Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά Συστήματα

Προσαρμογή Υπηρεσιών Video με βάση το πρωτόκολλο HTTP σε ασύρματα δίκτυα

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΥΡΙΑΖΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : Phd ΑΝΑΣΤΑΣΙΟΣ ΝΤΑΓΙΟΥΚΛΑΣ

Ιούνιος 2013

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Προσαρμογή Υπηρεσιών Video με βάση το πρωτόκολλο HTTP σε ασύρματα δίκτυα

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΥΡΙΑΖΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : Phd ΑΝΑΣΤΑΣΙΟΣ ΝΤΑΓΙΟΥΚΛΑΣ

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Ιούνιος 2013

Περίληψη :

Η συνεχής αύξηση των χρηστών στο διαδίκτυο αλλά και τοπικών δικτύων έχουν πλέον βάλει τη μετάδοση βίντεο μέσω IP δικτύων στην καθημερινότητά μας σε πολύ μεγάλο βαθμό. Μικρά βίντεο ή μεγαλύτερα, ακόμα και ολόκληρες ταινίες μας δίνουν τη δυνατότητα να ενημερωθούμε, να ψυχαγωγηθούμε, να διασκεδάσουμε με την παρέα μας ακόμα και να διδαχθούμε αν εκπαιδευόμαστε σε κάποιο τομέα. Στα πλαίσια της πτυχιακής εργασίας μελετήθηκε το πρωτόκολλο HTTP και πώς γίνεται η μετάδοση υπηρεσιών video streaming μέσα από το μηχανισμό DASH. Τα συμπεράσματα που έχουν εξαχθεί μπορεί να έχουν ευρεία γκάμα ανταπόκρισης. Μπορούν να χρησιμοποιηθούν για πανεπιστημιακή, εμπορική ή όποια άλλη χρήση.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κύριο Αναστάσιο Νταγιούκλα για την αμέριστη υποστήριξή του καθώς και το Ανοιχτό Πανεπιστήμιο Κύπρου που μου έδωσε την ευκαιρία να αναπτύξω ένα τέτοιο θέμα και να το θέσω προ πάσα χρήση στη διάθεση της πανεπιστημιακής κοινότητας

Περιεχόμενα

Εισαγωγή

Σε αυτή την εργασία πρόκειται να ασχοληθούμε με τη μελέτη του video streaming , μια δυνατότητα που προσφέρεται και τη χρησιμοποιούν όλο και περισσότεροι χρήστες του διαδικτύου σε όλον τον κόσμο. Οι ευρυζωνικές ταχύτητες αλλά και οι συσκευές σταθερές και κινητές έχουν δώσει μια τεράστια ώθηση και σε αυτόν τον τομέα. Οι τεχνολογίες πολλές που επιχειρούν να δώσουν λύσεις στην πολυπλοκότητα του όλου εγχειρήματος που παρουσιάζει ακόμα και σήμερα προβλήματα στη χρήση και την ποιότητα των υπηρεσιών. Στην παρούσα εργασία θα πειραματιστούμε με την τεχνολογία DASH όπως την προσφέρει η πλατφόρμα της Adobe.

Πιο συγκεκριμένα στο κεφάλαιο 1 θα μιλήσουμε για εισαγωγικές έννοιες, όπως οι προοπτικές που ανοίγονται με τις νέες τεχνολογίες, τα βασικά πρωτόκολλα που μας ενδιαφέρουν, ασύρματη ζεύξη, τρόποι μετάδοσης δεδομένων, εφαρμογές στην καθημερινότητα που βρίσκουν ανταπόκριση

Στη συνέχεια στο κεφάλαιο 2 θα γίνει μια αναφορά στις online μεθόδους αναπαραγωγής βίντεο για να περάσουμε στο κεφάλαιο 3 και να αναπτύξουμε το μηχανισμό DASH που ξεχωρίσαμε να πλεονεκτεί έναντι των άλλων. Στο κεφάλαιο 4 γίνεται μια αναφορά στις πλατφόρμες που ασχολούνται με την τεχνολογία DASH ενώ τέλος στο κεφάλαιο 5 μιλάμε επί του πρακτέου πλέον για την πλατφόρμα της Adobe και μέσα από ένα πείραμα βγάζουμε χρήσιμα συμπεράσματα.

