυστέρηση. Για παράδειγμα, στέλνοντας ένα broadcast μήνυμα είναι σαν να έχουμε ένα κτίριο του οποίου όλα τα τηλέφωνα να είναι συνδεδεμένα στην ίδια γραμμή. Έτσι όταν κάποιος καλεί, υποχρεώνει τον καθένα να απαντήσει στην κλήση για να διαπιστώσει αν το τηλεφώνημα προορίζεται γι' αυτόν ή όχι. Επίσης σχεδόν σε όλα τα δίκτυα απαγορεύεται τα broadcast μηνύματα να μεταδίδονται από ένα υποδίκτυο (subnets) σε άλλο. Αυτό περιορίζει σημαντικά το μέγεθος ενός broadcast δικτύου.

1.2.3 Multicast τρόπος μετάδοσης δεδομένων

Ο συμβιβασμός μεταξύ των δύο παραπάνω σχημάτων μετάδοσης είναι ο multicast τρόπος μετάδοσης δεδομένων. Το multicast επιτρέπει σε ένα απλό μήνυμα να σταλεί σε πολλαπλούς παραλήπτες (υπολογιστές), με τη διαφορά όμως ότι μόνο όσοι από αυτούς είναι «συντονισμένοι» στο συγκεκριμένο κάθε φορά multicast group θα το λάβουν και μόνο αυτοί. Έτσι ένα πακέτο που πρέπει να το λάβουν Ν παραλήπτες (και μόνο αυτοί) μπορεί να σταλεί σαν ένα απλό πακέτο.



(Εικόνα 2: Τρόποι μετάδοσης δεδομένων)

Οι διάφορες δικτυακές τοπολογίες συνήθως ταιριάζουν από τη φύση τους σε συγκεκριμένα μοντέλα προώθησης. Έτσι π.χ. οι οπτικές ίνες και τα καλώδια χαλκού από-σημείο-σε-σημείο τα οποία διασυνδέουν μεταξύ τους δύο μηχανήματα, αποτελούν φυσικά μέσα unicast. Από την άλλη οι κόμβοι μπορούν επίσης να προωθήσουν πακέτα δεδομένων ώστε να δημιουργήσουν τελικά κατανομές multicast ή εκπομπής από μέσα unicast. Παρόμοια, το παραδοσιακό καλώδιο του Ethernet αποτελεί φυσικό μέσο εκπομπής αφού όλοι οι κόμβοι είναι προσκολλημένοι σε ένα

μακρύ καλώδιο και κάθε πακέτο που αποστέλλεται από έναν κόμβο γίνεται αντιληπτό από όλους τους άλλους. Έτσι οι κάρτες δικτύου Ethernet υλοποιούν τη unicast αγνοώντας τα πακέτα που δεν προορίζονται αποκλειστικά γι' αυτούς (δεν φέρουν δηλαδή ως ένδειξη παραλήπτη τη διεύθυνση MAC της συγκεκριμένης κάρτας δικτύου). Ένα ασύρματο δίκτυο είναι εκ φύσεως μέσο multicast, αφού όλοι οι κόμβοι εντός της εμβέλειας του αποστολέα λαμβάνουν τα πακέτα του. Οι ασύρματοι κόμβοι αγνοούν τα πακέτα που απευθύνονται σε άλλες συσκευές, ενώ τα πακέτα απαιτούν προώθηση για να φτάσουν σε έναν κόμβο εκτός της εμβέλειας του αποστολέα (Ad hoc δίκτυο).

Σε κόμβο με πολλαπλές διαθέσιμες εξερχόμενες συνδέσεις, η απόφαση για το ποια θα χρησιμοποιηθεί για προώθηση απαιτεί έναν αλγόριθμο ο οποίος μπορεί να χαρακτηρίζεται από μεγάλη υπολογιστική πολυπλοκότητα. Αφού μία απόφαση προώθησης πρέπει να ληφθεί για κάθε εισερχόμενο πακέτο το οποίο δεν απευθύνεται αποκλειστικά στον τρέχοντα κόμβο, αυτό το γεγονός μπορεί να περιορίσει σημαντικά την απόδοση και τη διαμεταγωγή του δικτύου. Οι πιο γρήγοροι δρομολογητές αξιοποιούν ειδικούς αλγορίθμους για να προωθούν γρήγορα μεγάλο πλήθος πακέτων.

Η απόφαση προώθησης λαμβάνεται συνήθως με έναν από δύο τρόπους: δρομολόγηση, η οποία χρησιμοποιεί πληροφορίες κωδικοποιημένες στη διεύθυνση μίας συσκευής για να υπολογίσει τη θέση της στο δίκτυο, ή γεφύρωση, η οποία δεν υπολογίζει την τοπολογική θέση που αντιστοιχεί σε κάθε διεύθυνση και βασίζεται σε εκπομπή για να εντοπίσει άγνωστες διευθύνσεις. Η μεγάλη επιβάρυνση στην οποία οδηγεί η εκπομπή έχει οδηγήσει σε επικράτηση της δρομολόγησης σε μεγάλα δίκτυα όπως το Διαδίκτυο. Επειδή όμως τα μεγάλα δίκτυα συνήθως αποτελούνται από

μικρότερα υποδίκτυα διασυνδεδεμένα μεταξύ τους, στην πραγματικότητα η γεφύρωση χρησιμοποιείται στο Internet αλλά σε τοπικό μόνο επίπεδο.

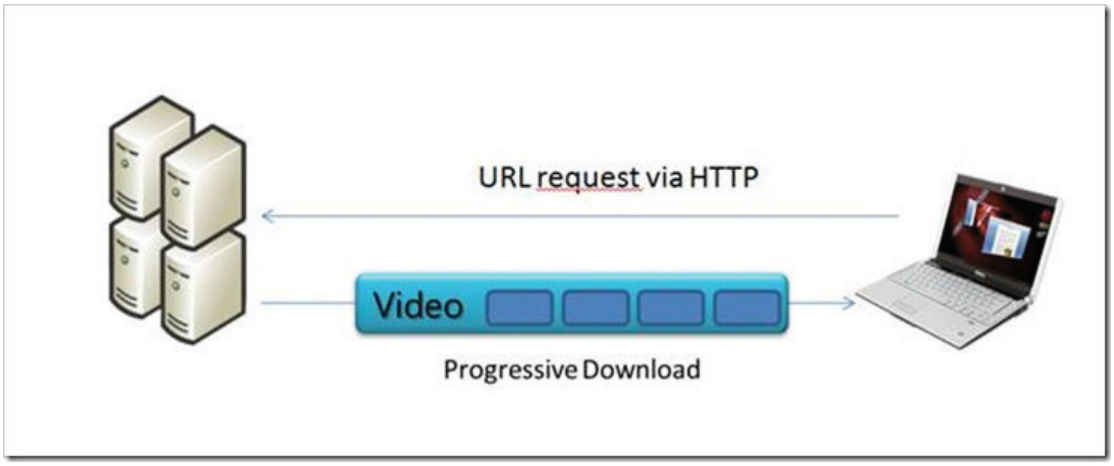
Στα παράλληλα και κατανεμημένα συστήματα και στον παράλληλο προγραμματισμό η εκπομπή και η πολυδιανομή αποτελούν μόνο δύο τύπους συλλογικής επικοινωνίας, όπου στο πλαίσιο ενός κατανεμημένου υπολογισμού μία ή περισσότερες διεργασίες πρέπει να αποστείλουν δεδομένα σε μία ή περισσότερες άλλες διεργασίες. Κάθε διεργασία εκτελείται σε διαφορετικό υπολογιστή και η αποστολή γίνεται μέσω του υποκείμενου δικτύου διασύνδεσης, το οποίο μπορεί να είναι (αλλά όχι απαραίτητα) ενα σύνηθες δίκτυο υπολογιστών. Άλλοι τύποι συλλογικών επικοινωνιών είναι η διασκόρπιση (scatter), όπου ο αποστολέας πρέπει να αποστείλει διαφορετικό μήνυμα σε καθεμία από τις άλλες διεργασίες που συμμετέχουν στον υπολογισμό, η συλλογή (gather), όπου ένας παραλήπτης πρέπει να συλλέξει ένα διαφορετικό μήνυμα από όλες τις άλλες διεργασίες, η πολλαπλή εκπομπή (all to all broadcast), όπου όλες οι διεργασίες εκτελούν εκπομπή, και η ολική ανταλλαγή (total exchange), όπου όλες οι διεργασίες εκτελούν διασκόρπιση ή συλλογή.

1.3 Μέθοδοι online αναπαραγωγής βίντεο

1.3.1 Progressive Download

Η μέθοδος Progressive Download αφορά το ολοκληρωτικό κατέβασμα ενός απλού αρχείου βίντεο με την διαφορά ότι το βίντεο μπορεί να αναπαραχθεί κανονικά μέχρι το ολοκληρωμένο σημείο που έχει κατεβεί την στιγμή που συνεχίζει να γίνεται ταυτόχρονα η υπολειπόμενη λήψη. Τα μειονεκτήματα είναι η μεγάλη σπατάλη του διαθέσιμου εύρους ζώνης π.χ. την στιγμή που ο χρήστης δει μόνο ένα λεπτό από ένα δεκάλεπτο βίντεο και θα έχει κατεβεί ήδη ολόκληρο το βίντεο. Επιπλέον δεν

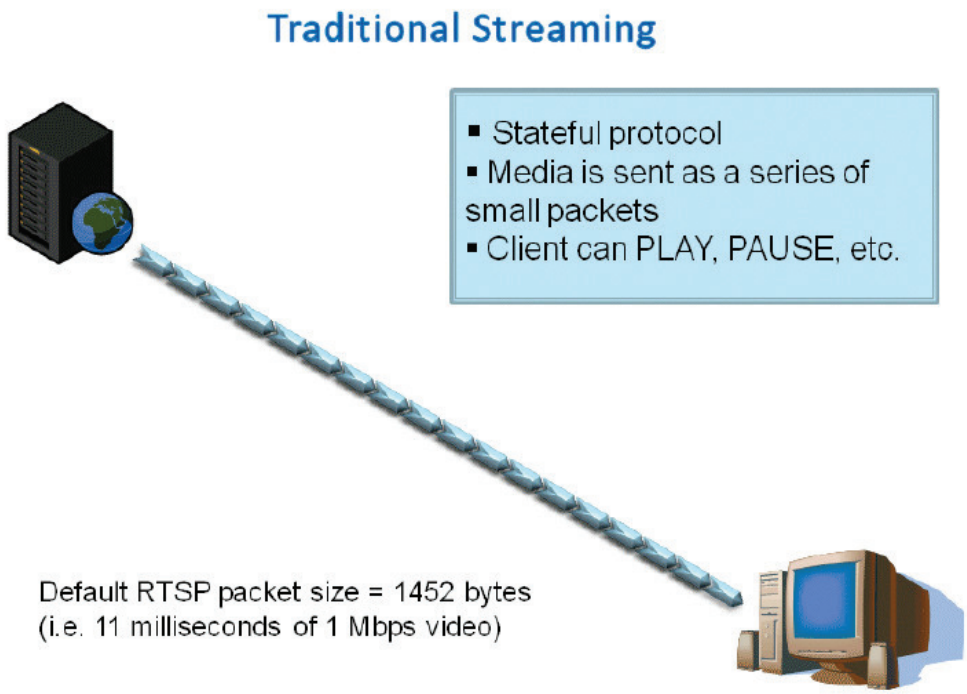
προσαρμόζεται δυναμικά στις αλλαγές του δικτύου αλλά και στις συνθήκες κάθε χρήστη και τέλος δεν υποστηρίζει live streaming.



(Εικόνα 3: Progressive Download)

1.3.2 Stateful streaming

Θεωρείται μία παραδοσιακή μέθοδος streaming με το γνωστό RTSP πρωτόκολλο (Real-Time Streaming Protocol) δηλωμένο ως Stateful πρωτόκολλο που σημαίνει ότι δημιουργείται μια μόνιμη σύνδεση μεταξύ τελικού χρήστη και server κατά την διάρκεια streaming αναπαραγωγής βίντεο οπού ο server ελέγχει συνέχεια την κατάσταση του χρήστη μέχρι τη στιγμή που θα κάνει αποσύνδεση. Κατά την διάρκεια της σύνδεσης η αποστολή του βίντεο γίνεται από τον server σαν μία σταθερή ροή από μικρά πακέτα γνωστά σαν RTP πακέτα πάνω από μεταφορά UDP ή TCP για λόγους ταχύτητας ή επιτρεπόμενης κίνησης στα τείχη προστασίας αντίστοιχα. Ο χρήστης επικοινωνεί με το server με εντολές όπως PLAY, PAUSE κ.τ.λ.. Τα μειονεκτήματα είναι ότι απαιτεί την ύπαρξη ειδικών εξειδικευμένων server και γρήγορο κατάλληλο υλικό. Επιπλέον χρειάζεται και τα ειδικά κατάλληλα πρωτόκολλα μαζί με τα ειδικά δίκτυα διανομής περιεχομένου και την αντίστοιχη υποδομή. Τέλος το Stateful streaming είναι πολύ εύθραυστο, ένα μικρό σφάλμα στη σύνδεση και η αναπαραγωγή έχει διακοπεί.



(Εικόνα 4: Stateful streaming)

1.3.3 Adaptive bitrate streaming

1.3.3.1 Εισαγωγή

Το adaptive bitrate streaming είναι μια τεχνική που χρησιμοποιείται για streaming πολυμέσων μέσω δικτύων Η/Υ. Ενώ στο παρελθόν οι περισσότερες τεχνολογίες video streaming χρησιμοποιούσαν πρωτόκολλα streaming όπως RTP με RTSP, η σημερινή τεχνολογία adaptive streaming είναι σχεδόν αποκλειστικά βασισμένη σε http το οποίο θεωρείται και stateless πρωτόκολλο γιατί δεν χρειάζεται μόνιμη σύνδεση αλλά στιγμιαίες αιτήσεις-συνεδρίες και έχει σχεδιαστεί για να λειτουργεί αποτελεσματικά σε μεγάλα κατανεμημένα δίκτυα HTTP, όπως το Internet.

Λειτουργεί ανιχνεύοντας το εύρος ζώνης του χρήστη και τις ικανότητες της CPU σε πραγματικό χρόνο προσαρμόζοντας την ποιότητα της ροής βίντεο αναλόγως. Απαιτεί τη χρήση ενός κωδικοποιητή που μπορεί να κωδικοποιήσει ένα βίντεο σε πολλαπλούς ρυθμούς μετάδοσης. Η αναπαραγωγή του χρήστη εναλλάσσει ανάμεσα στις διάφορες κωδικοποιήσεις, ανάλογα με τους διαθέσιμους πόρους. Το αποτέλεσμα: πολύ λίγο

buffering, γρήγορος χρόνος εκκίνησης και μια καλή εμπειρία για high-end και low-end συνδέσεις.

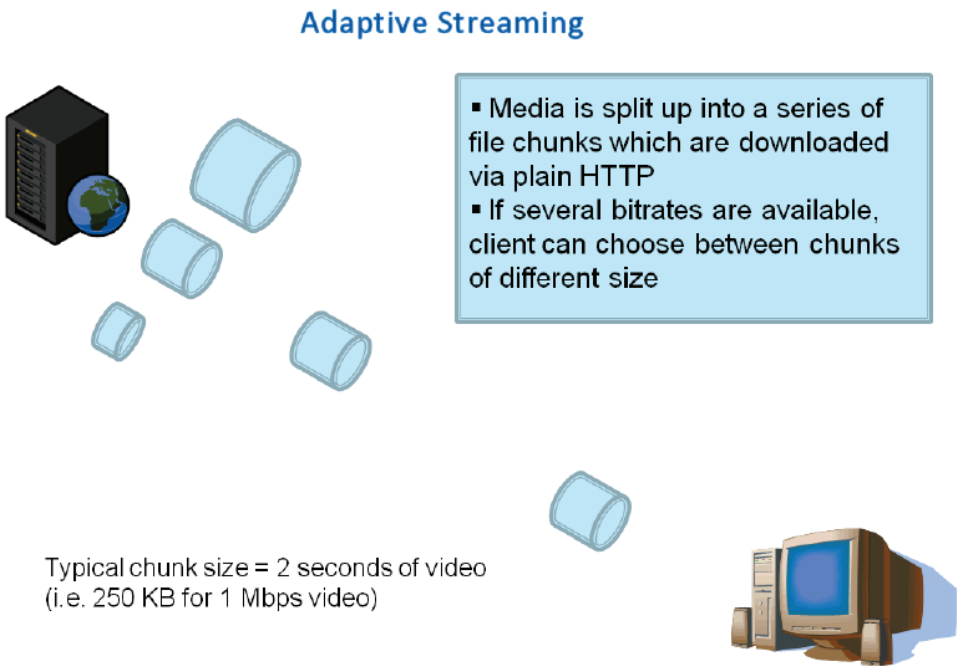
1.3.3.2 Τρέχουσα χρήση του adaptive bitrate streaming

Εταιρίες παραγωγής ήχου και εικόνας, δίκτυα διανομής περιεχομένου και studios χρησιμοποιούν την τεχνολογία adaptive bitrate streaming, ώστε να παρέχουν στους καταναλωτές την υψηλότερη ποιότητα εικόνας με χρήση λιγότερου ανθρώπινου δυναμικού και λιγότερους πόρους. Όταν όλα έχουν συμφωνηθεί και γίνει, η δημιουργία πολλαπλών εξόδων βίντεο, ιδιαίτερα, για το adaptive bitrate streaming, προσθέτει μεγάλη αξία στους καταναλωτές. Αν η τεχνολογία λειτουργεί όπως σχεδιάστηκε, ο τελικός χρήστης ή ο καταναλωτής θα πρέπει να την αγνοεί εντελώς. Ως εκ τούτου, αν και εταιρίες μέσων ενημέρωσης χρησιμοποιούν ενεργά την τεχνολογία adaptive bitrate streaming για πολλά χρόνια τώρα και έχει ουσιαστικά μετατραπεί μία συνηθισμένη πρακτική για τους high-end παρόχους streaming, γενικά οι καταναλωτές είναι σχετικά ανίδεοι για την αναγκαιότητά της.

1.3.3.3 Πλεονεκτήματα του adaptive bitrate streaming

Οι χρήστες των media streaming έρχονται αντιμέτωποι με την υψηλότερη ποιότητα όταν το adaptive bitrate streaming χρησιμοποιείται επειδή το δίκτυο του χρήστη και οι συνθήκες αναπαραγωγής αυτόματα προσαρμόζονται στις ανά πάσα στιγμή μεταβαλλόμενες συνθήκες.

Τα μέσα μαζικής ενημέρωσης και η βιομηχανία της ψυχαγωγίας είναι οι βασικοί κυρίαρχοι του adaptive bitrate streaming. Δεδομένου ότι ο χώρος του βίντεο αυξάνεται εκθετικά, τα δίκτυα διανομής περιεχομένου και οι πάροχοι του βίντεο μπορούν να παρέχουν στους πελάτες μια ανώτερη εμπειρία προβολής. Το adaptive bitrate streaming απαιτεί λιγότερη κωδικοποίηση απλοποιώντας γενικά την επεξεργασία και δημιουργεί καλύτερα αποτελέσματα.



(Εικόνα 5: adaptive bitrate streaming)

1.4 Προκλήσεις και Quality of Experience (QoE)

1.4.1 Δίκτυο σύνδεσης και τρέχουσα ασφάλεια

Κατά την προσπάθεια σύνδεσης με ένα μη-ελεγχόμενο δίκτυο, τα routers, τα τείχη προστασίας και ποιες θύρες είναι ανοιχτές είναι άγνωστα. Σε ένα οικιακό δίκτυο μπορεί να υπάρχουν προσωπικά τείχη προστασίας, πιθανοί routers και λογισμικό ασφαλείας που σαρώνουν την κάθε δραστηριότητα του συστήματος. Σε ένα WiFi hot spot η πρόσβαση μπορεί να είναι εξαιρετικά περιορισμένη λόγω ανησυχιών για την ασφάλεια.

Αυτό είναι ένα γνωστό εμπόδιο με τις εφαρμογές του δικτύου και μπορεί να ξεπεραστεί χρησιμοποιώντας το πρωτόκολλο HTTP για την επικοινωνία. Το HTTP χρησιμοποιεί τη θύρα 80 για αιτήσεις. Οι αιτήσεις προς αυτή την θύρα είναι οι πιο γνωστές ώστε να επιτραπούν από οποιοδήποτε τείχος προστασίας ή router που χρησιμοποιούνται για όλο το web surfing.