Κεφάλαιο 1

1.1 Νέες Τεχνολογίες, νέες προοπτικές

Το Διαδίκτυο επιβάλλει νέες προκλήσεις για streaming video δεδομένου ότι καλείτε να προσφέρει την καλύτερη δυνατή υπηρεσία. Αν και το διαδίκτυο δεν προβλέπει τον έλεγχο εισόδου, κάποιοι μηχανισμοί πρέπει να αναπτυχθούν για την αποφυγή συμφόρησης του δικτύου.

Η μεταφορά των δεδομένων για την οποία είναι υπεύθυνο το πρωτοκόλο TCP είναι υπεύθυνο για την αποφυγή της κυκλοφοριακής συμφόρησης.
Ένα παράδειγμα για το πώς δουλεύει το πρωτόκολλο TCP με βίντεο συνεχούς ροής στο Διαδίκτυο είναι τα βίντεο της πύλης YouTube, στο οποίο ανεβάζουν, παρακολουθούν, σχολιάζουν και διαδίδουν σε σελίδες κοινωνικής δικτύωσης, εκατομμύρια χρήστες από όλο τον κόσμο. Στο πλαίσιο που γίνεται η αναπαραγωγή του βίντεο, πρέπει να αντιμετωπιστεί η πακέτο-απώλεια που μπορεί να προκύψει λόγω της κυκλοφοριακής συμφόρησης.

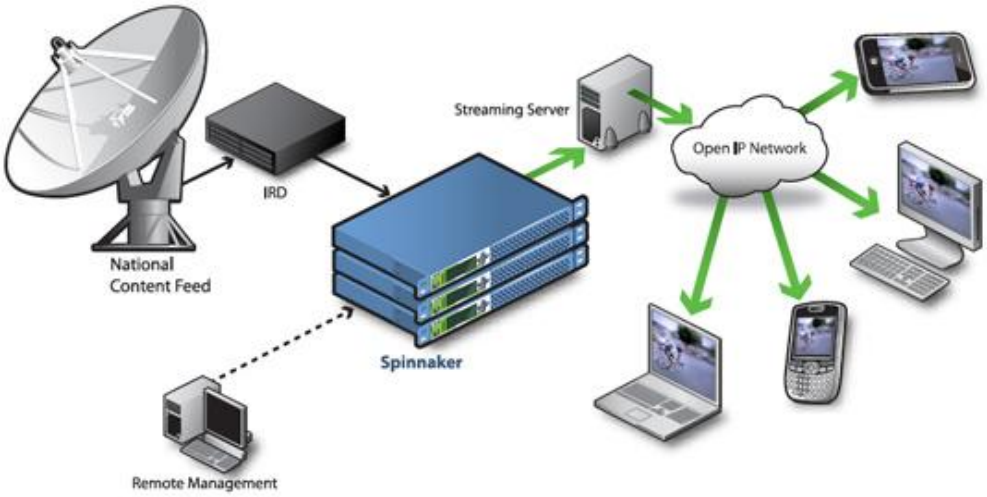
Στην περίπτωση αυτή, έχουμε επικεντρωθεί σε δίκτυα πρόσβασης τα οποία θεωρείται ότι αποτελούν τους συνδέσμους συμφόρησης. Τη στιγμή της προσαρμογής στις διαφορετικές συνθήκες του δικτύου γίνεται, με το χρήστη να επιλέγει από διαφορετικές ποιότητες του περιεχομένου, που κυμαίνονται από τη χαμηλά ανάλυση βίντεο σε HD (π.χ., 720p). Παρόλο που ο χρήστης έχει επιλέξει την ποιότητα του βίντεο, με βάση την εμπειρία του μπορεί να είναι δυσκίνητο και επιρρεπές σε λάθη, γιατί το διαθέσιμο εύρος ζώνης δεν είναι σταθερό. Το Adaptive video streaming έχει ως στόχο να απλοποιήσει αυτή τη διαδικασία με τη συνεχή προσαρμογή του bit rate περιεχομένου στο διαθέσιμο εύρος ζώνης του δικτύου.

Το πρωτόκολλο HTTP βασίζεται στον να προσαρμόσει το streaming video για χρήση στο Διαδίκτυο. Ειδικότερα, διερευνήθηκε η απόδοση πολλαπλών παράλληλων HTTP με βάση το αίτημα-απάντηση και οι ροές για streaming video και η προσαρμογή αυτών σε ασύρματα δίκτυα. Είναι γεγονός ότι παρουσιάζονται πολλαπλές ροές με ευνοϊκά χαρακτηριστικά σε περίπτωση απώλειας πακέτου ώστε να μετριάσουν τα αποτελέσματα της σύνδεσης, TCP timeouts , διατηρώντας παράλληλα TCP-φιλικότητα.

Το Dynamic Streaming Adaptive μέσω HTTP (DASH), αναφέρεται σε μια μέθοδο μεταφοράς βίντεο, όπου οι πελάτες (clients) προσαρμόζουν τα αιτήματα με βάση ορισμένες εκτιμήσεις του διαθέσιμου ρυθμού λήψης τους σε κάθε στιγμή της υπηρεσία streaming.

Με το DASH ένα βίντεο προσφέρεται σε διάφορες (συνήθως 4 έως 10) εκδόσεις. Κάθε τερματικό μπορεί να επιλέξει ποια έκδοση να κάνει λήψη ανάλογα με τις δυνατότητές του και το επίπεδο συμφόρησης του δικτύου. Αυτή η επιλογή δεν ισχύει μόνο κατά την έναρξη της ροής, αλλά συχνά μέσα από διασκορπισμένες στιγμές του χρόνου κατά τη διάρκεια της ροής του βίντεο, στο οποίο ο πελάτης μπορεί να εναλλαχθεί από τη μία έκδοση στην άλλη. Συνήθως αυτό επιτυγχάνεται με την κατάτμηση κάθε

εκδοχής σε κομμάτια, έτσι ώστε τα όρια των τμημάτων να ευθυγραμμίζονται κάθε φορά.

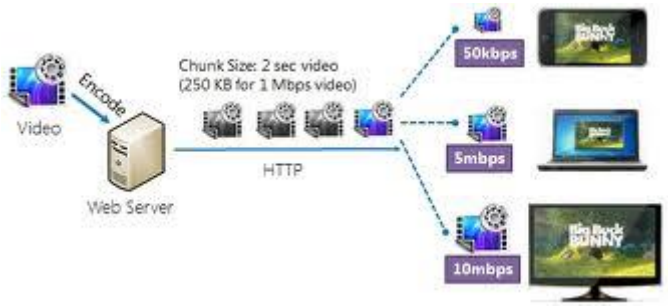


1.1.1 Βλέποντας τηλεόραση στο μέλλον….

Ένα μη αμφισβητήσιμο γεγονός από έρευνες που έχουν γίνει είναι ότι στην Αμερική οι τηλεθεατές ξοδεύουν το χρόνο τους μπροστά από μια τηλεόραση πέντε ώρες την ημέρα κατά μέσο όρο. Η πλειοψηφία ενώ προτιμάει τα αναμεταδιδόμενα προγράμματα με τον παραδοσιακό τρόπο υπάρχει μια σταθερά αυξανόμενη μερίδα που προτιμάει να παρακολουθεί τηλεόραση χρησιμοποιώντας ψηφιακά μέσα.

Μια άλλη μέτρηση έρχεται να μας δείξει πως το 1/3 των Αμερικανών καταναλωτών χρησιμοποιούν μια ψηφιακή συσκευή εγγραφής video (DVR) γιατί προτιμούν να έχουν τη δυνατότητα να «γράψουν» την αγαπημένη τους εκπομπή και να την παρακολουθήσουν όποτε αυτοί επιθυμούν.

Από την άλλη μεριά υπάρχει η τρομακτική έκρηξη του διαδικτύου που δε θα μπορούσε να αφήσει ανεπηρέαστο τον τομέα της τηλεόρασης, video, πολυμέσων. Στη σημερινή εποχή μπορεί ο καθένας να έχει πρόσβαση στο διαδίκτυο από τον υπολογιστή του, σταθερό ή φορητό, ένα tablet και να παρακολουθήσει το πρόγραμμα, video της αρεσκείας του. Όλο και πιο πολύ ο παγκόσμιος ιστός ενσωματώνει την ψηφιακή τηλεόραση δίνοντας δυνατότητες απίστευτες όπως το κατέβασμα ταινιών ή συνεχούς ροής με ειδικά πρωτόκολλα που επιλογές που τα προηγούμενα χρόνια μόνο όποιος είχε φαντασία μπορούσε να σκεφτεί.