1.4.2 Έλεγχος εύρους ζώνης

Το εύρος ζώνης είναι ένα σημαντικό ζήτημα. Αν ένας χρήστης παρακολουθεί ένα βίντεο και κάποιος άλλος στο ίδιο δίκτυο ξαφνικά αποφασίζει να εκτελέσει μια μεταφορά αρχείων, το διαθέσιμο εύρος ζώνης για το βίντεο μπορεί να έχει σοβαρές συνέπειες. Προκειμένου να διατηρηθεί μια καλή ποιότητα εμπειρίας(QOE), το περιεχόμενο πρέπει συνεπώς να έχει κωδικοποιηθεί σε διάφορες ποιότητες και το πρωτόκολλο μεταφοράς θα πρέπει να είναι σε θέση να αλλάξει δυναμικά την ποιότητα χωρίς διακοπή της αναπαραγωγής ή ενέργεια από το χρήστη.

1.4.3 Πολλαπλοί χρήστες και πολλαπλές αναλύσεις

Καθώς όλο και περισσότερες συσκευές συνδεδεμένες στο Διαδίκτυο εμφανίζονται στην αγορά κάθε μήνα, με όλο και μεγαλύτερες δυνατότητες για περιήγηση και αναπαραγωγή βίντεο, ο αριθμός των αναλύσεων και ρυθμών μετάδοσης που απαιτούνται για την υποστήριξη αυτών των συσκευών αυξάνεται εκθετικά. Συνήθως πολλές συσκευές επικοινωνούσαν μέσω ειδικού πρωτόκολλου. Δεν είναι κατάλληλο να υπάρχουν ξεχωριστοί κωδικοποιητές, ξεχωριστά συστήματα και ειδικοί server για κάθε συσκευή που πρέπει να υποστηριχθεί.

1.4.4 Ασφάλεια περιεχομένου

Η παραδοσιακή μέθοδος μετάδοσης που περιορίζει την πρόσβαση σε ζωντανό περιεχόμενο επιτυγχάνεται με την κρυπτογράφηση μετάδοσης της ροής. Αυτή η μέθοδος έχει εφαρμοστεί με επιτυχία στην μετάδοση του ζωντανού βίντεο μέσω του Διαδικτύου: το περιεχόμενο είναι κρυπτογραφημένο μεταξύ του κωδικοποιητή και του server και στη συνέχεια και πάλι μεταξύ του server και του χρήστη. Ωστόσο, ο κίνδυνος με αυτή την προσέγγιση είναι ότι το περιεχόμενο είναι προσωρινά γραμμένο στην μνήμη του server πριν την άμεση αναμετάδοση και μετά πάλι εγγράφεται στη μνήμη του χρήστη πριν από την άμεση παρουσίαση και διαγραφή του. Κατά τη

διάρκεια αυτών των σύντομων χρονικών περιόδων, το περιεχόμενο είναι εκτεθειμένο και αυτό αποτελεί κίνδυνο για την ασφάλεια. Επίσης το περιεχόμενο θα πρέπει να αποθηκεύεται σε servers του διαδικτύου ή σε συσκευές χρηστών και πρέπει να εξακολουθεί να προστατεύεται για την αποφυγή της πειρατείας.

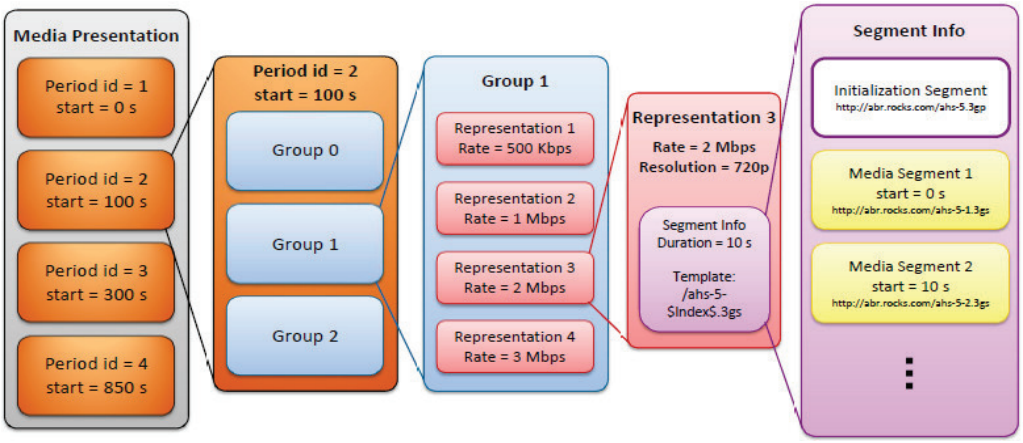
2. Dynamic Adaptive Streaming over HTTP (DASH).

2.1 Εισαγωγή.

Το Dynamic Adaptive Streaming over HTTP (DASH) είναι μία multimedia streaming τεχνολογία της οποίας η υλοποίηση βασίζεται στο MPEG. Οι εργασίες πάνω στο DASH ξεκίνησαν το 2010 και τον Ιανουάριο του 2011 ορίστηκε ως σχέδιο για ένα Διεθνές Πρότυπο το οποίο αναμένεται να έχει υλοποιηθεί μέχρι το Νοέμβριο του 2011.

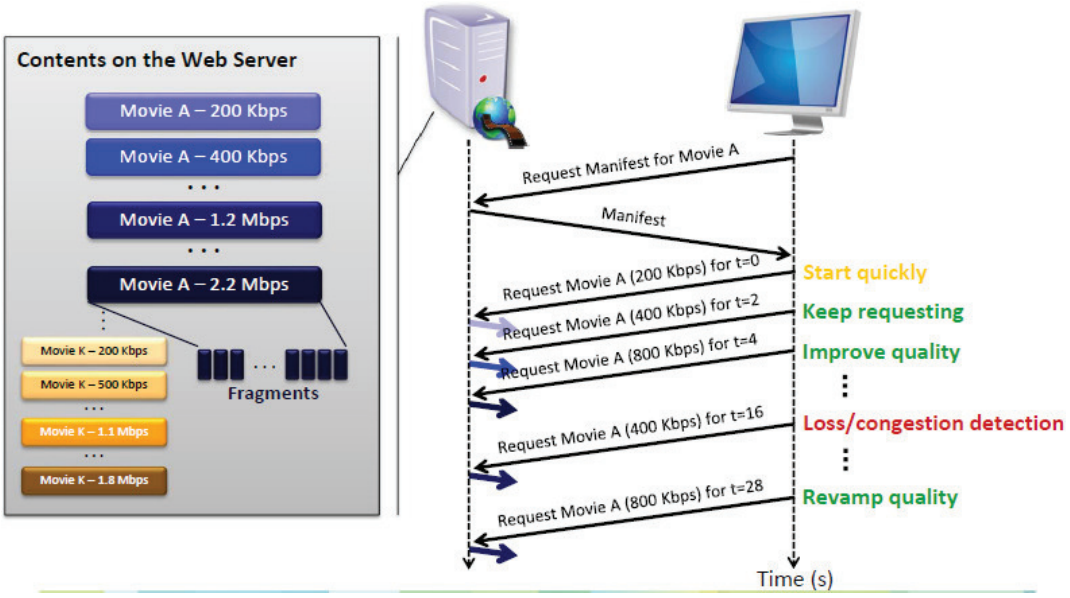
2.2 Λειτουργία.

Το DASH είναι μία multimedia streaming τεχνολογία όπου ένα βίντεο είναι αποθηκεμένο σε ένα server χωρισμένο σε ένα ή περισσότερα τμήματα διαφορετικής ποιότητας και ανάλυσης και στέλνεται σε έναν client μέσω HTTP. Ένα ειδικό αρχείο μορφής xml περιγράφει τη δομή και το περιεχόμενο παρουσίασης του βίντεο π.χ. το media presentation description (MPD) στην εικόνα 6 που περιέχει διάφορες πληροφορίες όπως χρονοδιάγραμμα, χαρακτηριστικά όπως αναλύσεις και ρυθμοί μετάδοσης, URL διευθύνσεις κ.τ.λ.. Κατά την διάρκεια αναπαραγωγής του βίντεο σε τμήματα μπορεί να αλλάξει η ανάλυση και η ποιότητα του βίντεο αναλόγως τις συνθήκες του δικτύου, τις δυνατότητες του συστήματος και τις προτιμήσεις του χρήστη επιτρέποντας το adaptive bitrate streaming.

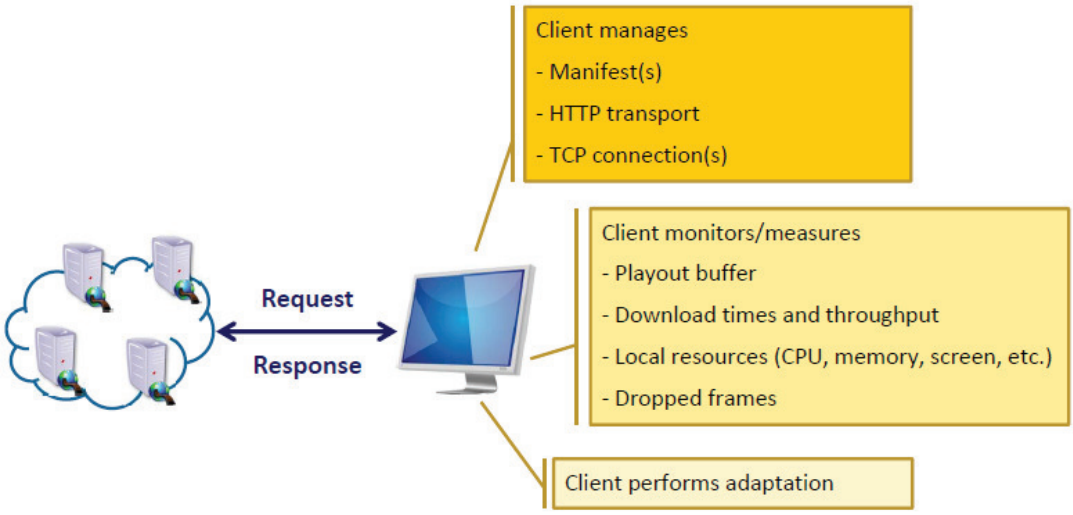


(Εικόνα 6: MPD αρχιτεκτονική)

Συγκεκριμένα η λειτουργία του DASH ξεκινάει με τον client να ζητάει από τον server το MPD και αφού το παραλάβει τότε ο client είναι αυτός που κάνει τις απαραίτητες μετρήσεις και υπολογισμούς όπως throughput, εύρος ζώνης, ικανότητες υλικού κ.τ.λ. και ύστερα παίρνει αποφάσεις για την βέλτιστη ικανή αναπαραγωγή του βίντεο όπως φαίνεται και στις εικόνες 7 και 8.



(Εικόνα 7: Λειτουργία του DASH)



(Εικόνα 8: Η θέση του Client στο DASH)

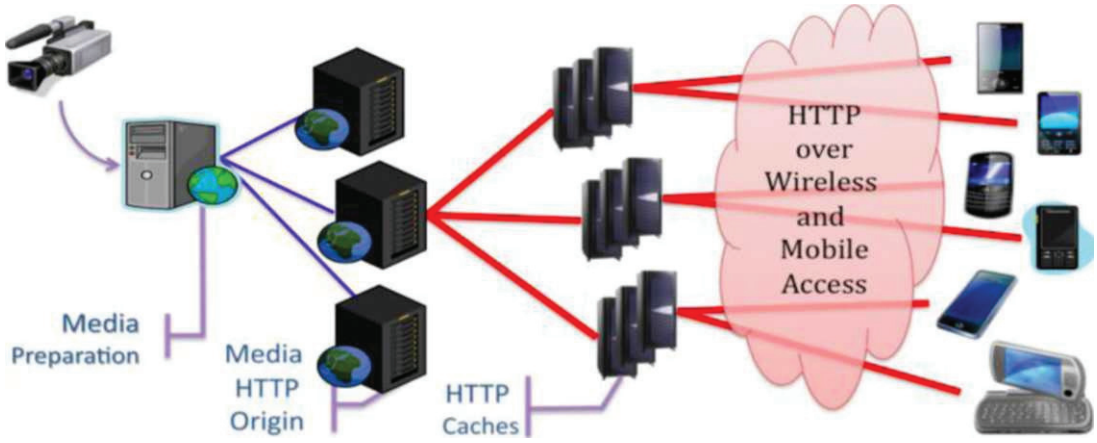
2.3 Πλεονεκτήματα και σύγκριση του DASH

Το παραδοσιακό streaming χρησιμοποιεί κάποιο είδος stateful πρωτοκόλλου όπως για παράδειγμα το Real-Time Streaming Protocol (RTSP). Όταν ο client συνδεθεί με ένα streaming server, ο server έχει συνεχή γνώση της κατάστασης του client, μέχρι ο χρήστης να αποσυνδεθεί. Συνήθως, υπάρχει συχνή επικοινωνία μεταξύ του server και του client κατά τη διάρκεια της μεταξύ τους σύνδεσης. Όταν εγκαθιδρυθεί μια τέτοια σύνοδος, ο server στέλνει το περιεχόμενο, ως ένα συνεχές stream από πακέτα πάνω από τα πρωτόκολλα μεταφοράς TCP ή UDP. Αντιθέτως το πρωτόκολλο HTTP είναι stateless. Όταν γίνεται ένα αίτημα HTTP, ο server αποστέλλει στο αντίστοιχο τερματικό τα δεδομένα που ζήτησε και στη συνέχεια τερματίζεται η συναλλαγή. Δηλαδή, κάθε αίτημα HTTP αντιμετωπίζεται ως μια μοναδική και ανεξάρτητη συναλλαγή.

Εναλλακτικά με το streaming, μπορεί να χρησιμοποιηθεί μια τεχνική που ονομάζεται προοδευτική λήψη (progressive download) από τους συνήθεις HTTP Web servers. Εφόσον ο client υποστηρίζει το πρωτόκολλο HTTP/1.1, είναι δυνατό να αποστείλει αιτήματα HTTP στον εξυπηρετητή Web server(πρέπει να υποστηρίζει και ο Web server το HTTP/1.1), τα οποία να ζητούν συγκεκριμένες ομάδες από Bytes ενός πολυμεσικού αρχείου. Έτσι, είναι δυνατό να λάβει ο χρήστης οποιοδήποτε τμήμα του βίντεο επιθυμεί. Τα κύρια μειονεκτήματα της προοδευτικής λήψης είναι 1)σπατάλη πόρων σε περίπτωση που ο χρήστης αποφασίσει να σταματήσει να παρακολουθεί το ληφθέν περιεχόμενο αφότου έχει ξεκινήσει η προοδευτική λήψη (για παράδειγμα αν αλλάξει κανάλι), 2) δεν είναι δυνατό να προσαρμόσει το bitrate adaptive και 3) δεν υποστηρίζει πολυμεσικές υπηρεσίες πραγματικού χρόνου.

Το DASH δύναται να αντιμετωπίσει όλα τα παραπάνω προβλήματα και περιορισμούς. Επιπλέον, στηρίζεται στο ήδη ιδιαίτερα δημοφιλές πρωτόκολλο HTTP και έχει αρκετά κοινά στοιχεία με την προοδευτική λήψη πάνω από HTTP, με

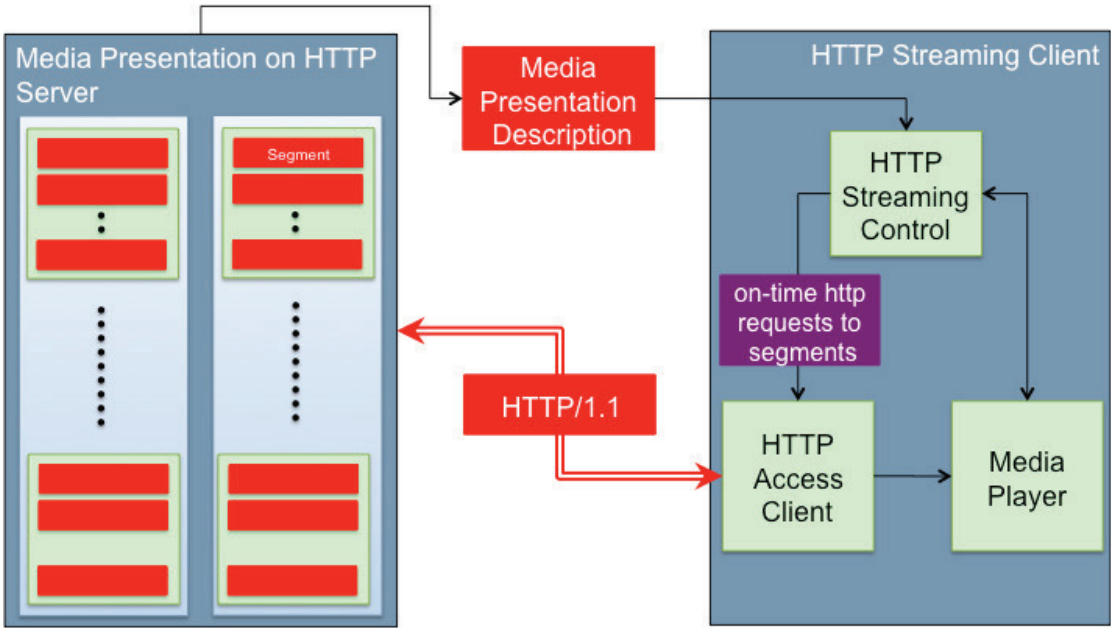
συνέπεια να μην είναι αναγκαία η αλλαγή εξοπλισμού από χρήστες/παρόχους για την λειτουργία του.



(Εικόνα 9: Παράδειγμα αρχιτεκτονικής DASH)

Στην εικόνα 9 φαίνεται μια πιθανή αρχιτεκτονική μετάδοσης πολυμέσων για streaming πάνω από HTTP. Κατά το στάδιο της προετοιμασίας του πολυμεσικού περιεχομένου, δημιουργούνται τεμάχια τα οποία περιέχουν διαφορετικές εκδόσεις ενός ή όλων των μερών του πολυμεσικού περιεχομένου, κάθε μια από τις οποίες έχει διαφορετική κωδικοποίηση. Κατόπιν, τα τεμάχια μαζί με τα αντίστοιχα αρχεία που περιγράφουν το περιεχόμενό τους (media presentation description -MPD). αποστέλλονται σε ένα ή περισσότερους servers περιεχομένου (media origin servers) . Αυτοί οι servers είναι κατά προτίμηση http servers, ώστε το σύνολο της επικοινωνίας με αυτούς να γίνεται πάνω από HTTP (οι έντονες γραμμές στην εικόνα συμβολίζουν HTTP επικοινωνία). Επιπλέον, μπορεί να χρησιμοποιηθεί η ήδη υπάρχουσα υποδομή σε HTTP caches και proxies, ώστε να μειωθεί ο φόρτος των servers πηγής (όπως φαίνεται και στην εικόνα). Ο client στηριζόμενος στα MPD αρχεία που περιγράφουν την σχέση που έχουν μεταξύ τους τα τεμάχια και πως αυτά σχηματίζουν στο σχετικό πολυμεσικό περιεχόμενο, αποστέλλει αιτήματα HTTP GET στους αντίστοιχους

servers, ζητώντας τα σχετικά τεμάχια. Ο client έχει πλήρη έλεγχο της συνόδου, πχ είναι υπεύθυνο να κάνει τα σχετικά αιτήματα "στην ώρα τους" (on-time request). Επίσης φροντίζει για την ομαλή αναπαραγωγή των τεμαχίων που λαμβάνει, μέσω της προσαρμογής του bitrate και άλλων ιδιοτήτων. Η ανάγκη για αυτές τις αλλαγές μπορεί να προκύψει επειδή άλλαξε η κατάσταση στην οποία βρίσκεται η συσκευή ή επειδή υπήρξε μεταβολή στις προτιμήσεις του χρήστη.



(Εικόνα 10: Adaptive streaming πάνω από πρωτόκολλο HTTP 1.1)

Συνοπτικά τα πλεονεκτήματα του Adaptive Streaming πάνω από HTTP (DASH) είναι:

- Μπορεί να χρησιμοποιηθεί η ήδη υπάρχουσα υποδομή (HTTP server κλπ.) ή σχετικά (με άλλες υλοποιήσεις φθηνή) υποδομή.
- Χρησιμοποιεί τη θύρα 80, η οποία είναι σχεδόν πάντα ανοιχτή στους δρομολογητές, οπότε δεν τίθεται θέμα firewall ή NAT.
- Το HTTP streaming διαδίδεται ταχύτατα ως τρόπος μετάδοσης βίντεο πάνω από το διαδίκτυο.