Παράδειγμα video streaming

Ένα ενδεικτικό παράδειγμα που απεικονίζει ξεκάθαρα την τάση που επικρατεί στην αγορά είναι τα μέσα που προβάλλονται μεγάλες διεθνείς διοργανώσεις και το πώς επιλέγει ο τηλεθεατής να τις παρακολουθήσει. Για παράδειγμα στο παγκόσμιο κύπελλο ποδοσφαίρου το 2010 25 αγώνες από τους συνολικά 64 που παίχθηκαν προβλήθηκαν με δυνατότητα 3D θέασης . Το πιο ενδιαφέρον είναι ότι περισσότεροι χρήστες από ποτέ παρακολούθησαν τους αγώνες μέσω web χρησιμοποιώντας streaming τεχνολογίες με HD ή περίπου HD ποιότητα εικόνα μέσα από υπολογιστές, ταμπλέτες, έξυπνα κινητά, τηλεοράσεις που ήταν συνδεδεμένες στο διαδίκτυο. Το ίδιο φαινόμενο παρατηρήθηκε και στους χειμερινούς αγώνες του 2010 στο Βανκούβερ του Καναδά. Πάνω από 4 εκατομμύρια τηλεθεατές παρακολούθησαν τους αγώνες μέσω web με τους τρόπους που αναφέρθηκαν παραπάνω.

Εκτός τις μεγάλες αθλητικές διοργανώσεις αρκετοί μεγάλοι παροχείς προσφέρουν υπηρεσίες σε εκατομμύρια τηλεθεατές σε όλο τον κόσμο με αναμετάδοση βίντεο, ζωντανό ή μη, ειδήσεων ταινιών, εκπομπών και ότι άλλο περιλαμβάνει ένα κλασικό τηλεοπτικό πρόγραμμα. Οι επιλογές που δίνονται είναι τόσες πολλές και ελκυστικές που δείχνουν να ικανοποιούν και τον πλέον απαιτητικό τηλεθεατή.

Τα βίντεο στο διαδίκτυο χωρίζονται σε τρεις κατηγορίες. α) αυτά που κάνουν απλοί χρήστες ερασιτεχνικά ή ημιεπαγγελματικά και τα μοιράζονται με άλλους χρήστες του διαδικτύου όπως στο YouTube και β) τα άκρως επαγγελματικά βίντεο που γυρίζονται με τις αντίστοιχες υποδομές όπως studios, κάμερες, επαγγελματικά συνεργεία και προβάλλονται αντίστοιχα επαγγελματικά με παροχείς όπως ο ABC.com, hulu.com και γ) ταινίες ολόκληρες που γυρίζονται και προβάλλονται μέσω διαδικτύου, αυτό χαρακτηρίζεται ως EST (Electronic Sell-Through). Netflix, Apple TV και Ultraviolet είναι οι πιο γνωστοί πάροχοι για αυτή την κατηγορία ταινιών.

1.2 Πρωτόκολλα επικοινωνίας

1.2.1 HTTP

Με δυο λόγια :

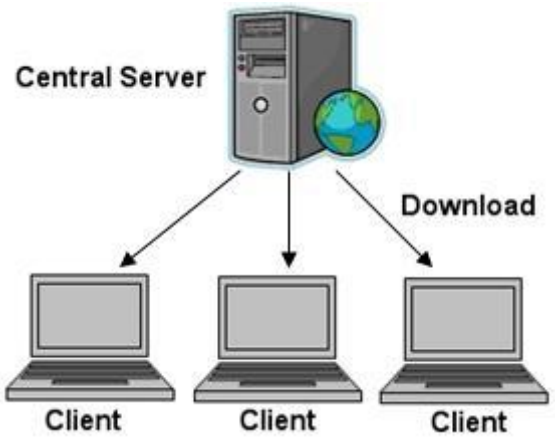
Το HTTP πρωτόκολλο είναι συντομογραφία του «HyperText Transfer Protocol». Περιλαμβάνει ένα σύνολο κανόνων, όπως άλλωστε δηλώνουν τα πρωτόκολλα, που καθορίζει τον τρόπο με τον οποίο θα γίνει η μεταφορά του υπερκειμένου (hypertext) μεταξύ δύο ή περισσοτέρων υπολογιστών.