- Το HTTP θεωρείται ιδιαίτερα αξιόπιστο και απλό, επειδή τόσο το HTTP, όσο και τα πρωτόκολλα μεταφοράς και δικτύου που χρησιμοποιεί (TCP/IP) είναι ευρέως διαδεδομένα.
- Ο έλεγχος της συνόδου (streaming session) δίνεται εξολοκλήρου στο χρήστη. Το τερματικό εγκαθιδρύει μια ή περισσότερες συνδέσεις TCP σε έναν ή περισσότερους servers ή cache HTTP.
- Δίνεται η ευκαιρία στο τερματικό να επιλέξει αυτόματα το bitrate που είναι περισσότερο κατάλληλο για την σύνδεσή του, χωρίς την ανάγκη επικοινωνίας με κάποιο stream server.
- Δίνεται η δυνατότητα απρόσκοπτης και δυναμικής αλλαγής του bitrate ενός περιεχομένου ή μιας υπηρεσίας με βάση τις συνθήκες που επικρατούν στο δίκτυο και τις ανάγκες του χρήστη, χωρίς την ανάγκη επικοινωνίας με κάποιο stream server.
- Έχει τη δυνατότητα να επιταχύνει τη σύγκλιση στις προσφερόμενες υπηρεσίες streaming σε σταθερά και κινητά τερματικά, καθώς μπορεί να χρησιμοποιηθεί ως μια κοινή πλατφόρμα διανομής.

Αυτή τη στιγμή το DASH υποστηρίζει πολλαπλές υπηρεσίες. Μεταξύ άλλων υποστηρίζει:

- streaming κατά απαίτηση. (On-demand streaming)
- παραδοσιακή τηλεόραση (δηλαδή τηλεόραση στην οποία προβάλλεται ένα συγκεκριμένο πρόγραμμα στο οποίο δεν υπάρχει καμία δυνατότητα να παρέμβει ο χρήστης)/πολυμεσική ευρυεκπομή σε πραγματικό χρόνο
- θέαση με δυνατότητα χρονικής ολίσθησης (Time-shift viewing), με δυνατότητες εγγραφής βίντεο (Personal Video Recording-PVR)

Ενώ κάποιες από τις υπηρεσίες που προβλέπεται να υποστηρίζει το DASH στο μέλλον είναι:

- επιπλέον λειτουργίες αναπαραγωγής, όπως fast forward και rewind
- απλή ενσωμάτωση διαφημίσεων ή άλλου περιεχομένου στις υπηρεσίες on demand και live streaming
- αποδοτική μετάδοση πολλαπλών γλωσσών και ηχητικών κομματιών
- προστασία περιεχομένου και ασφάλεια μετάδοσης

3. Γνωστές υλοποιήσεις του DASH.

3.1 Adobe Systems's HTTP Dynamic Streaming

Η Adobe μετά την έκδοση 10.1 του Flash Player διαθέτει πλέον την δυνατότητα του DASH χρησιμοποιώντας κοινούς HTTP servers και συσκευές δικτύου και εφαρμόζοντας την MP4 διαμόρφωση κατακερματισμού(F4F). Υποστηρίζει ζωντανή η κατά απαίτηση μετάδοση πολυμεσικού περιεχομένου προσαρμοσμένη στην ταχύτητα σύνδεσης δικτύου και επιδόσεων της συσκευής προβολής του χρήστη χρησιμοποιώντας την ήδη υπάρχουσα υποδομή του HTTP πρωτοκόλλου.

Συνοπτικά περιλαμβάνει:

- Μεγάλη συμβατότητα καθώς το 99% των Η/Υ έχουν Flash Player.
- Χρήση της γνωστής MP4 διαμόρφωσης κατακερματισμού(F4F).
- Προστασία πολυμεσικού περιεχομένου με χρήση του Adobe Flash Access.
- Υποστήριξη με δίκτυα διανομής περιεχομένου CDNs.
- Ελεύθερο ανοιχτό λογισμικό αναπαραγωγής περιεχομένου(Open Source Media Framework—OSMF).
- Εργαλεία για την προετοιμασία και δημιουργία αρχείων.
- Υποστήριξη πολυμεσικών αρχείων τύπου .flv και .f4v.
- Υποστήριξη xml αρχείου δομής περιεχομένου(manifest) τύπου .f4m.

Η λειτουργία του σύμφωνα με το DASH ξεκινάει με την ζήτηση του .f4m από τον χρήστη και έπειτα την αναπαραγωγή του .flv ή .f4v προσαρμοσμένη στις συνθήκες δικτύου και συστήματος προβολής του χρήστη όπως φαίνεται στην εικόνα 11.

(Εικόνα 11: Αρχιτεκτονική του Adobe Systems's HTTP Dynamic Streaming)

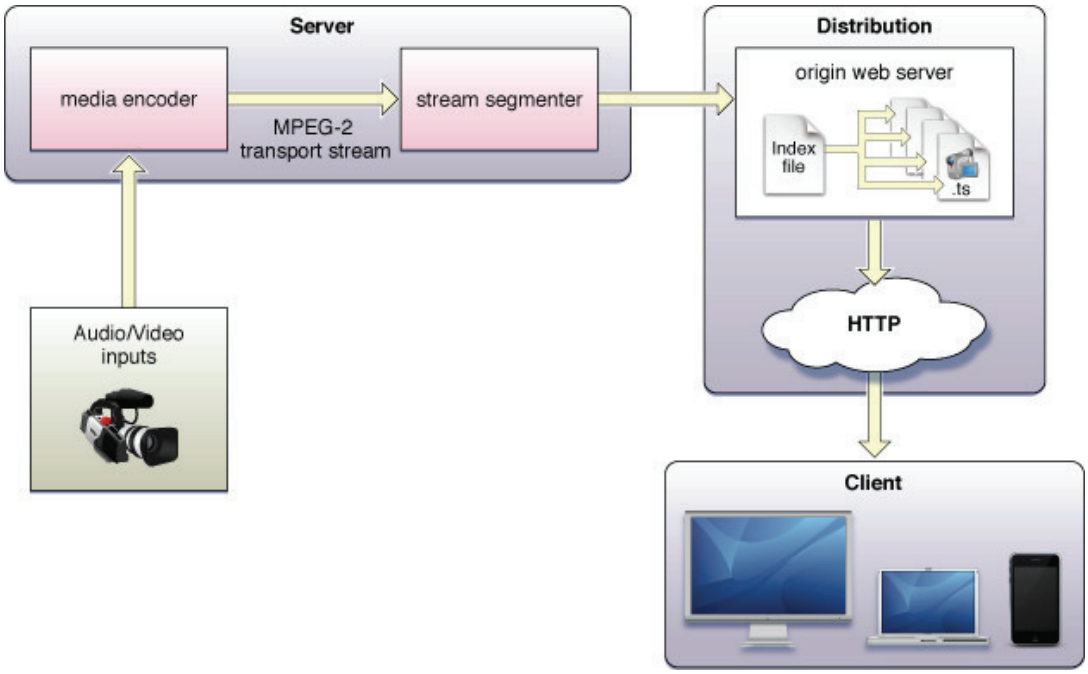
3.2 Apple HTTP Live Streaming

Το HTTP Live Streaming αποτελείται από τρία μέρη: τον server, το σύστημα διανομής και τον client.

- Ο server προετοιμάζει το πολυμεσικό περιεχόμενο ώστε να κωδικοποιηθεί και να μετατραπεί κατάλληλα σε αρχείο έτοιμο προς μετάδοση από το σύστημα διανομής.
- Το σύστημα διανομής περιλαμβάνει web servers που περιμένουν τις αιτήσεις των χρηστών και έπειτα μεταφέρουν το έτοιμο διαμορφωμένο περιεχόμενο σε αυτούς.
- Ο client επιλέγει το περιεχόμενο προς αναπαραγωγή και εκτελεί τις απαραίτητες λειτουργίες για την ανάγνωση και τελική διαμόρφωση του ώστε να γίνει αναγνωρίσιμο από το τερματικό σύστημα. Η εφαρμογή βρίσκεται από τις εκδόσεις 3.0 του iOS και 4.0 του Safari.

Η κωδικοποίηση του πολυμεσικού περιεχομένου είναι σε MPEG-4 (H.264 video και AAC audio) και η κατάληξη τους είναι σε .ts. Τα αρχεία δομής περιεχομένου έχουν κατάληξη .M3U8.

Σε μια τυπική λειτουργία ο server μετατρέπει το περιεχόμενο σε μικρά κομμάτια κωδικοποιημένα σε MPEG-4 με κατάληξη .ts και ταυτόχρονα δημιουργεί ένα αρχείο δομής τους και πληροφοριών με κατάληξη .M3U8 τα οποία στέλνει ομαδικά στο σύστημα διανομής όπου οι web servers διαθέτουν ένα URL link με το οποίο ο χρήστης μπορεί να έχει πρόσβαση για αναπαραγωγή του περιεχομένου όπως φαίνεται στην εικόνα 12.



(Εικόνα 12: Αρχιτεκτονική του HTTP Live Streaming)

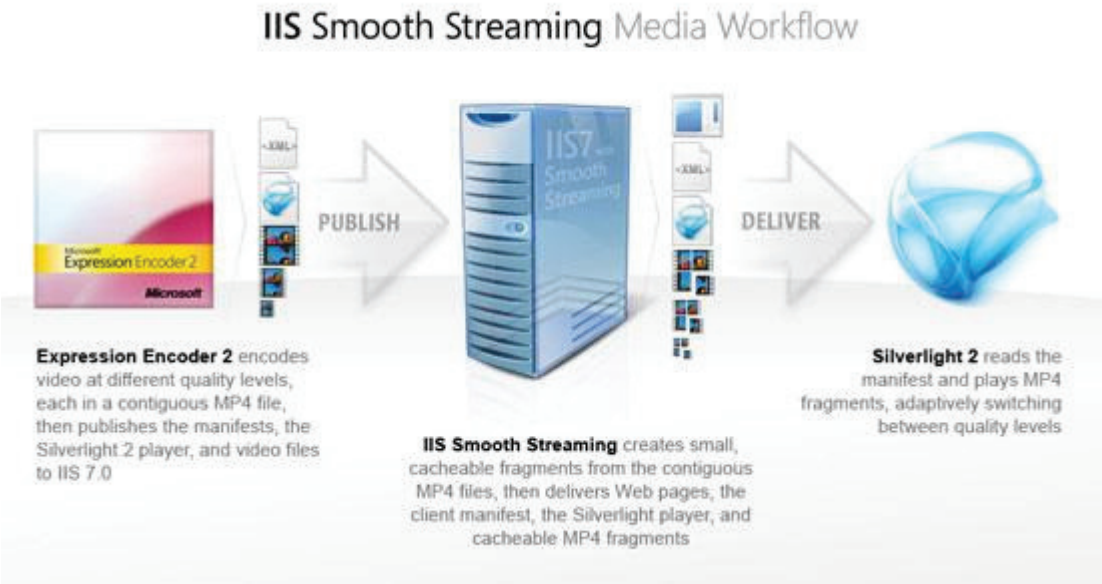
3.3 Microsoft's Smooth Streaming

Από το 2008 με την έκδοση του Internet Information Services (IIS) 7.0 εμφανίστηκε μια νέα ικανότητα τύπου DASH που ονομάστηκε Smooth Streaming. Στόχος ήταν η συνεργασία με το δίκτυο διανομής περιεχομένου Akamai και η ομαλή μετάδοση πολυμεσικού περιεχομένου προσαρμοσμένη δυναμικά στις συνθήκες δικτύου και συστήματος προβολής με σκοπό την καλύτερη δυνατή ποιότητα οπτικής εμπειρίας σε αναλύσεις που έφταναν τεχνολογίες HD. Έχει ήδη χρησιμοποιηθεί και στην ιστοσελίδα του NBC Olympics. Για την διαμόρφωση των αρχείων χρησιμοποιείται το MPEG-4 Part 14 (ISO/IEC 14496-12).

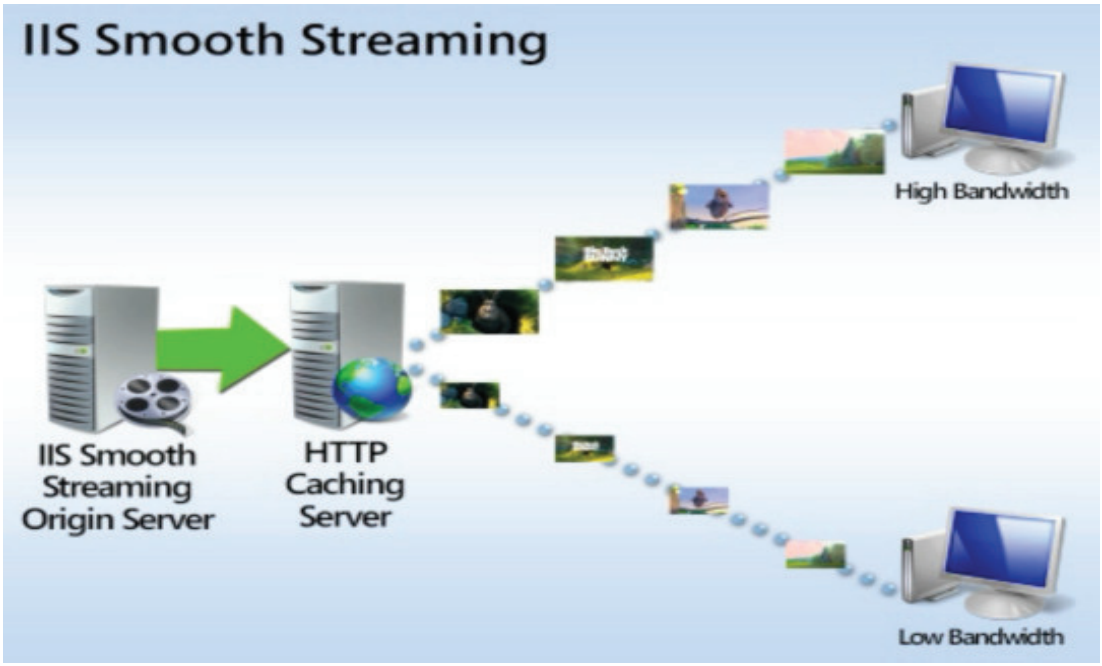
Από την έκδοση 2 του Silverlight επίσης υποστηρίζεται το Smooth Streaming και λειτουργεί κανονικά σύμφωνα με όλα τα χαρακτηριστικά του DASH.

Το Smooth Streaming χρησιμοποιεί αρχεία πολυμεσικού περιεχομένου τύπου .ismv και δύο είδη αρχείων δομής περιεχομένου xml ένα για το Server τύπου .ism που περιγράφει τις σχέσεις μεταξύ των κομματιών και τις θέσεις αποθήκευσης τους και ένα για το Client τύπου .ismc που περιγράφει τις αναλύσεις, τις ποιότητες, τους κωδικοποιητές κ.τ.λ..

Σε μια τυπική λειτουργία ο Client ζητάει το .ismc αρχείο από το Server και ο Server ζητάει από το σκληρό δίσκο του το .ism με σκοπό την δυναμική διαχείριση και προβολή των κομματιών του .ismv αρχείου περιεχομένων. Στις εικόνες 13 και 14 φαίνεται η τεχνολογία του DASH από την πλευρά του Smooth Streaming.



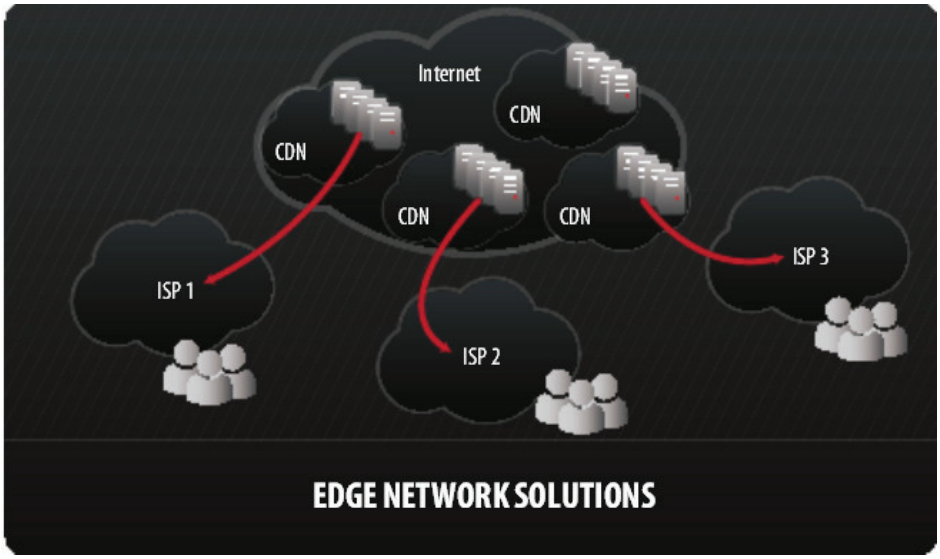
(Εικόνα 13: Αρχιτεκτονική του Smooth Streaming)



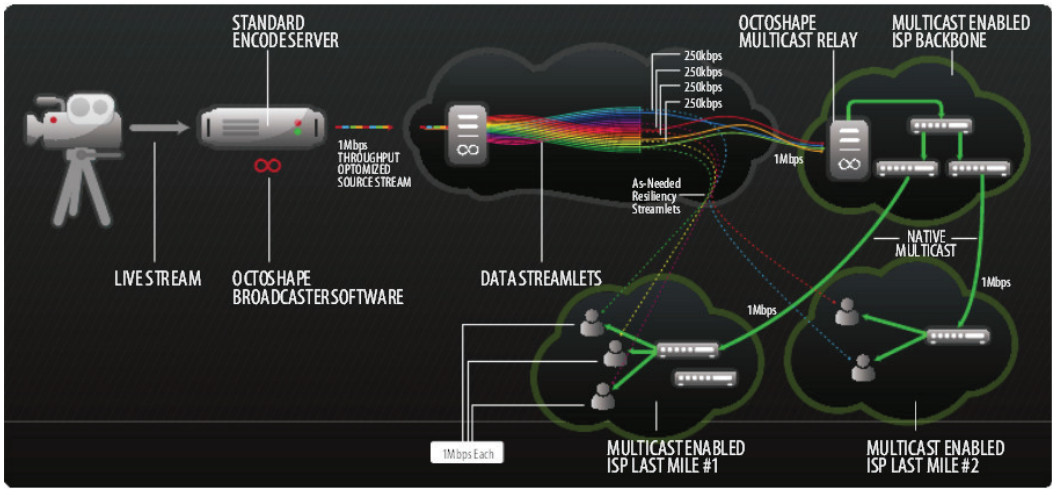
(Εικόνα 14: Λειτουργία DASH με Smooth Streaming[26])

3.4 Octoshape Multi-BitRate

Η Octoshape χωρίς να παρέχει πολλές πληροφορίες για την τεχνολογία που χρησιμοποιεί έχει ακριβώς το ίδιο σκεπτικό με τις παραπάνω εταιρίες και συνδυάζει το http streaming με τα δίκτυα διανομής περιεχομένου(CDNs) για κατανεμημένη απόδοση στην μετάδοση πολυμεσικού περιεχομένου δίνοντας έμφαση στην απόσταση των μερών που συνδέονται για την μεταφορά των δεδομένων και την αύξηση της ποιότητας με βάση αυτή.



(Εικόνα 15: Μετάδοση περιεχομένου μέσα από CDNs[21])



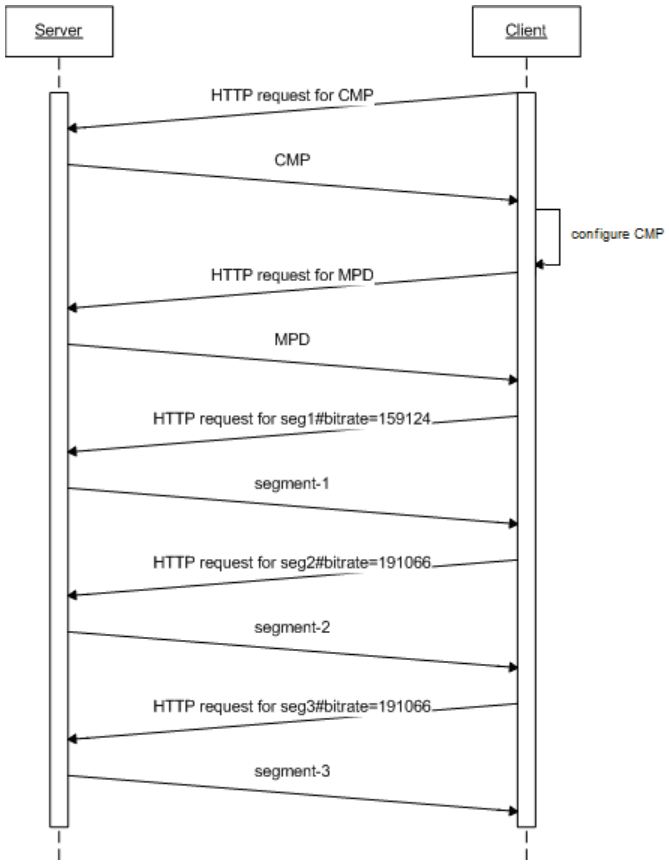
(Εικόνα 16: Αρχιτεκτονική του Octoshape Multi-BitRate[21])

3.5 DASH VLC plugin

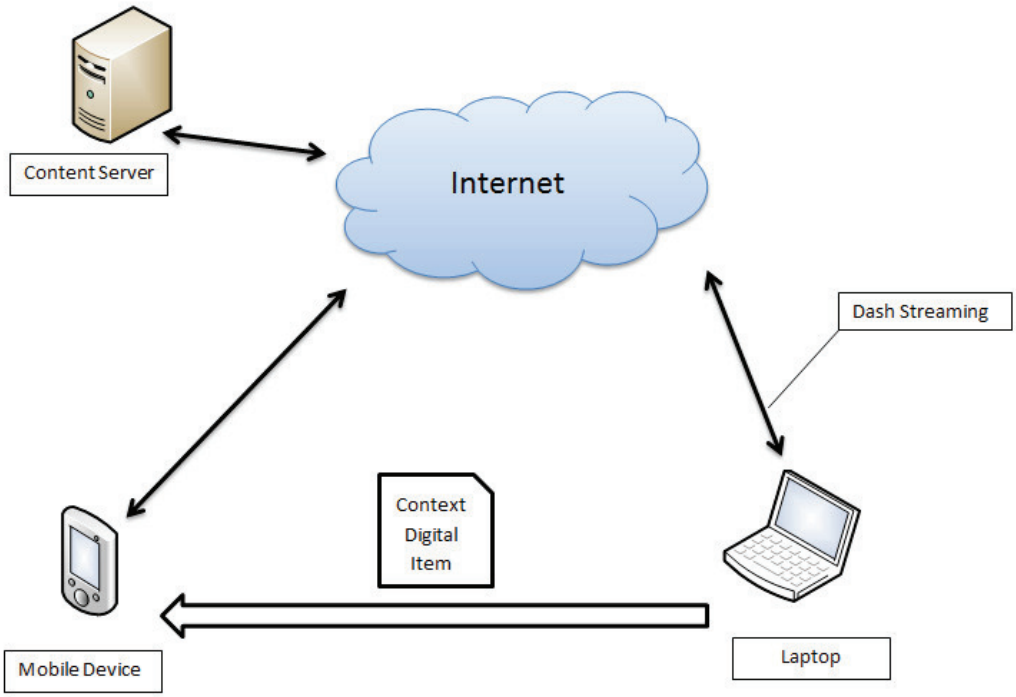
Ο Christopher Müller και ο Christian Timmerer[7] σχεδίασαν ένα dash plugin για το vlc player με σκοπό την αναπαραγωγή πολυμεσικού περιεχομένου μέσω http streaming και την δυνατότητα προσαρμογής του με διαφορετικό τρόπο ανάλογα την συσκευή που δέχεται το περιεχόμενο όπως netbook, smartphone κ.τ.λ.. Στην τελευταία έκδοση του vlc player το plugin ήταν ήδη εγκαταστημένο ένω αρχικά χρειαζόταν ξεχωριστή εγκατάσταση.

Το plugin σχεδιάστηκε για ερευνητικές και πειραματικές δραστηριότητες και δεν χρησιμοποιείται επίσημα για την αναπαραγωγή περιεχομένου μέσω http streaming.

Η λογική που ακολουθείται είναι παρόμοια με όλες τις υπόλοιπες και αναφέρεται σε ένα ένα client που ζητάει το αρχείο ρυθμίσεων προβολής-CMP και περιγραφής περιεχομένου-MPD από το server και τα αρχεία είναι σε μορφή mp4.



(Εικόνα 17: Λειτουργία του DASH VLC plugin)



(Εικόνα 18: Αρχιτεκτονική του DASH VLC plugin)

3.6 GPAC

Το GPAC είναι ένα ανοιχτού κώδικα πολυεργαλείο πολυμέσων για ακαδημαϊκούς και ερευνητικούς σκοπούς. Καλύπτει διάφορους τομείς των πολυμέσων δίνοντας έμφαση στις τεχνολογίες παρουσίασης όπως γραφικά, κινούμενα σχέδια, διαδραστικότητα κ.τ.λ.. Τρέχει σχεδόν σε όλα τα λειτουργικά συστήματα και είναι γραμμένο σε γλώσσα ANSI C από τους δημιουργούς της Telecom ParisTech (a.k.a. ENST).

Το GPAC συμμετέχει ενεργά στην λειτουργία του DASH προσφέροντας εργαλεία για την δημιουργία και την επεξεργασία DASH περιεχομένων.

Τα εργαλεία είναι:

- MP4Box, εργαλείο για την δημιουργία MP4 περιεχομένων
- MP42TS, εργαλείο για την μετάδοση MPEG-2 περιεχομένου από είσοδο MP4
- Osmo4, MP4Client κ.τ.λ., εργαλέια για αναπαραγωγή DASH περιεχομένου

Οι τύποι των αρχείων που διαχειρίζεται το GPAC είναι ίδιοι με το DASH VLC plugin με καταλήξεις MPD και MP4 για την περιγραφή περιεχομένου και την μορφή του αρχείου αντίστοιχα.

3.7 GStreamer

Το GStreamer παρόμοια με το GPAC προσφέρει βιβλιοθήκη εργαλείων για ερευνητικούς και ακαδημαϊκούς σκοπούς. Έχει σκοπό να προσφέρει αλλά και να δέχεται υλικό χρήσιμο για την νέα τεχνολογία πάνω σε θέματα μετάδοσης και αναπαραγωγής βίντεο.

Αν και δεν διαθέτει κάποια τελική έκδοση με πολλές πληροφορίες έχει ασχοληθεί και με το DASH διαβάζοντας τα αρχεία της Apple με καταλήξεις .m3u8 για το αρχείο περιγραφής περιεχομένου και .ts για τα αρχεία πολυμέσων.

Μία τυπική πρόταση εντολών για την έναρξη της μετάδοσης πολυμέσων είναι η παρακάτω:

souphttpsrc location=http://localhost/myvideo.m3u8 ! livestreamdec ! queue ! mpegtsdemux ! queue ! ffdec_h264 ! queue ! ffmpegcolorspace ! queue ! xvimagesink

4. Επισκόπηση του DASH με το “Adobe HTTP Dynamic Streaming”

4.1 Εισαγωγή

Για την εξέταση της λειτουργίας του DASH επιλέχθηκε το Adobe HTTP Dynamic Streaming από το σύνολο των υλοποιήσεων που είδαμε στο 3 κεφάλαιο και πάνω σε αυτό έγιναν κάποιες αλλαγές στην λειτουργία του για τις ανάγκες της πτυχιακής εργασίας και του υλικού που χρειάστηκε να γίνει ο έλεγχος συμπεριφοράς και οι απαραίτητες μετρήσεις μέσα από μία σειρά επαναλαμβανόμενων πειραμάτων για διαφορετικές περιπτώσεις συνθηκών απόδοσης δικτύου αλλά και διαφορετικού πολυμεσικού περιεχομένου.

Οι λόγοι που επιλέχθηκε η υλοποίηση της Adobe από τις υπόλοιπες είναι οι εξής:

- Η λειτουργία του DASH από την Adobe ήταν επιτυχής και δεν προκάλεσε προβλήματα ή σφάλματα κατά την διάρκεια εκτέλεσης της καθώς αποτελεί ένα επίσημα ολοκληρωμένο έργο έτοιμο προς εφαρμογή στο χώρο της μετάδοσης πολυμέσων.
- Η Adobe προσφέρει όλα τα απαραίτητα εργαλεία που χρειάζονται για την υλοποίηση του DASH δωρεάν ή σε δοκιμαστική χρήση με σκοπό την δημιουργία και την επεξεργασία ενός DASH περιβάλλοντος.
- Όλα τα εγχειρίδια χρήσεως και τεχνικής υποστήριξης είναι εύκολα και χωρίς χρέωση προσβάσιμα στον καθέναν καθώς επίσης αμέτρητα έγγραφα άλλων χρηστών και σχεδιαστών βρίσκονται αναρτημένα στην ιστοσελίδα της Adobe με σκοπό να προσφέρουν συμβουλές και διευκολύνσεις πάνω σε θέματα σχεδιασμού και χρήσης του DASH περιβάλλοντος.
- Πλήρης συμβατότητα με όσα λειτουργικά συστήματα τρέχουν flash player δηλαδή το 99%.

- Ένα σημαντικό κομμάτι από την πλευρά του client το DASH player πρόγραμμα είναι ανοιχτού κώδικα από την Open Source Media Framework και εκτός ότι προσφέρει έτοιμα sample players για την αναπαραγωγή πολυμεσικού περιεχομένου υπάρχει και πρόσβαση στο κώδικα έτσι ώστε να τροποποιηθεί κατάλληλα για διαφορετικές εφαρμογές.

Στις παρακάτω ενότητες θα δούμε την εγκατάσταση και την δικτυακή τοπολογία του περιβάλλοντος που χρησιμοποιήθηκε στο πείραμα στα πλαίσια της πτυχιακής εργασίας και τα αποτελέσματα διαφόρων μετρήσεων από την εκτέλεση, στο παράρτημα αναλύονται τα σημαντικότερα τμήματα του κώδικα από το DASH player της Open Source Media Framework.

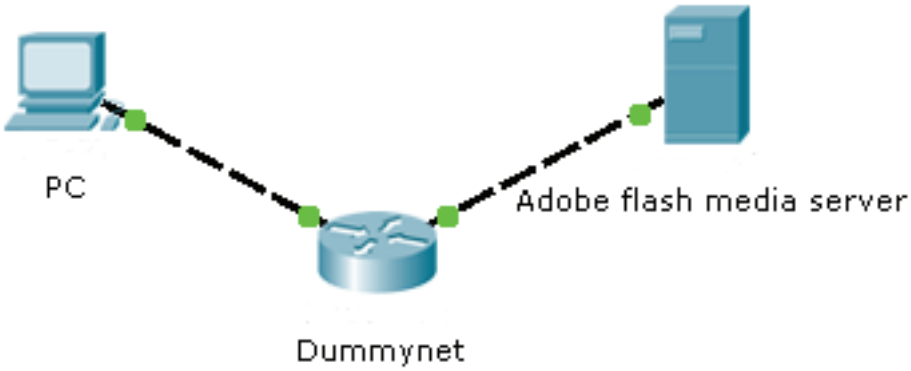
4.2 Περιβάλλον

Για το περιβάλλον στο οποίο εκτελέστηκε το πείραμα χρησιμοποιήθηκε ένα windows xp pc σαν client, ένα Flash Media Server με λειτουργικό windows και ανάμεσα τους ένα Dummynet System βασισμένο σε λειτουργικό Unix όπως φαίνεται στην εικόνα 19.

Στον client εγκαταστάθηκε η τελευταία έκδοση flash player και το google chrome χρησιμοποιήθηκε σαν browser για περιήγηση στις σελίδες με το πολυμεσικό υλικό που περιέχει ο server.

Στον Server εγκαταστάθηκε το free Flash Media Server 4.5(Παρ:εικόνα 31) για την υποστήριξη του Adobe DASH και έπειτα για την αναπαραγωγή του video streaming χρησιμοποιήθηκε το ανοιχτό OSMF 1.6.1 από το οποίο επιλέχθηκε το StrobeMediaPlayback player σε debug μορφή(Παρ:εικόνα 32,33) για πρακτικούς λογους. Για την επεξεργασία του κώδικά και το χτίσιμο του StrobeMediaPlayback χρησιμοποιήθηκε το Adobe Flash Builder 4.5(Παρ:εικόνα 34).

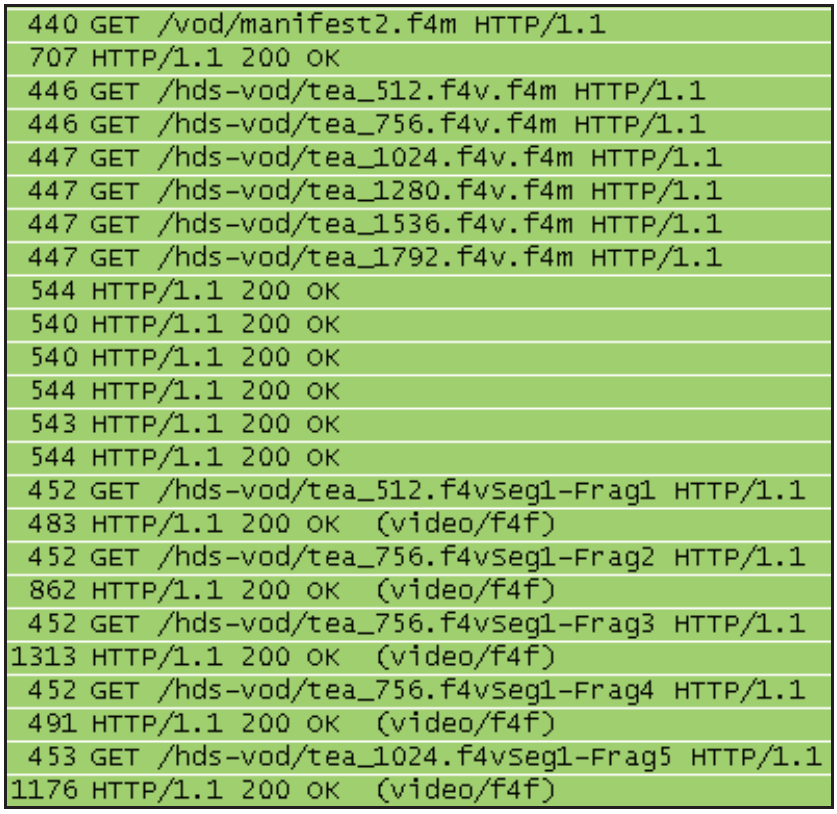
Στο πείραμα χρησιμοποιήθηκαν δύο διαφορετικά βίντεο τα οποία κωδικοποιήθηκαν μέσω του Adobe Media Encoder CS5.5(Παρ:εικόνα 35) σε μορφή .f4v σε 5 διαφορετικές ποιότητες: 512, 756, 1024, 1280, 1536 από την χειρότερη στην καλύτερη και με την χρήση του εργαλείου F4M File Generator(Παρ:εικόνα 36) δημιουργήθηκε το αρχείο δομής περιεχομένου ή manifest file(Παρ:εικόνα 37) σε μορφή xml.



(Εικόνα 19: Περιβάλλον εκτέλεσης DASH)

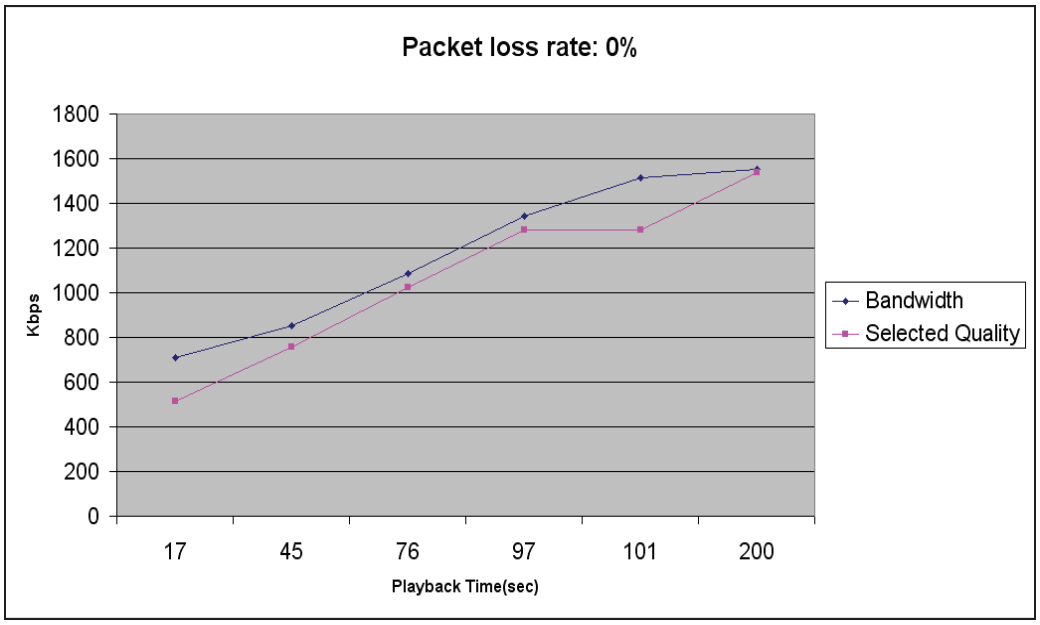
4.3 Πείραμα

Στο πείραμα ο σκοπός ήταν να παρατηρηθεί η συμπεριφορά στην αναπαραγωγή του βίντεο όταν γίνονται παράλληλα κάποιες αλλαγές στην απόδοση του δικτύου. Έτσι το Dummynet ρυθμίστηκε για να προσφέρει δυναμικά και τυχαία τιμές bandwidth με βάση την κατανομή Gauss από 500 Kbps μέχρι 2 Mbps όπως επίσης και σταθερή απώλεια πακέτων σε τιμές 0%, 2%, 3%, 5% και 8% εξετάζοντας κάθε περίπτωση ξεχωριστά. Για την καταγραφή οποιασδήποτε πληροφορίας και μέτρησης χρησιμοποιήθηκε η debug έκδοση του player και το πρόγραμμα wireshark(εικόνα 20).