Το πρωτόκολλο αυτό ανήκει στο έβδομο επίπεδο OSI, αυτό των εφαρμογών και είναι το πιο συνηθισμένο στον ηλεκτρονικό χώρο του World Wide Web. Δουλεύει πάνω από το TCP/IP πρωτόκολλο

Είναι γενικό και χρησιμοποιείται σε ένα πλήθος εφαρμογών, για παράδειγμα σε εξυπηρέτες-διανομείς (servers) και κατανεμημένα συστήματα διαχείρισης αντικειμένων. Το βασικότερο και πιο σημαντικό χαρακτηριστικό του είναι ότι δεν επηρεάζει τις διάφορες τεχνολογίες μετάδοσης δεδομένων ανεξάρτητα από τα δεδομένα που αυτές μεταφέρουν.

Πιο ειδικά μπορούμε να πούμε ότι, όπως όλες οι υπηρεσίες του διαδικτύου έτσι και η υπηρεσία WWW στηρίζεται στο μοντέλο client/server. Αυτό σημαίνει, πως το

σύνολο των πληροφοριών βρίσκεται σε κάποιον υπολογιστή με το κατάλληλο λογισμικό που ονομάζεται server που εξυπηρετεί κλήσεις ανάσυρσης. Οι κλήσεις αποστέλλονται από το server στον υπολογιστή που τις έχει αιτηθεί, με το κατάλληλο λογισμικό ονομάζεται client.



Το μοντέλο Client/server

Στην υπηρεσία WWW ο server ονομάζεται **Web server** και ο client ονομάζεται **Web client** ή **Web browser**

Το πρωτόκολλο που χρησιμοποιείται για τη μεταφορά των δεδομένων

και των κλήσεων από τον Web server στον Web browser (και αντίστροφα) ονομάζεται HTTP (HyperText Transfer Protocol).

Το πρωτόκολλο HTTP επιτρέπει στον υπολογιστή Χ (τον πελάτη-client) να

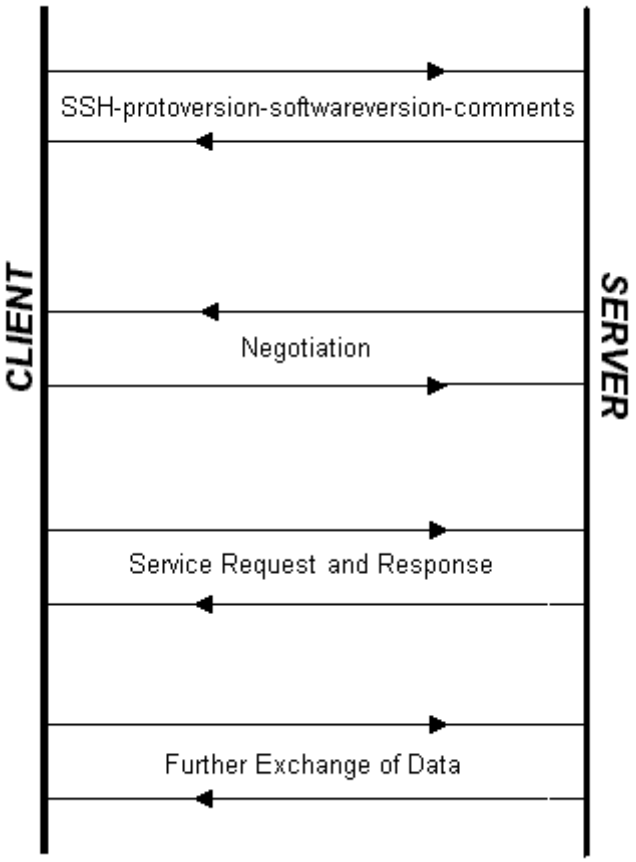
αιτηθεί πληροφορίες από το server Υ. Αυτό σημαίνει ότι θα πραγματοποιήσει μια σύνδεση με τον υπολογιστή Υ (τον διανομέα- server) ώστε να υλοποιηθεί η αίτησή του. Ο server αποδέχεται την σύνδεση, αν φυσικά δεν υπάρχει κάποιο πρόβλημα, που ξεκίνησε από τον client και στέλνει πίσω μια απάντηση. Η HTTP αίτηση αναζητά και βρίσκει την πηγή για την οποία ο client ενδιαφέρεται και λέει στο server ποια ενέργεια να κάνει αναφορικά με αυτή την πηγή. Η αίτηση μπορεί να αφορά video, εικόνες, έγγραφα, εφαρμογές και ότι άλλο υπάρχει στο διαδίκτυο.

1.2.2 HTTPS

Το **HTTPS** *(Hypertext Transfer Protocol Secure)* χρησιμοποιείται στα δίκτυα για να δηλώσει μία ασφαλή http σύνδεση. Ένας σύνδεσμος (URL) που αρχίζει με το πρόθεμα https υποδηλώνει ότι θα χρησιμοποιηθεί κανονικά το πρωτόκολλο HTTP, με τη διαφορά ότι η σύνδεση γίνεται από διαφορετική πόρτα την 443 αντί 80 και τα δεδομένα θα ανταλλάσσονται κρυπτογραφημένα με το πρωτόκολλο Secure Socket Layer

Η κρυπτογράφηση που χρησιμοποιείται διασφαλίζει ότι τα κρυπτογραφημένα δεδομένα δεν θα μπορούν να υποκλαπούν από άλλους κακόβουλους χρήστες ή επιθέσεις

Για να εφαρμοσθεί το HTTPS σε έναν εξυπηρέτη, θα πρέπει ο διαχειριστής του να εκδώσει ένα πιστοποιητικό κλειδιού δημόσιου. Σε εξυπηρέτες που χρησιμοποιούν το λειτουργικό σύστημα Linux αυτό μπορεί να γίνει μέσω του προγράμματος OpenSSL.



Το μοντέλο Client/Server με το πρωτόκολλο HTTPS

1.3 Ασύρματα δίκτυα

Ασύρματο χαρακτηρίζεται ένα δίκτυο όπου δεν υπάρχει φυσική ζεύξη (καλώδιο) μεταξύ των πόρων του με ότι πλεονεκτήματα, μειονεκτήματα συνεπάγεται αυτό.

Στις μέρες μας μπορούμε να μιλάμε για ένα τοπικό δίκτυο όπου συνδέονται όλα ασύρματα αλλά και για ένα δίκτυο πολύ μεγαλύτερης εμβέλειας όπως αυτό της κινητής τηλεφωνίας. Σε κάθε περίπτωση όμως αυτό που χρειάζεται είναι να υπάρχουν κόμβος ή κόμβοι που θα μοιράζουν την πληροφορία και άλλοι που απλά έχουν το ρόλο του αιτούντα.

Οικιακό Δίκτυο wifi

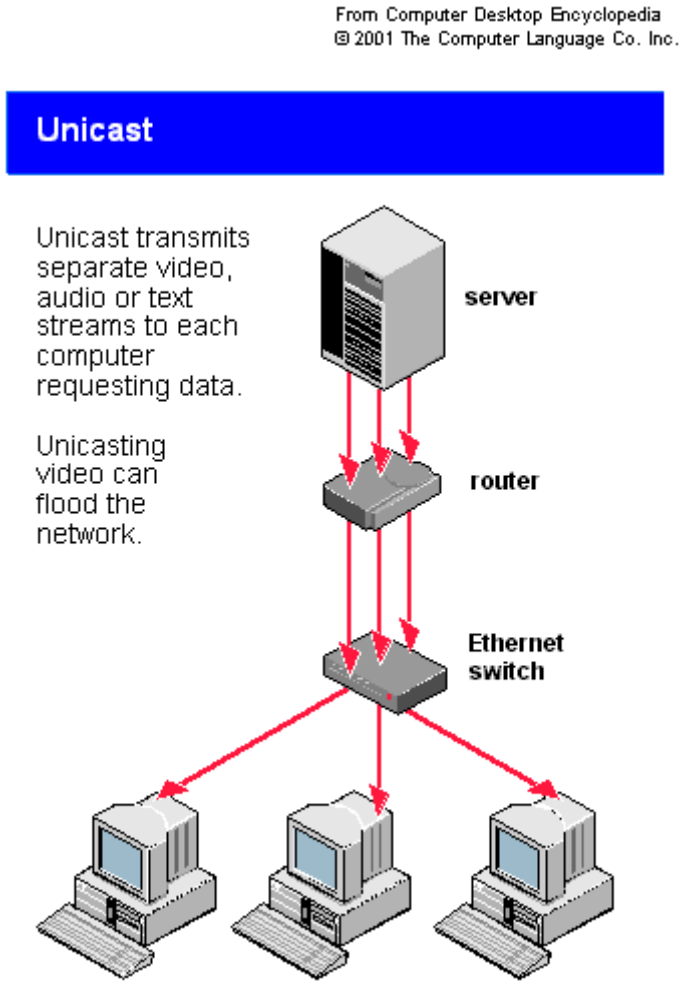
Μεγαλύτερο γεωγραφικό δίκτυο ασύρματης ζεύξης