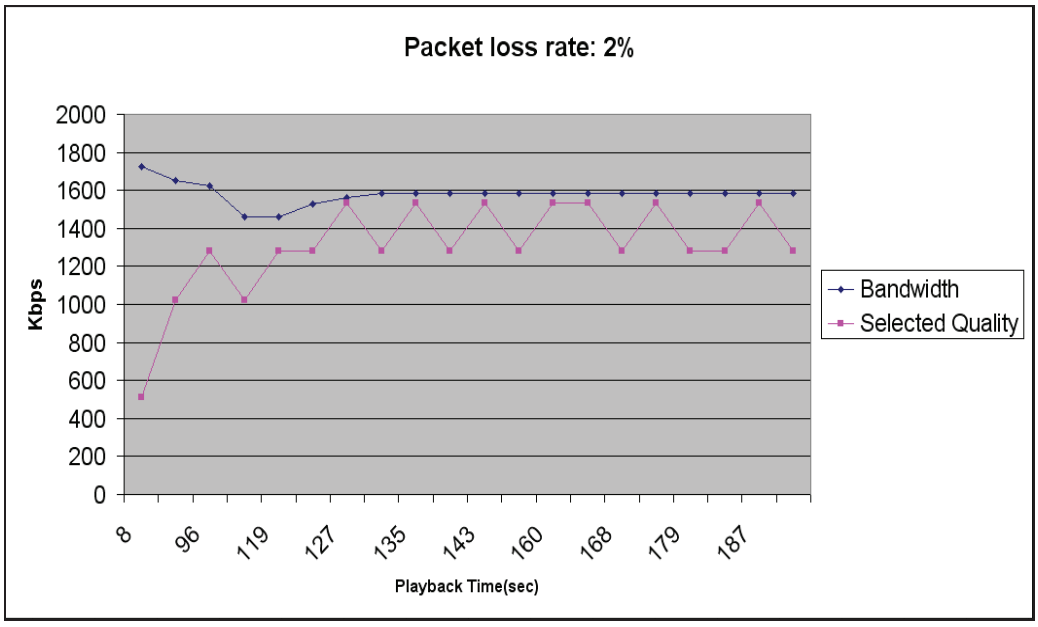


(Εικόνα 20: HTTP DASH Handshaking καταγραφή από wireshark)

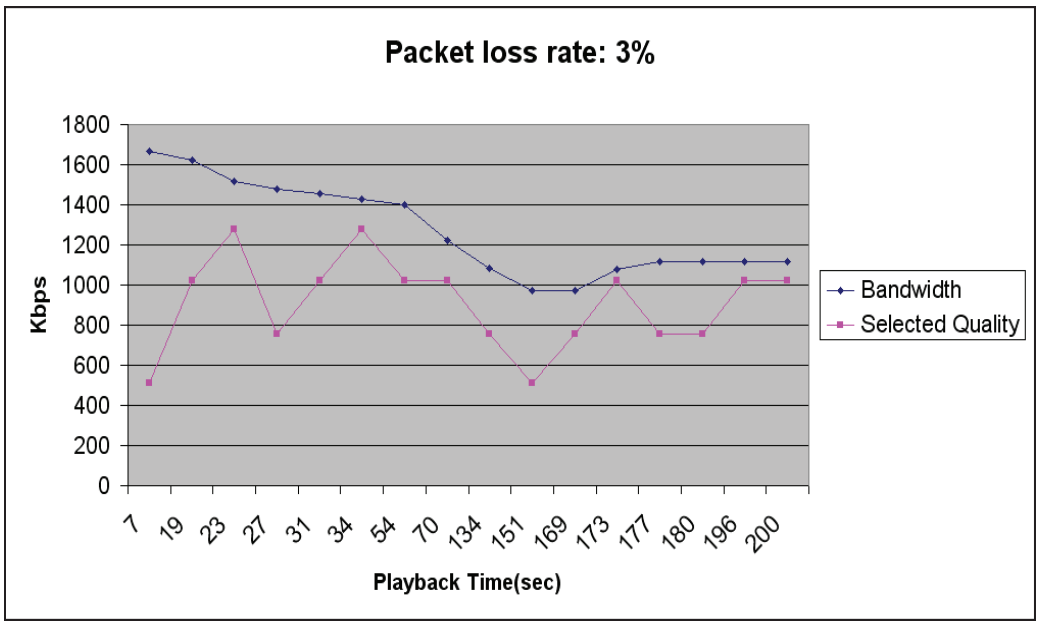
Παρακάτω στις εικόνες 21-28 φαίνονται τα αποτελέσματα από τις μετρήσεις πολλών πειραμάτων που έγιναν.



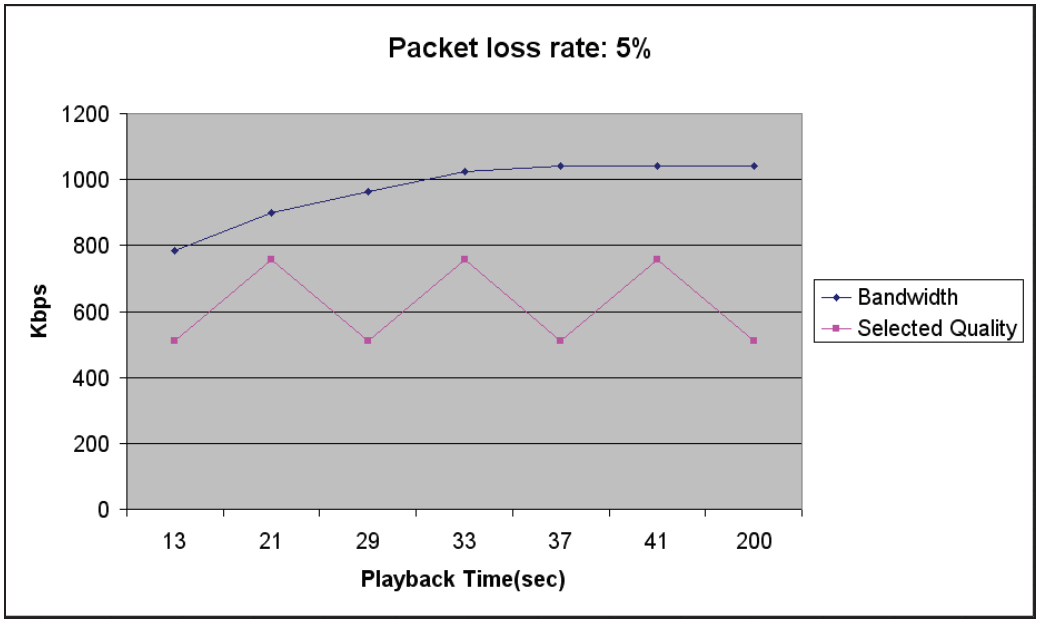
(Εικόνα 21: Network Bandwidth vs Selected Quality for packet loss rate 0%)



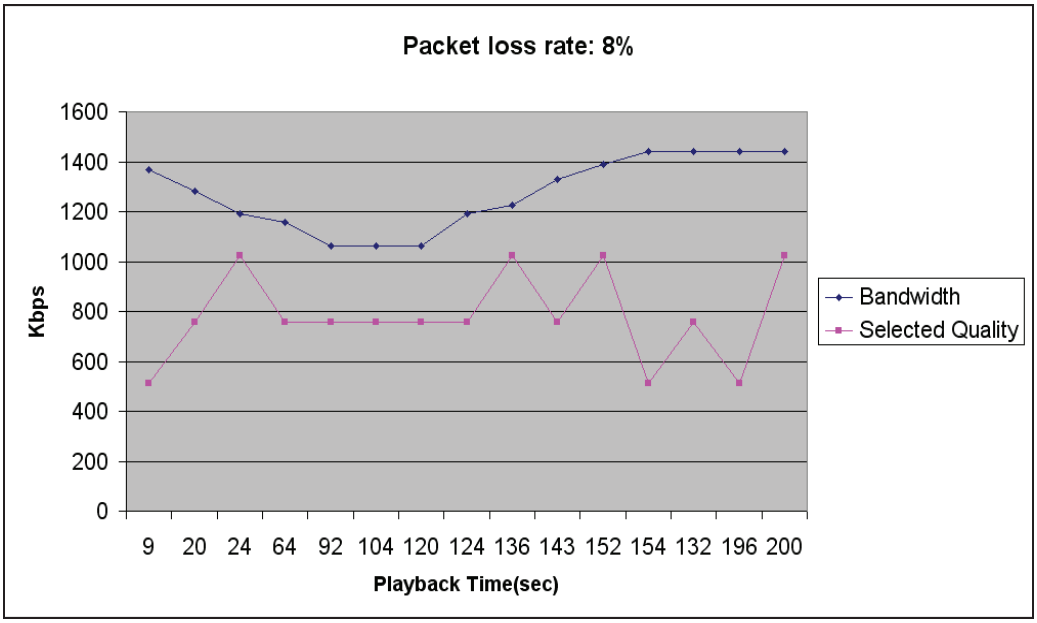
(Εικόνα 22: Network Bandwidth vs Selected Quality for packet loss rate 2%)



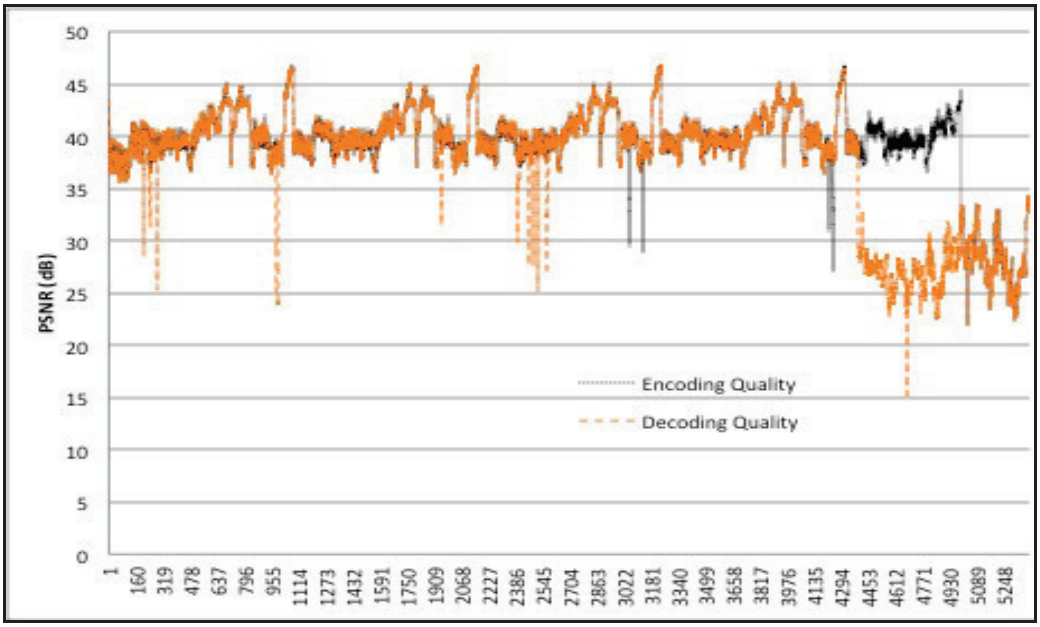
(Εικόνα 23: Network Bandwidth vs Selected Quality for packet loss rate 3%)



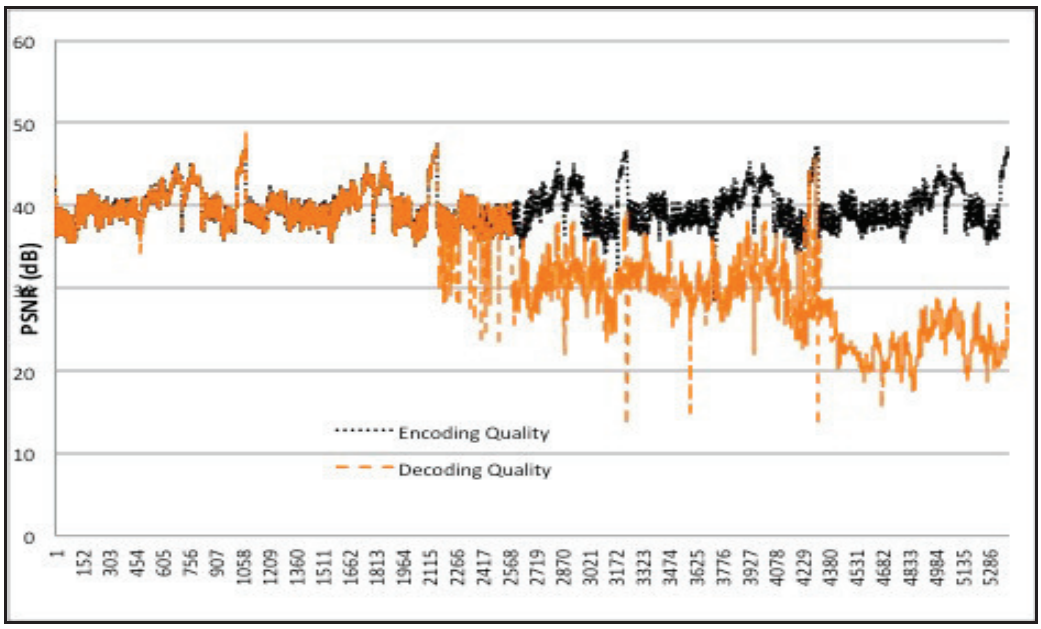
(Εικόνα 24: Network Bandwidth vs Selected Quality for packet loss rate 5%)



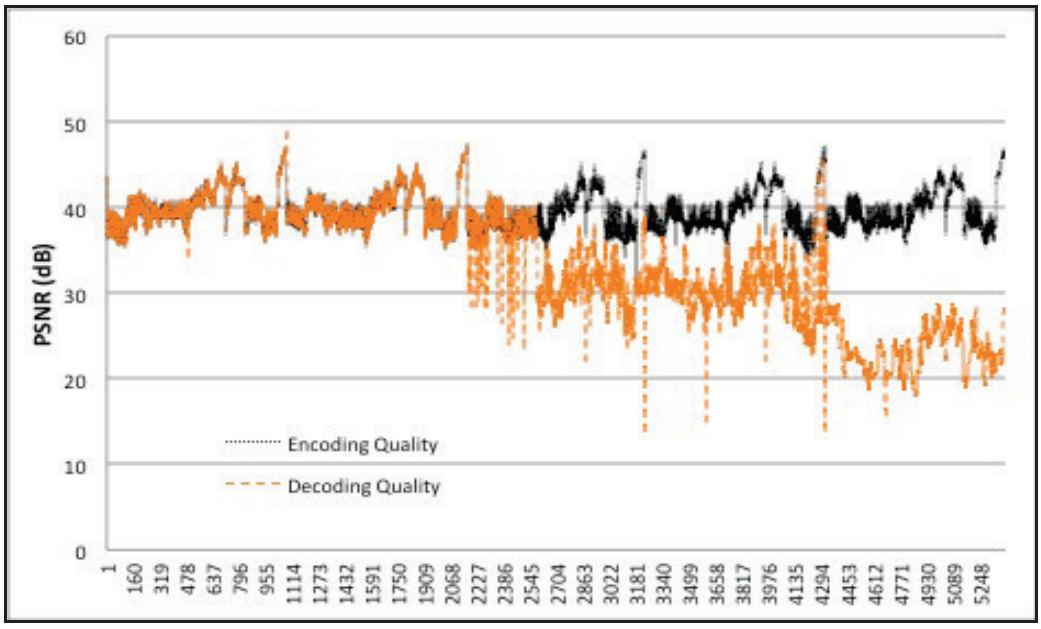
(Εικόνα 25: Network Bandwidth vs Selected Quality for packet loss rate 8%)



(Εικόνα 26: PSNR Encoding-Decoding Quality for 2% packet loss)



(Εικόνα 27: PSNR Encoding-Decoding Quality for 3% packet loss)



(Εικόνα 28: PSNR Encoding-Decoding Quality for 5% packet loss)

4.4 Συμπεράσματα από τα πειράματα

Με βάση τα πειράματα από τις αλλαγές στην απόδοση του δικτύου βγαίνει το συμπέρασμα ότι όταν αυξάνεται η απώλεια των πακέτων επηρεάζεται αρνητικά το throughput και το PSNR με αποτέλεσμα να υπάρχει μεγαλύτερη απόσταση στην σχέση bandwidth και ποιότητας βίντεο. Η λογική του DASH επιτρέπει την δυναμική αλλαγή στην ποιότητα του βίντεο με βάση τις τυχαίες αλλαγές που μπορούν να συμβούν στην απόδοση του δικτύου. Ένα στοιχείο αδυναμίας του δικτύου όπως η απώλεια πακέτων επηρεάζει με αρνητικό τρόπο την λογική και τις αποφάσεις του αλγορίθμου προσαρμογής ποιότητας του βίντεο έτσι ώστε να μην επιλέγεται πάντα η σωστή ποιότητα που αναλογεί στο bandwidth του δικτύου και αυτό αποτελεί ένα από τα ζητήματα που χρειάζεται να ερευνηθούν μελλοντικά. Επίσης σε ερευνητικό στάδιο μελετάται και η χρήση DASH εφαρμογών κάτω από δίκτυα κινητής τηλεφωνίας με την χρήση μιας κινητής συσκευής η οποία μεταφέρεται από διάφορα μέσα μεταφοράς και ελέγχεται η αντίδραση του αλγορίθμου για κάθε διαφορετικό περιβάλλον προσαρμογής.

5. Βιβλιογραφία.

[1] A. C. Begen, T. Akgul, and M. Baugher, “Watching video over the Web, part I: streaming protocols”, IEEE Internet Computing, Vol. 15, March 2011.

[2] S.-F. Chang and A. Vetro, “Video Adaptation: Concepts, Technologies, and Open Issues”, Vol. 93, January 2005.

[3] IETF RFC 2616: "Hypertext Transfer Protocol – HTTP/1.1", June 1999.

[4] R. Pantos, W. May, “HTTP Live Streaming”, IETF Draft, March 2012.

[5] T.Stockhammer, “Dynamic adaptive streaming over http:standards and design principles”, Second Annual ACM Conference on Multimedia Systems, MMSys’11, San-Jose, California, USA, Ferbuary 2011.

[6] S. Akhshabi, S. Narayanaswamy, A. C. Begen and C. Dovrolis, “An experimental evaluation of rate-adaptive video players over HTTP”, Signal Processing: Image Communication, Vol. 27, April 2012.

[7] C. Müller and C. Timmerer, “A Test-Bed for the Dynamic Adaptive Streaming over HTTP featuring Session Mobility”.

[8] Ricky K. P. Mok, Edmond W. W. Chan and Rocky K. C. Chang, “Measuring the Quality of Experience of HTTP Video Streaming”.

[9] Adobe. HTTP Dynamic Streaming on the Adobe.

[10] Flash Platform. Adobe Systems Incorporated, 2010. http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf.

[11] The consumer digital video library, 〈http://www.cdvl.org>.

[12] L. Rizzo, “Dummynet: a simple approach to the evaluation of network protocols”, ACMSIGCOMM Computer Communication Review, 27 (1997).

[13] Alex Zambelli, IIS Smooth Streaming Technical Overview.

[14] Niels Laukens, adaptive streaming-a brief tutorial.

[15] www.longtailvideo.com, Adaptive HTTP Streaming Framework.

[16] http://el.wikipedia.org/wiki/TCP/IP

[17] http://en.wikipedia.org/wiki/Adaptive_bitrate_streaming

[18] http://el.wikipedia.org/wiki/Προώθηση_(υπολογιστές)

[19] http://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP

[20] https://developer.apple.com/library/ios/#documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html

[21] http://www.scribd.com/doc/52606590/Octoshape-Solution-Paper

[22] http://gpac.wp.mines-telecom.fr/

[23] http://www.movingpics.tv/zbutcher/gstreamer-supports-mpeg-dash-through-dashbin-bugzilla-gnome-org

[24] http://gstreamer.freedesktop.org/conference/speakers.html

[25] http://noc.auth.gr/services/voice-video/mbone/index.html

[26] Img14-http://www.misfitgeek.com/2009/10/iis-media-services-3-0/

6. Παράρτημα: Κώδικας Adobe DASH player

6.1 Κώδικας

Στο πείραμα χρησιμοποιήσαμε το StrobeMediaPlayback player του οποίου ο κώδικας που μας ενδιαφέρει να αναφέρουμε βρίσκεται στην διαδρομή StrobeMediaPlayback\src\org\osmf\net σε διάφορες κλάσεις καθώς πρόκειται για την αντικειμενοστραφής γλώσσα Action Script 3.0. Από αυτές οι πιο σημαντικές είναι οι StrobeNetStreamSwitchManager,NetStreamBufferManagerBase,PlaybackOptimizatio nMetrics και HTTPStreamingNetLoaderAdapter.

Παρακάτω γίνεται περιγραφή και παρουσίαση του κώδικα των κλάσεων:

a) HTTPStreamingNetLoaderAdapter: Ο σκοπός αυτής της κλάσης είναι να αρχικοποιήσει την λειτουργία DASH από πλευράς συνδέσεων δικτύου και επικοινωνίας.

```
package org.osmf.net
{
	import flash.events.NetStatusEvent;
	import flash.net.NetConnection;
	import flash.net.NetStream;

	import org.osmf.events.*;
	import org.osmf.media.URLResource;
	import org.osmf.net.httpstreaming.*;
	import org.osmf.net.rtmpstreaming.DroppedFramesRule;
	import org.osmf.net.httpstreaming.f4f.*;
	/**
	 * PlaybackOptimization adapter for RTMPDynamicStreamingNetLoader.
	 */
	public class HTTPStreamingNetLoaderAdapter extends HTTPStreamingNetLoader
	{
		/**
		 * Constructor.
		 */
		public function
HTTPStreamingNetLoaderAdapter(playbackOptimizationManager:PlaybackOptimizationManager
)
		{
			this.playbackOptimizationManager = playbackOptimizationManager;
			super();
		}
```

```
/**
                * @private
                *
                * Overridden to allow the creation of a NetStreamSwitchManager object.
                *
                *  @langversion 3.0
                *  @playerversion Flash 10
                *  @playerversion AIR 1.5
                *  @productversion OSMF 1.0
                */
               override protected function
createNetStreamSwitchManager(connection:NetConnection, netStream:NetStream,
dsResource:DynamicStreamingResource):NetStreamSwitchManagerBase
               {
                        playbackOptimizationManager.optimizePlayback(connection,
netStream, dsResource);
                        netStream.addEventListener(NetStatusEvent.NET_STATUS,
onNetStatus);
                        return super.createNetStreamSwitchManager(connection, netStream,
dsResource);
               }

               // Internals
               //
               private function onNetStatus(event:NetStatusEvent):void
               {
                        var netStream:NetStream = event.currentTarget as NetStream;
                        if (event.info.code ==
NetStreamCodes.NETSTREAM_BUFFER_EMPTY)
                        {
                                if (netStream.bufferTime >= 2.0)
                                {
                                        netStream.bufferTime += 1.0;
                                }
                                else
                                {
                                        netStream.bufferTime = 2.0;
                                }
                        }
               }

               private var playbackOptimizationManager:PlaybackOptimizationManager;
        }
}
```

b) PlaybackOptimizationMetrics: Υπολογίζει διάφορες τιμές απόδοσης δικτύου με βάση τις οποίες ο αλγόριθμος του DASH παίρνει βέλτιστες αποφάσεις για την προσαρμογή του streaming στις ικανότητες του τερματικού.

```
package org.osmf.net
{
	import flash.events.NetStatusEvent;
	import flash.net.NetStream;
	import flash.net.SharedObject;
	import flash.system.System;

	import org.osmf.player.utils.StrobeUtils;

	CONFIG::LOGGING
	{
	import org.osmf.logging.Log;
	import org.osmf.player.debug.StrobeLogger;
	}

	public class PlaybackOptimizationMetrics extends NetStreamMetricsBase
	{
		public static const ID:String = "PlaybackOptimizationMetrics";

		/**
		 * Constructor.
		 *
		 * @param netStream The NetStream to provide metrics for.
		 **/
		public function PlaybackOptimizationMetrics(netStream:NetStream)
		{
			super(netStream);
			// Retrieve the metadata from the NetStream

	NetClient(netStream.client).addHandler(NetStreamCodes.ON_META_DATA,
onMetaData);
			function onMetaData(value:Object):void
			{

	NetClient(netStream.client).removeHandler(NetStreamCodes.ON_META_DATA,
onMetaData);
				// We assume that the asset metadata has the most accurate
value,
				// that's why we overwrite any existing value.

				// audiodatarate and videodatarate are set in kilobits. We need
to transform it to bytes.
				averagePlaybackBytesPerSecond =
StrobeUtils.kbitsPerSecond2BytesPerSecond(value.audiodatarate + value.videodatarate);

				duration = value.duration;
			}

			try
			{
				var sharedObject:SharedObject =
SharedObject.getLocal(STROBE_LSO_NAMESPACE);
				averageDownloadBytesPerSecond =
StrobeUtils.kbitsPerSecond2BytesPerSecond(sharedObject.data.downloadKbitsPerSecond);
				CONFIG::LOGGING
				{
					logger.qos.lsoDownloadKbps =
sharedObject.data.downloadKbitsPerSecond;
				}
			}
```

```
catch(ignore: Error)
                        {
                                // Ignore this error
                                CONFIG::LOGGING
                                {
                                        logger.info("LSO access is disabled.");
                                }
                        }

                        // Register a NET_STATUS handler, so that we update the local
SharedObject with the new bandwidth measurement.
                        netStream.addEventListener(NetStatusEvent.NET_STATUS,
onNetStatus);
                        function onNetStatus(event:NetStatusEvent):void
                        {
                                if (event.info.code ==
NetStreamCodes.NETSTREAM_PLAY_STOP || event.info.code ==
NetStreamCodes.NETSTREAM_PAUSE_NOTIFY)
                                {
                                        if (averageDownloadBytesPerSecond > 0)
                                        {
                                                try
                                                {

                                                        // Update the Local SharedObject
with the latest bandwidth measurement.
                                                        var sharedObject:SharedObject =
SharedObject.getLocal(STROBE_LSO_NAMESPACE);

        sharedObject.data.downloadKbitsPerSecond =
Math.round(StrobeUtils.bytesPerSecond2kbitsPerSecond(averageDownloadBytesPerSecond));
                                                        sharedObject.flush(10000);
                                                        CONFIG::LOGGING
                                                        {

        logger.qos.lsoDownloadKbps = sharedObject.data.downloadKbitsPerSecond;
                                                        }
                                                }
                                                catch (ingore:Error)
                                                {
                                                        // Ignore this error
                                                        CONFIG::LOGGING
                                                        {

        logger.info("lsoDownloadKbps value is not updated since LSO access is disabled.");
                                                        }
                                                }
                                        }
                                }
                        }
                }

                public function get duration():Number
                {
                        return _duration;
                }

                public function set duration(value:Number):void
                {
                        _duration = value;
                }
```

```
/**
                     * The average download ration per second value, calculated based on a
                     * recent set of samples.
                     *
                     *  @langversion 3.0
                     *  @playerversion Flash 10
                     *  @playerversion AIR 1.5
                     *  @productversion OSMF 1.0
                     */
                    public function get downloadRatio():Number
                    {
                            return averageDownloadBytesPerSecond /
averagePlaybackBytesPerSecond;
                    }

                    /**
                     * The average playback bytes per second value, calculated based on a
                     * recent set of samples.
                     *
                     *  @langversion 3.0
                     *  @playerversion Flash 10
                     *  @playerversion AIR 1.5
                     *  @productversion OSMF 1.0
                     */
                    public function get averagePlaybackBytesPerSecond():Number
                    {
                            return avgPlaybackBytesPerSecond.average;
                    }

                    public function set averagePlaybackBytesPerSecond(value:Number):void
                    {
                            avgPlaybackBytesPerSecond.clearSamples();
                            avgPlaybackBytesPerSecond.addSample(value);
                    }

                    /**
                     * The average max bytes per second value, calculated based on a
                     * recent set of samples.
                     *
                     *  @langversion 3.0
                     *  @playerversion Flash 10
                     *  @playerversion AIR 1.5
                     *  @productversion OSMF 1.0
                     */
                    public function get averageDownloadBytesPerSecond():Number
                    {
                            return avgMaxBytesPerSecond.average;
                    }

                    public function set averageDownloadBytesPerSecond(value:Number):void
                    {
                            avgPlaybackBytesPerSecond.clearSamples();
                            return avgMaxBytesPerSecond.addSample(value);
                    }

                    public function get averageDownloadKbps():Number
                    {
                            return averageDownloadBytesPerSecond  / 128;
                    }
```

```
public function get averagePlaybackKbps():Number
                    {
                              return averagePlaybackBytesPerSecond / 128;
                    }

                    /**
                     * @private
                     **/
                    override protected function calculateMetrics():void
                    {
                              super.calculateMetrics();
                              // XXX cdobre - this needs to be refactored in order
                              // to eliminate the dynamic dependency on property name
                              if (netStream.hasOwnProperty("qosInfo") && netStream["qosInfo"] !=
null)
                              {
                                        // HTTPStreaming content.
                                        if (netStream.info.videoBufferLength > 0)
                                        {

          avgPlaybackBytesPerSecond.addSample(netStream.info.videoBufferByteLength /
netStream.info.videoBufferLength);
                                        }
                                        maxBytesPerSecond =
netStream["qosInfo"]["downloadRatio"] * netStream.info.videoBufferByteLength /
netStream.info.videoBufferLength;
                                        if (maxBytesPerSecond > 0)
                                        {

          avgMaxBytesPerSecond.addSample(maxBytesPerSecond);
                                        }
                              }
                              else
                              {
                                        if (netStream.bytesLoaded > 0 )
                                        {
                                                  // Progressive content
                                                  if (netStream.bytesLoaded < netStream.bytesTotal)
                                                  {
                                                            if (netStream.info.videoBufferLength > 0)
                                                            {

          avgPlaybackBytesPerSecond.addSample(netStream.info.videoBufferByteLength /
netStream.info.videoBufferLength);
                                                            }

          avgMaxBytesPerSecond.addDeltaTimeRatioSample(netStream.bytesLoaded, new
Date().time / 1000);
                                                  }
                                        }
                                        else
                                        {
```

```
// RTMP Content
                                        var playbackBytesPerSecond:Number =
netStream.info.playbackBytesPerSecond;
                                        var maxBytesPerSecond:Number =
netStream.info.maxBytesPerSecond;
                                        if (playbackBytesPerSecond > 0)
                                        {

        avgPlaybackBytesPerSecond.addSample(playbackBytesPerSecond);
                                        }

                                        if (maxBytesPerSecond > 0)
                                        {

        avgMaxBytesPerSecond.addSample(maxBytesPerSecond);
                                        }
                                    }
                                }

                                CONFIG::LOGGING
                                {
                                logger.qos.duration = _duration;

                                logger.qos.downloadRatio = this.downloadRatio;
                                logger.qos.playbackKbps  = this.averagePlaybackBytesPerSecond / 128;
                                logger.qos.downloadKbps  = this.averageDownloadBytesPerSecond /
128;
                                logger.qos.avgDroppedFPS = averageDroppedFPS;

                                logger.trackObject("PlaybackOptimizationMetrics", this);
                                }
                        }

                        // Internals
                        //
                        private var _duration:Number;

                        private var avgPlaybackBytesPerSecond:RunningAverage = new
RunningAverage(DEFAULT_AVG_MAX_BYTES_SAMPLE_SIZE);
                        private var avgMaxBytesPerSecond:RunningAverage = new
RunningAverage(DEFAULT_AVG_MAX_BYTES_SAMPLE_SIZE);

                        private var previousMeasurementTimestamp:Number = NaN;
                        private static const DEFAULT_AVG_MAX_BYTES_SAMPLE_SIZE:Number
= 50;

                        private const STROBE_LSO_NAMESPACE:String =
"org.osmf.strobemediaplayback.lso";

                        CONFIG::LOGGING
                        private var logger:StrobeLogger = Log.getLogger("StrobeMediaPlayback") as
StrobeLogger;

        }
}
```

c) NetStreamBufferManagerBase: Διαχειρίζεται βέλτιστα και δυναμικά τις τιμές του Buffer κατά την διάρκεια αναπαραγωγής του βίντεο. Η αποστολή του βίντεο γίνεται ανά 100 frames δηλαδή ανά 4 δευτερόλεπτα από το σύνολο διάρκειας του και για αυτό η προεπιλεγμένη τιμή του δυναμικού Buffer είναι 4 δευτερόλεπτα.

```
package org.osmf.net
{
	import flash.events.NetStatusEvent;
	import flash.net.NetStream;

	CONFIG::LOGGING
	{
	import org.osmf.logging.Log;
	import org.osmf.player.debug.StrobeLogger;
	}
	/**
	 * BufferManager is responsible for updating the bufferTime based on the NetStream
events and
	 * bandwidth versus video bitrate ratio.
	 */
	public class NetStreamBufferManagerBase
	{
		/**
		 * Constructor. Starts listening on the NetStream events.
		 */
		public function NetStreamBufferManagerBase(netStream:NetStream,
metrics:PlaybackOptimizationMetrics)
		{
			this.netStream = netStream;
			this.metrics = metrics;

			netStream.addEventListener(NetStatusEvent.NET_STATUS,
onNetStatus);
		}

		public function get expandedBufferTime():Number
		{
			return _expandedBufferTime;
		}

		public function set expandedBufferTime(value:Number):void
		{
			_expandedBufferTime = value;
		}

		public function get initialBufferTime():Number
		{
			return _initialBufferTime;
		}
```

```
public function set initialBufferTime(value:Number):void
		{
			_initialBufferTime = value;
		}

		public function get minContinuousTime():Number
		{
			return _minContinuousTime;
		}

		public function set minContinuousTime(value:Number):void
		{
			_minContinuousTime = value;
		}

		public function computeBufferTime():Number
		{
			if (isNaN(metrics.downloadRatio) || metrics.downloadRatio > 1)
			{
				return NaN;
			}

			if (isNaN(metrics.duration - netStream.time))
			{
				return NaN;
			}

			var result:Number;
			// The bandwidth rate is smaller then the video bit rate. Let's wait a little
bit longer but provide
			// continuous playback for a specified interval.
			var continuousTime:Number = minContinuousTime;

			if (metrics.duration - netStream.time > 0)
			{
				// If the remaining time is smaller then the remaining video
				// duration we'll set a smaller buffer size, and thus reduce the
wait time.
				continuousTime = Math.min(metrics.duration -
netStream.time, continuousTime);
			}

			// The expandedBufferTime will be computed to allow a continous
playback
			var bufferComputedTime:Number = continuousTime*(1-
metrics.downloadRatio);

			// Prevent the new buffer size from being smaller then the previous
value of the bufferExpandedTime
			bufferComputedTime = Math.max(netStream.bufferLength,
bufferComputedTime);

			return bufferComputedTime;
		}
```

```
// Protected
                    //

                    protected var netStream:NetStream;
                    protected var metrics:PlaybackOptimizationMetrics;

                    protected function onNetStatus(event:NetStatusEvent):void
                    {
                            switch (event.info.code)
                            {
                                    case NetStreamCodes.NETSTREAM_PLAY_START:
                                            if (!started)
                                            {
                                                    // On PAUSE/RESUME the buffer is not
reset, so don't update it.
                                                    // Set the initialBufferTime only for the first
play event.
                                                    netStream.bufferTime = initialBufferTime;
                                                    started = true;
                                            }
                                            break;
                                    case NetStreamCodes.NETSTREAM_SEEK_NOTIFY:
                                            // When seeking set a small bufferTime in the
beginning to allow the user to see some of the content
                                            // before providing a bigger buffer for continuous
playback. This should prevent the user from having
                                            // to wait too long if he seeked to an undesired
position.
                                            netStream.bufferTime = initialBufferTime;

                                            break;
                                    case NetStreamCodes.NETSTREAM_BUFFER_FULL:
                                            if (isNaN(metrics.downloadRatio) ||
metrics.downloadRatio > 1)
                                            {
                                                    // For high or unknown downloadRatio
simply set the buffer to the expandedBufferTime config setting.
                                                    CONFIG::LOGGING
                                                    {
                                                    logger.info("ExpandedBuffer:.
previousBufferTime={0} currentBufferTime={1}", netStream.bufferTime, expandedBufferTime);
                                                    logger.qos.buffer.time =
expandedBufferTime;
                                                    }
                                                    netStream.bufferTime =
expandedBufferTime;
                                                    break;
                                            }
                                    case NetStreamCodes.NETSTREAM_BUFFER_EMPTY:


                                            // On buffer empty we will compute a dynamic buffer
time based on bandwidth measurement.
                                            var bufferComputedTime:Number =
computeBufferTime();
                                            if (bufferComputedTime > 0 &&
bufferComputedTime != netStream.bufferTime)
                                            {
```

```
CONFIG::LOGGING
                                                    {
                                                    logger.info("DynamicBuffer:
previousBufferTime={0} currentBufferTime={1}", netStream.bufferTime, bufferComputedTime);
                                                    logger.qos.buffer.time =
bufferComputedTime;
                                                    }
                                                    netStream.bufferTime =
bufferComputedTime;
                                            }
                                            break;
                        }
                }

                // Internals
                //

                private var _initialBufferTime:Number = 1;
                private var _expandedBufferTime:Number = 10;
                private var _minContinuousTime:Number = 30;

                private var started:Boolean = false;

                CONFIG::LOGGING
                {
                private var logger:StrobeLogger = Log.getLogger("StrobeMediaPlayback") as
StrobeLogger;
                }
        }
}
```

d) StrobeNetStreamSwitchManager: Η κλάση αυτή έχει τον μεγαλύτερο ρόλο καθώς ορίζει την ίδια την λειτουργία του DASH καθώς παίρνει και εκτελεί δυναμικά τις αποφάσεις που αφορούν την ομαλή αναπαραγωγή του βίντεο αλλά και τις αλλαγές στην ποιότητα και την ανάλυση του. Συνεργάζεται με όλες τις παραπάνω κλάσεις και μπορεί να τροποποιηθεί για να θέσει όρια στις επιλογές που αφορούν την ποιότητα.

```
package org.osmf.net
{
	import __AS3__.vec.Vector;

	import flash.errors.IllegalOperationError;
	import flash.events.NetStatusEvent;
	import flash.events.TimerEvent;
	import flash.net.NetConnection;
	import flash.net.NetStream;
	import flash.net.NetStreamPlayOptions;
	import flash.net.NetStreamPlayTransitions;
	import flash.utils.Dictionary;
	import flash.utils.Timer;
	import flash.utils.getTimer;

	import org.osmf.utils.OSMFStrings;

	CONFIG::LOGGING
	{
	import org.osmf.player.debug.StrobeLogger;
	import org.osmf.logging.Logger;
	import org.osmf.logging.Log;
	}

	/**
	 * NetStreamSwitchManager is a default implementation of
	 * NetStreamSwitchManagerBase.   It manages transitions between
	 * multi-bitrate (MBR) streams using configurable switching rules.
	 *
	 *  @langversion 3.0
	 *  @playerversion Flash 10
	 *  @playerversion AIR 1.5
	 *  @productversion OSMF 1.0
	 **/
	public class StrobeNetStreamSwitchManager extends NetStreamSwitchManagerBase
	{
		/**
		 * Constructor.
		 *
		 * @param connection The NetConnection for the NetStream that will be
managed.
		 * @param netStream The NetStream to manage.
		 * @param resource The DynamicStreamingResource that is playing in the
NetStream.
		 * @param metrics The provider of runtime metrics.
		 * @param switchingRules The switching rules that this manager will use.
		 *
		 *  @langversion 3.0
		 *  @playerversion Flash 10
		 *  @playerversion AIR 1.5
		 *  @productversion OSMF 1.0
		 **/
		public function StrobeNetStreamSwitchManager
			( connection:NetConnection
			, netStream:NetStream
			, resource:DynamicStreamingResource
			, metrics:NetStreamMetricsBase
			, switchingRules:Vector.<SwitchingRuleBase>)
		{
```

```
super();

                                this.connection = connection;
                                this.netStream = netStream;
                                this.dsResource = resource;
                                this.metrics = metrics;
                                this.switchingRules = switchingRules || new
Vector.<SwitchingRuleBase>();

                                _currentIndex = Math.max(0, Math.min(maxAllowedIndex,
dsResource.initialIndex));

                                checkRulesTimer = new Timer(RULE_CHECK_INTERVAL);
                                checkRulesTimer.addEventListener(TimerEvent.TIMER, checkRules);

                                failedDSI = new Dictionary();

                                // We set the bandwidth in both directions based on a multiplier applied
to the bitrate level.
                                _bandwidthLimit = 1.4 *
resource.streamItems[resource.streamItems.length-1].bitrate * 1000/8;

                                netStream.addEventListener(NetStatusEvent.NET_STATUS,
onNetStatus);

                                // Make sure we get onPlayStatus first (by setting a higher priority)
                                // so that we can expose a consistent state to clients.

        NetClient(netStream.client).addHandler(NetStreamCodes.ON_PLAY_STATUS,
onPlayStatus, int.MAX_VALUE);
                        }

                        /**
                         * @private
                         */
                        override public function set autoSwitch(value:Boolean):void
                        {
                                super.autoSwitch = value;

                                CONFIG::LOGGING
                                {
                                        debug("autoSwitch() - setting to " + value);
                                }

                                if (autoSwitch)
                                {
                                        CONFIG::LOGGING
                                        {
                                                debug("autoSwitch() - starting check rules timer.");
                                        }
                                        checkRulesTimer.start();
                                }
                                else
                                {
                                        CONFIG::LOGGING
                                        {
                                                debug("autoSwitch() - stopping check rules timer.");
                                        }
                                        checkRulesTimer.stop();
                                }
                        }
```

```
/**
                 * @private
                 */
                override public function get currentIndex():uint
                {
                        return _currentIndex;
                }

                /**
                 * @private
                 */
                override public function get maxAllowedIndex():int
                {
                        var count:int = dsResource.streamItems.length - 1;
                        return (count < super.maxAllowedIndex ? count :
super.maxAllowedIndex);
                }

                /**
                 * @private
                 */
                override public function set maxAllowedIndex(value:int):void
                {
                        if (value > dsResource.streamItems.length)
                        {
                                throw new
RangeError(OSMFStrings.getString(OSMFStrings.STREAMSWITCH_INVALID_INDEX));
                        }
                        super.maxAllowedIndex = value;
                        metrics.maxAllowedIndex = value;
                }

                /**
                 * @private
                 **/
                override public function switchTo(index:int):void
                {
                        if (!autoSwitch)
                        {
                                if (index < 0 || index > maxAllowedIndex)
                                {
                                        throw new
RangeError(OSMFStrings.getString(OSMFStrings.STREAMSWITCH_INVALID_INDEX));
                                }
                                else
                                {
                                        CONFIG::LOGGING
                                        {
                                                debug("switchTo() - manually switching to
index: " + index);
                                        }

                                        if (metrics.resource == null)
                                        {
                                                prepareForSwitching();
                                        }
                                        executeSwitch(index);
                                }
                        }
```

```
else
                        {
                                throw new
IllegalOperationError(OSMFStrings.getString(OSMFStrings.STREAMSWITCH_STREAM_NOT
_IN_MANUAL_MODE));
                        }
                }

                // Protected
                //

                /**
                 * Override this method to provide additional decisioning around
                 * allowing automatic switches to occur.  This method will be invoked
                 * just prior to a switch request.  If false is returned, that switch
                 * request will not take place.
                 *
                 * <p>By default, the implementation does the following:</p>
                 * <p>1) When a switch down occurs, the stream being switched from has its
                 * failed count incremented. If, when the switching rules are evaluated
                 * again, a rule suggests switching up, since the stream previously
                 * failed, it won't be tried again until a duration (30s) elapses. This
                 * provides a better user experience by preventing a situation where
                 * the switch up is attempted but then fails almost immediately.</p>
                 * <p>2) Once a stream item has 3 failures, there will be no more
                 * attempts to switch to it until an interval (5m) has expired.  At the
                 * end of this interval, all failed counts are reset to zero.</p>
                 *
                 * @param newIndex The new index to switch to.
                 **/
                protected function canAutoSwitchNow(newIndex:int):Boolean
                {
                        // If this stream has failed, we don't want to try it again until
                        // the wait period has elapsed
                        if (dsiFailedCounts[newIndex] >= 1)
                        {
                                var current:int = getTimer();
                                if (current - failedDSI[newIndex] <
DEFAULT_WAIT_DURATION_AFTER_DOWN_SWITCH)
                                {
                                        CONFIG::LOGGING
                                        {
                                                debug("canAutoSwitchNow() - ignoring
switch request because index " + newIndex + " has " + dsiFailedCounts[newIndex]+" failure(s) and
only "+ (current - failedDSI[newIndex])/1000 + " seconds have passed since the last failure.");
                                        }
                                        return false;
                                }
                        }
                        // If the requested index is currently locked out, then we don't
                        // allow the switch.
                        else if (dsiFailedCounts[newIndex] >
DEFAULT_MAX_UP_SWITCHES_PER_STREAM_ITEM)
                        {
                                return false;
                        }

                        return true;
                }
```

```
/**
                 * The multiplier to apply to the maximum bandwidth for the client.  The
                 * default is 140% of the highest bitrate stream.
                 **/
                protected final function get bandwidthLimit():Number
                {
                        return _bandwidthLimit;
                }
                protected final function set bandwidthLimit(value:Number):void
                {
                        _bandwidthLimit = value;
                }

                // Internals
                //

                /**
                 * Executes the switch to the specified index.
                 *
                 *  @langversion 3.0
                 *  @playerversion Flash 10
                 *  @playerversion AIR 1.5
                 *  @productversion OSMF 1.0
                 */
                private function executeSwitch(targetIndex:int, force:Boolean = false):void
                {
                        var nso:NetStreamPlayOptions = new NetStreamPlayOptions();

                        var playArgs:Object =
NetStreamUtils.getPlayArgsForResource(dsResource);

                        nso.start = playArgs.start;
                        nso.len = playArgs.len;
                        nso.streamName = dsResource.streamItems[targetIndex].streamName;
                        nso.oldStreamName = oldStreamName;
                        if (force)
                        {
                                nso.transition = NetStreamPlayTransitions.RESET;
                        }
                        else
                        {
                                nso.transition = NetStreamPlayTransitions.SWITCH;
                        }

                        CONFIG::LOGGING
                        {
                                debug("executeSwitch() - Switching to index " + (targetIndex)
+ " at " + Math.round(dsResource.streamItems[targetIndex].bitrate) + " kbps");
                                logger.qos.ds.targetIndex = targetIndex;
                                logger.qos.ds.targetBitrate =
Math.round(dsResource.streamItems[targetIndex].bitrate);
                        }

                        switching = true;
                        switchingTimestamp = getTimer();

                        netStream.play2(nso);

                        oldStreamName = dsResource.streamItems[targetIndex].streamName;
```

```
if (targetIndex < actualIndex && autoSwitch)
                                {
                                        // This is a failure for the current stream, so let's tag it as such.
                                        incrementDSIFailedCount(actualIndex);

                                        // Keep track of when it failed so we don't try it again for
                                        // another failedItemWaitPeriod milliseconds to improve the
                                        // user experience.
                                        failedDSI[actualIndex] = getTimer();
                                }
                        }

                        /**
                         * Checks all the switching rules. If a switching rule returns -1, it is
                         * recommending no change.  If a switching rule returns a number greater than
                         * -1 it is recommending a switch to that index. This method uses the lesser of
                         * all the recommended indices that are greater than -1.
                         *
                         *  @langversion 3.0
                         *  @playerversion Flash 10
                         *  @playerversion AIR 1.5
                         *  @productversion OSMF 1.0
                         */
                        private function checkRules(event:TimerEvent):void
                        {
                                if (switchingRules == null || switching)
                                {
                                        CONFIG::LOGGING
                                        {
                                                var currentSwitchDuration:int = getTimer() -
switchingTimestamp;
                                                if (switching && currentSwitchDuration > 5000)
                                                {

                                                        logger.warn("Switch not complete after {0}
sec.", currentSwitchDuration / 1000);

                                                }
                                        }
                                        return;
                                }
                                var bufferRatio:Number = netStream.bufferLength /
netStream.bufferTime;
                                var newIndex:int = int.MAX_VALUE;

                                for (var i:int = 0; i < switchingRules.length; i++)
                                {
                                        var n:int =  switchingRules[i].getNewIndex();

                                        if (n != -1 && n < newIndex)
                                        {
                                                newIndex = n;
                                        }
                                }

                                if (    newIndex != -1
                                        &&      newIndex != int.MAX_VALUE
                                        &&      newIndex != actualIndex
                                  )
                                {
```

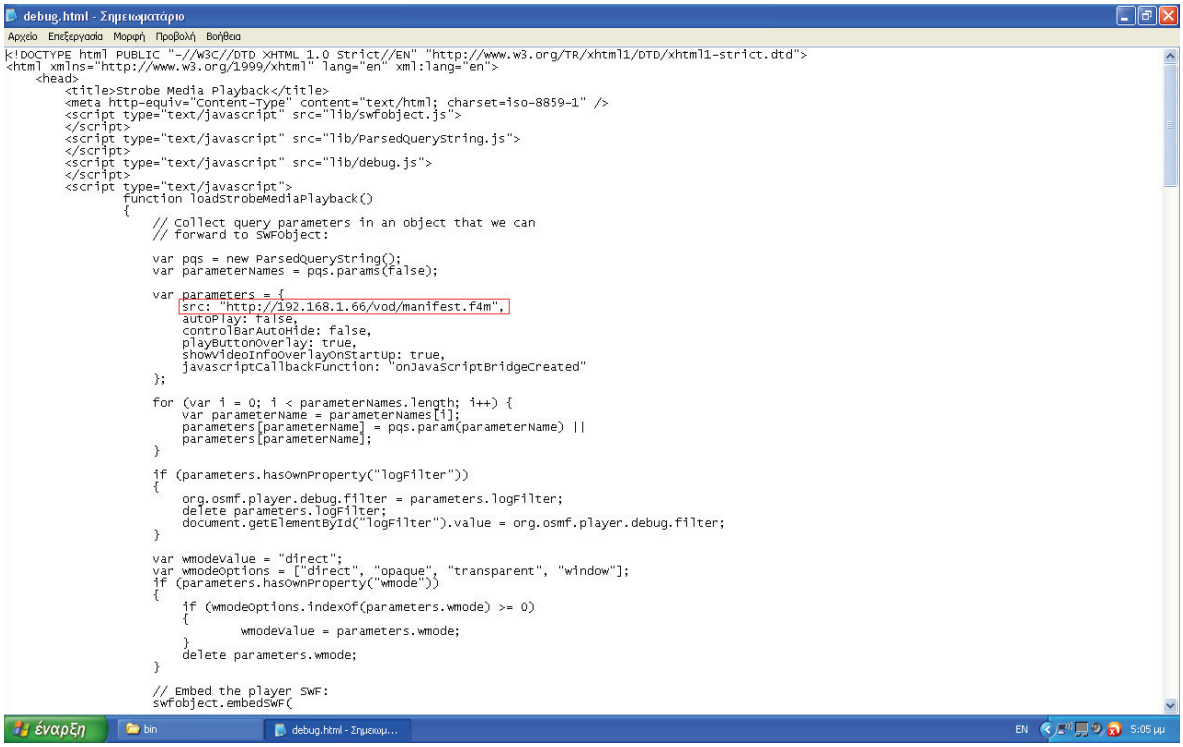
```
newIndex = Math.min(newIndex, maxAllowedIndex);
                        }

                        if (    newIndex != -1
                                &&      newIndex != int.MAX_VALUE
                                &&      newIndex != actualIndex
                                &&      !switching
                                &&      newIndex <= maxAllowedIndex
                                &&  canAutoSwitchNow(newIndex)
                                && (netStream.bufferTime == 0 || (newIndex < actualIndex
&& bufferRatio < 1) || (newIndex > actualIndex && bufferRatio > 1))
                          )
                        {
                                CONFIG::LOGGING
                                {
                                        debug("checkRules() - Calling for switch to " +
newIndex + " at " + dsResource.streamItems[newIndex].bitrate + " kbps");
                                }
                                executeSwitch(newIndex);
                        }
                }

                private function onNetStatus(event:NetStatusEvent):void
                {
                        CONFIG::LOGGING
                        {
                                debug("onNetStatus() - event.info.code=" + event.info.code);
                        }

                        switch (event.info.code)
                        {
                                case NetStreamCodes.NETSTREAM_PLAY_START:
                                        if (metrics.resource == null)
                                        {
                                                prepareForSwitching();
                                        }
                                        else if (autoSwitch && checkRulesTimer.running ==
false)
                                        {
                                                checkRulesTimer.start();
                                        }
                                        break;
                                case NetStreamCodes.NETSTREAM_PLAY_TRANSITION:
                                        switching  = false;
                                        actualIndex =
dsResource.indexFromName(event.info.details);
                                        metrics.currentIndex = actualIndex;
                                        lastTransitionIndex = actualIndex;
                                        break;
                                case NetStreamCodes.NETSTREAM_PLAY_FAILED:
                                        switching  = false;
                                        break;
                                case NetStreamCodes.NETSTREAM_SEEK_NOTIFY:
                                        switching  = false;
                                        if (lastTransitionIndex >= 0)
                                        {
                                                _currentIndex = lastTransitionIndex;
                                        }
                                        break;
```

```
case NetStreamCodes.NETSTREAM_PLAY_STOP:
                                        checkRulesTimer.stop();
                                        CONFIG::LOGGING
                                        {
                                                debug("onNetStatus() - Stopping rules since
server has stopped sending data");
                                        }
                                        break;
                        }
                }

                private function onPlayStatus(info:Object):void
                {
                        CONFIG::LOGGING
                        {
                                debug("onPlayStatus() - info.code=" + info.code);
                        }

                        switch (info.code)
                        {
                                case
NetStreamCodes.NETSTREAM_PLAY_TRANSITION_COMPLETE:
                                        if (lastTransitionIndex >= 0)
                                        {
                                                _currentIndex = lastTransitionIndex;
                                                lastTransitionIndex = -1;
                                        }

                                        CONFIG::LOGGING
                                        {
                                                debug("onPlayStatus() - Transition complete
to index: " + currentIndex + " at " + Math.round(dsResource.streamItems[currentIndex].bitrate) + "
kbps");
                                        }

                                        break;
                        }
                }

                /**
                 * Prepare the manager for switching.  Note that this doesn't necessarily
                 * mean a switch is imminent.
                 **/
                private function prepareForSwitching():void
                {
                        initDSIFailedCounts();

                        metrics.resource = dsResource;

                        actualIndex = 0;
                        lastTransitionIndex = -1;

                        if ((dsResource.initialIndex >= 0) && (dsResource.initialIndex <
dsResource.streamItems.length))
                        {
                                actualIndex = dsResource.initialIndex;
                        }
```

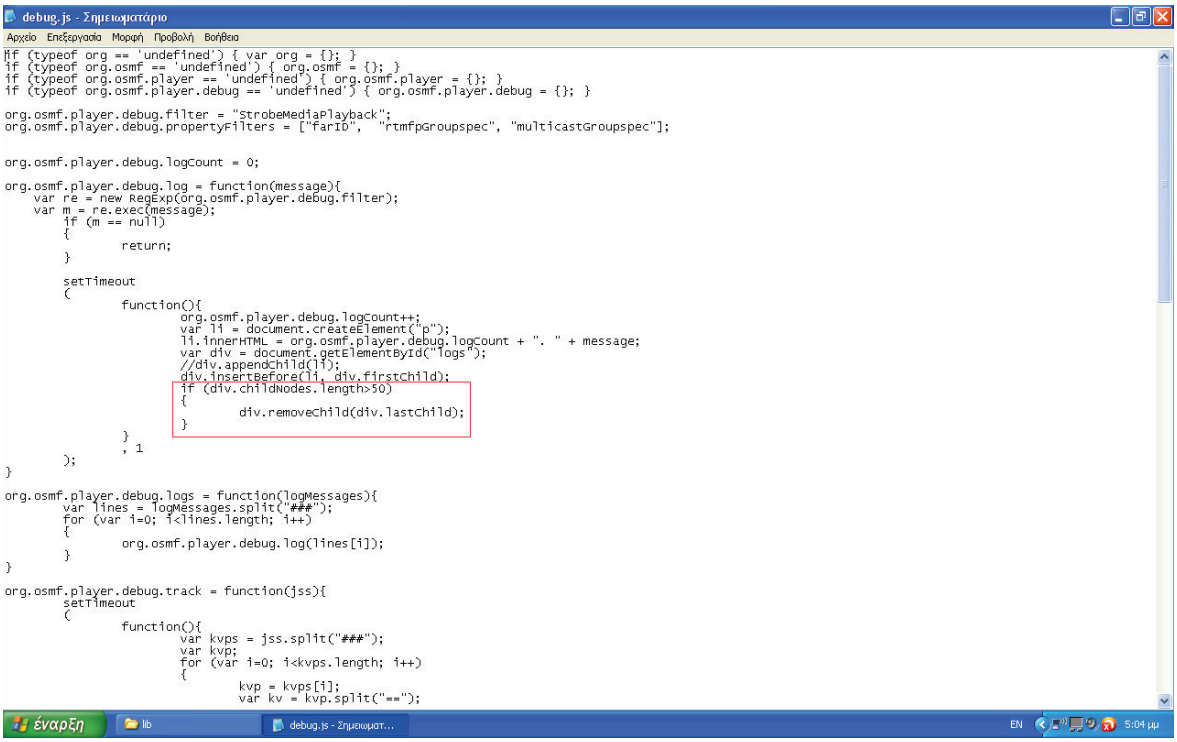
```
if (autoSwitch)
                                {
                                        checkRulesTimer.start();
                                }

                                setThrottleLimits(dsResource.streamItems.length - 1);
                                CONFIG::LOGGING
                                {
                                        debug("prepareForSwitching() - Starting with stream index " +
actualIndex + " at " + Math.round(dsResource.streamItems[actualIndex].bitrate) + " kbps");
                                }
                                metrics.currentIndex = actualIndex;
                        }

                        private function initDSIFailedCounts():void
                        {
                                if (dsiFailedCounts != null)
                                {
                                        dsiFailedCounts.length = 0;
                                        dsiFailedCounts = null;
                                }

                                dsiFailedCounts = new Vector.<int>();
                                for (var i:int = 0; i < dsResource.streamItems.length; i++)
                                {
                                        dsiFailedCounts.push(0);
                                }
                        }

                        private function incrementDSIFailedCount(index:int):void
                        {
                                dsiFailedCounts[index]++;

                                // Start the timer that clears the failed counts if one of them
                                // just went over the max failed count
                                if (dsiFailedCounts[index] >
DEFAULT_MAX_UP_SWITCHES_PER_STREAM_ITEM)
                                {
                                        if (clearFailedCountsTimer == null)
                                        {
                                                clearFailedCountsTimer = new
Timer(DEFAULT_CLEAR_FAILED_COUNTS_INTERVAL, 1);

        clearFailedCountsTimer.addEventListener(TimerEvent.TIMER, clearFailedCounts);
                                        }

                                        clearFailedCountsTimer.start();
                                }
                        }

                        private function clearFailedCounts(event:TimerEvent):void
                        {
                                clearFailedCountsTimer.removeEventListener(TimerEvent.TIMER,
clearFailedCounts);
                                clearFailedCountsTimer = null;
                                initDSIFailedCounts();
                        }
```

```
private function setThrottleLimits(index:int):void
                    {
                              connection.call("setBandwidthLimit", null, _bandwidthLimit,
_bandwidthLimit);
                    }

                    CONFIG::LOGGING
                    {
                    private function debug(...args):void
                    {
                              //trace(new Date().toTimeString() + ">>> NetStreamSwitchManager." +
args);
                              logger.debug(new Date().toTimeString() + ">>>
NetStreamSwitchManager." + args);
                    }
                    }

                    private var netStream:NetStream;
                    private var dsResource:DynamicStreamingResource;
                    private var switchingRules:Vector.<SwitchingRuleBase>;
                    private var metrics:NetStreamMetricsBase;
                    private var checkRulesTimer:Timer;
                    private var clearFailedCountsTimer:Timer;
                    private var actualIndex:int = -1;
                    private var oldStreamName:String;
                    private var switching:Boolean;
                    private var switchingTimestamp:int;
                    private var _currentIndex:int;
                    private var lastTransitionIndex:int = -1;
                    private var connection:NetConnection;
                    private var dsiFailedCounts:Vector.<int>;              // This vector keeps track
of the number of failures

                              // for each DynamicStreamingItem in the DynamicStreamingResource
                    private var failedDSI:Dictionary;
                    private var _bandwidthLimit:Number = 0;;


                    private static const RULE_CHECK_INTERVAL:Number = 2500;        //
Switching rule check interval in milliseconds
                    private static const
DEFAULT_MAX_UP_SWITCHES_PER_STREAM_ITEM:int = 3;
                    private static const
DEFAULT_WAIT_DURATION_AFTER_DOWN_SWITCH:int = 30000;
                    private static const
DEFAULT_CLEAR_FAILED_COUNTS_INTERVAL:Number = 300000;        // default of 5
minutes for clearing failed counts on stream items

                    CONFIG::LOGGING
                    {
                    //          private static const logger:Logger =
Log.getLogger("org.osmf.net.NetStreamSwitchManager");
                              protected var logger:StrobeLogger =
Log.getLogger("StrobeMediaPlayback") as StrobeLogger;
                    }
          }
}
```

Τέλος αφού χτιστεί ο κώδικάς θα πρέπει να τροποποιηθούν κάποιες απαραίτητες λεπτομέρειες για να μπορέσει να λειτουργήσει σωστά το player. Θα πρέπει να γίνει αλλαγή της διεύθυνσης ip μαζί με την διαδρομή του manifest file στον κώδικα του debug.html που βρίσκεται στο φάκελο bin(εικόνα 29) στην διεύθυνση του Server καθώς επίσης και μία μικρή αλλαγή στον κώδικα debug.js που βρίσκεται στο \bin\lib ώστε να μην κρατάει μόνο τα τελευταία 50 συμβάντα αλλά απεριόριστα(εικόνα 30) με την διαγραφή του συγκεκριμένου τμήματος του κώδικα.

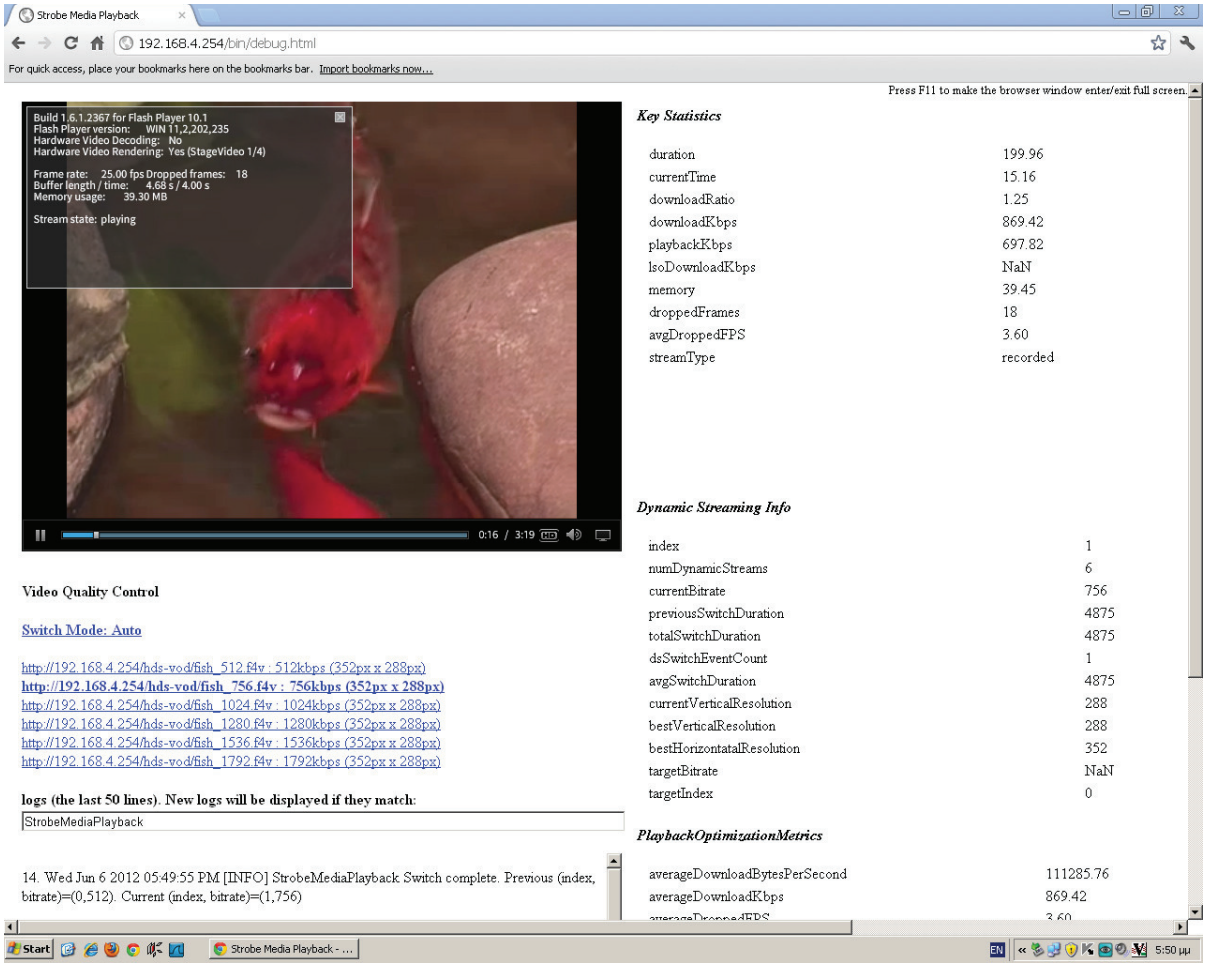


(Εικόνα 29: διαδρομή του manifest file)

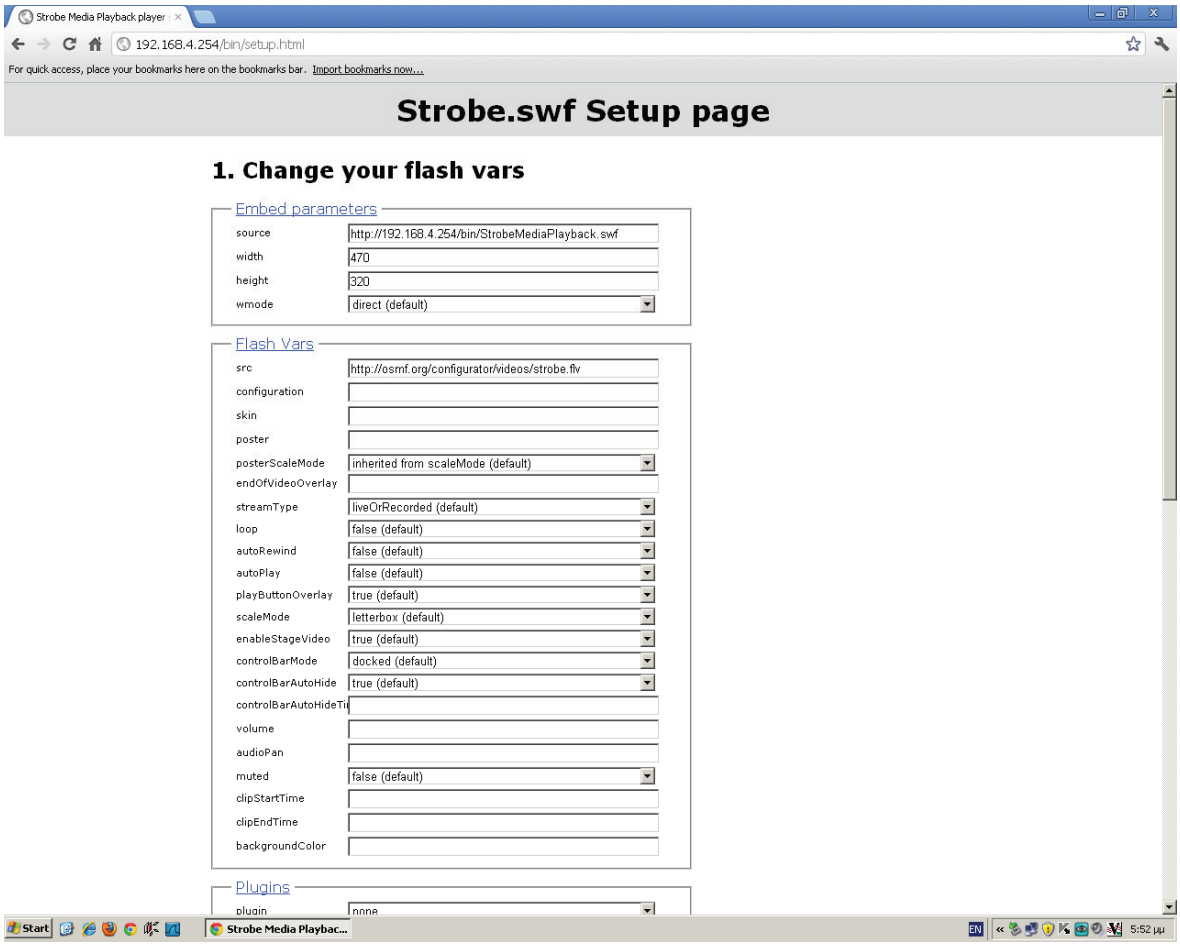


(Εικόνα 30: απεριόριστα συμβάντα)

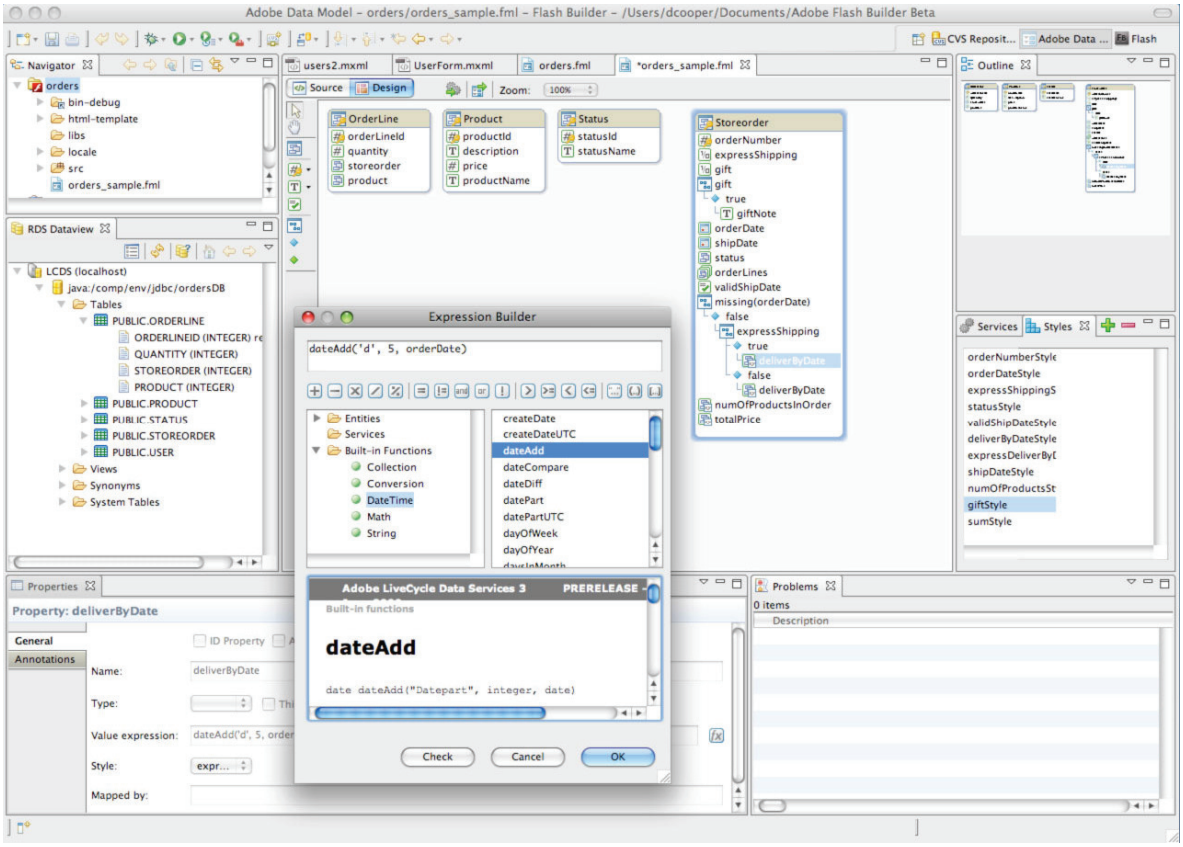
(Εικόνα 31: Flash Media Server 4.5)



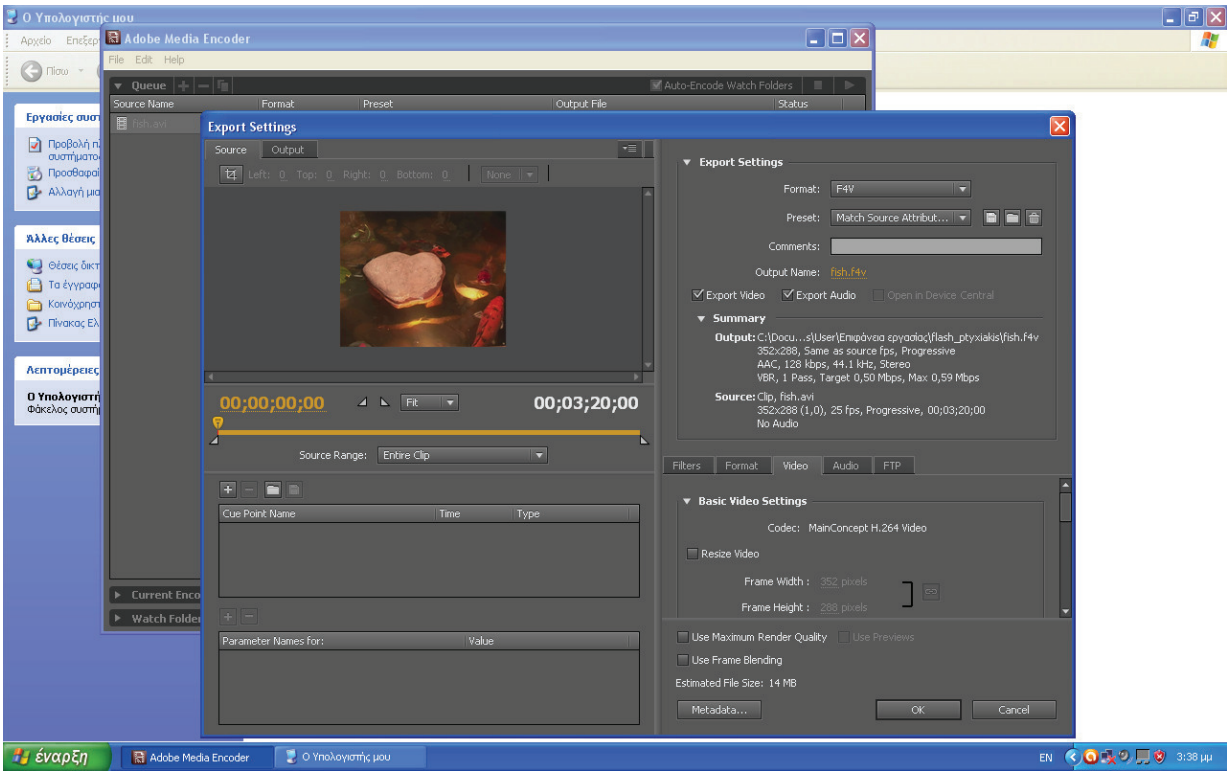
(Εικόνα 32: StrobeMediaPlayback debug player)



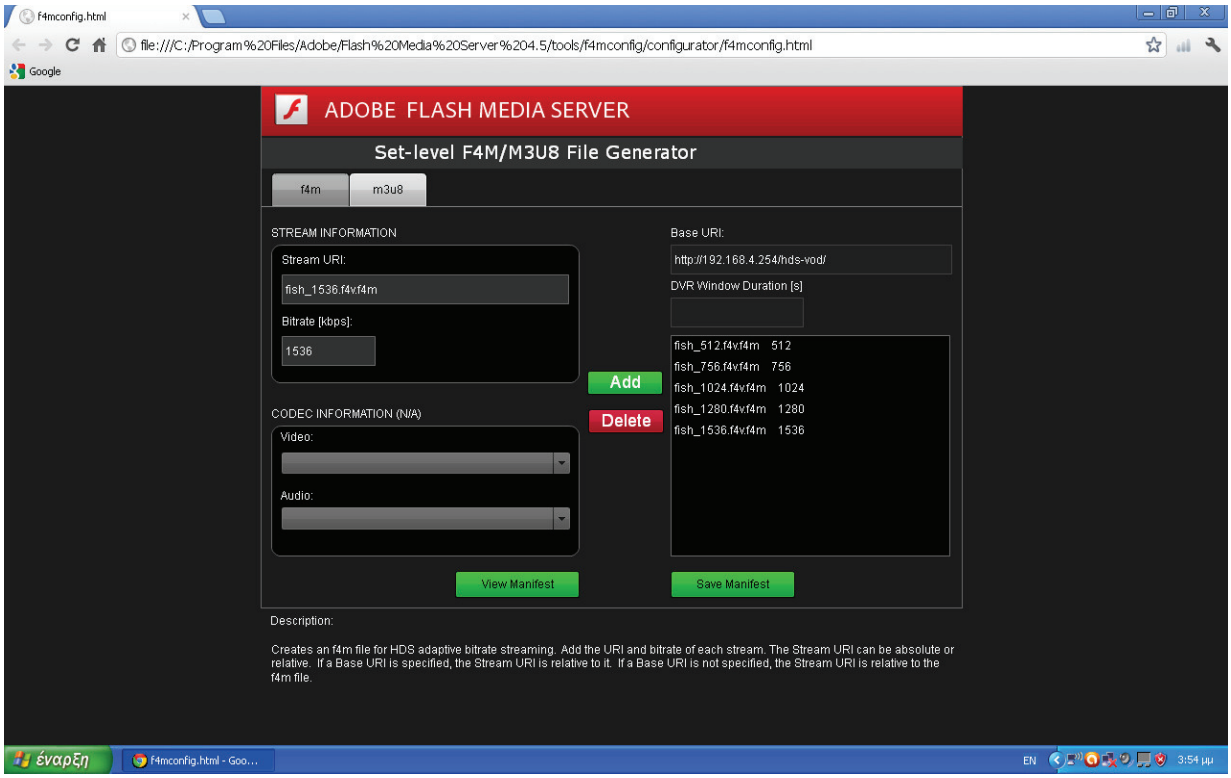
(Εικόνα 33: Παράμετροι StrobeMediaPlayback debug player)



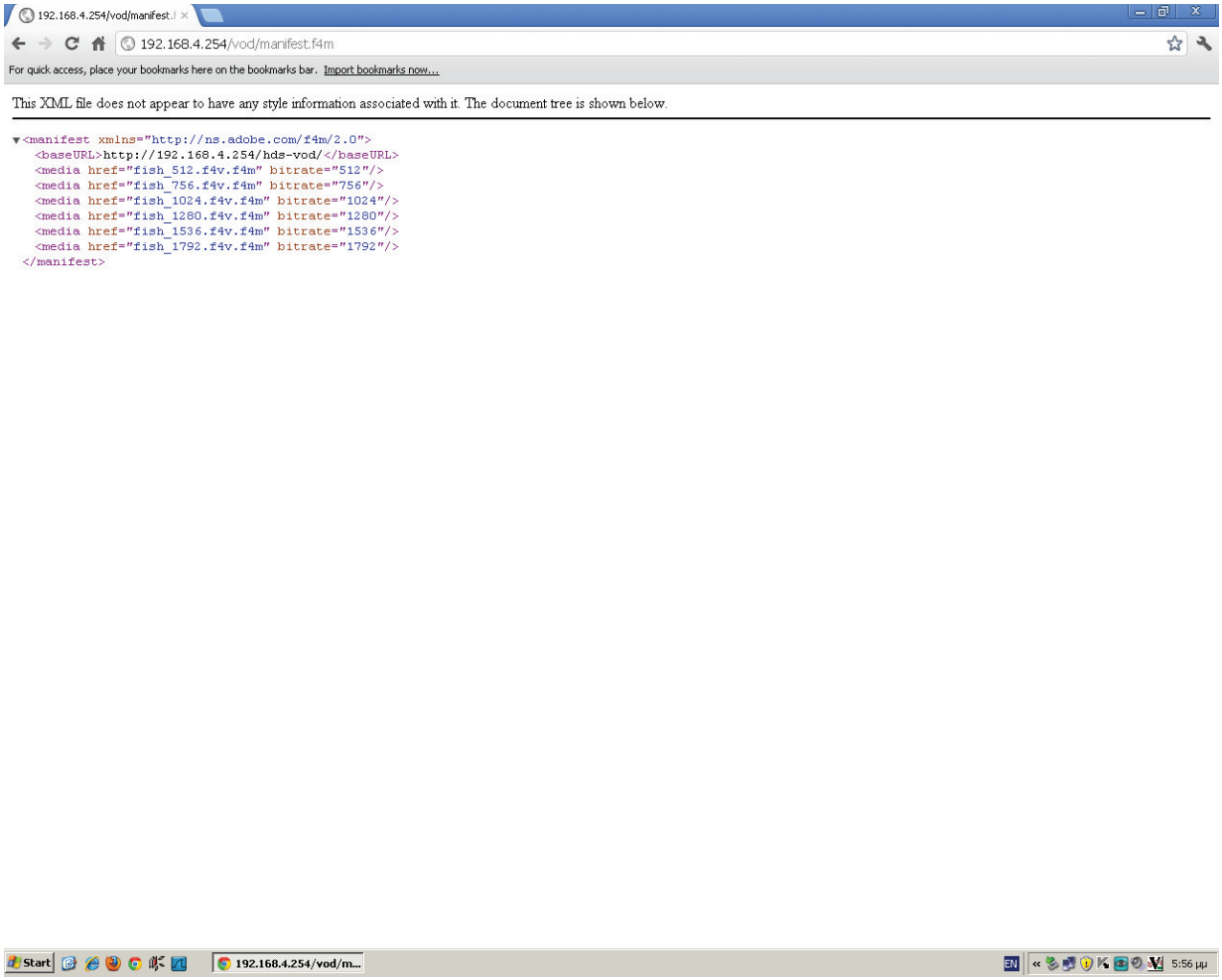
(Εικόνα 34: Adobe Flash Builder 4.5)



(Εικόνα 35: Adobe Media Encoder CS5.5)



(Εικόνα 36: F4M File Generator)



(Εικόνα 37: manifest file)