Οι ασύρματες ζεύξεις έχουν μεγάλη εφαρμογή στις μέρες μας κυρίως στην κινητή τηλεφωνία όπου συνδέονται εκατομμύρια χρήστες σε όλο τον κόσμο και μπορούν να εξυπηρετηθούν από σε υπηρεσίες φωνής, μηνυμάτων, διαδικτύου βίντεο με έξυπνα τηλέφωνα (smart phones) και ταμπλέτες (tablets).

1.4 Τρόποι μετάδοσης δεδομένων

1.4.1 Unicast μετάδοση

Η πρώτη και η συνηθέστερη για πολύ καιρό τεχνική μετάδοσης ήταν αυτή της Unicast. Με αυτή την τεχνική ο πομπός έχει τη δυνατότητα να στέλνει data σε έναν και μόνο δέκτη κάθε στιγμή κάτι που έχει ονομαστεί και point to point σύνδεση. Αν για παράδειγμα θέλήσει κάποιος πομπός να στείλει την ίδια πληροφοριά σε πολλούς δέκτες τότε θα πρέπει να στείλει την ίδια πληροφορία N φορές όσοι και οι δέκτες που την αιτούνται.

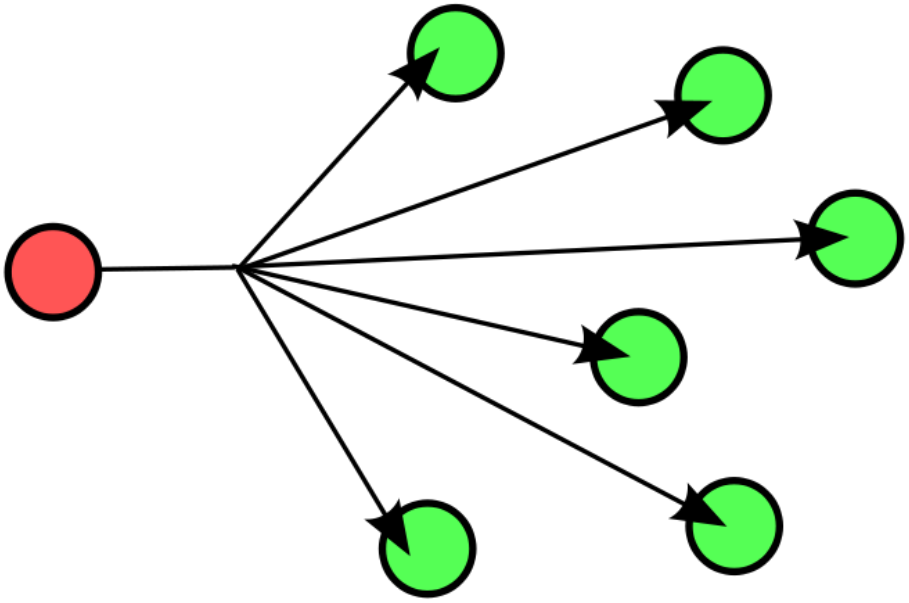


Unicast Σύνδεση ενός δικτύου

Στις περισσότερες των περιπτώσεων όπου εφαρμογές σε δίκτυα χρειάζονται διανομή δεδομένων τότε με την τεχνική αυτή απαιτείται μεγάλο κόστος διότι κάθε σύνδεση απαιτεί τη χρήση πόρων και ξεχωριστό εύρος ζώνης για κάθε μεταφορά δεδομένων.

1.4.2 Broadcast τρόπος μετάδοσης δεδομένων

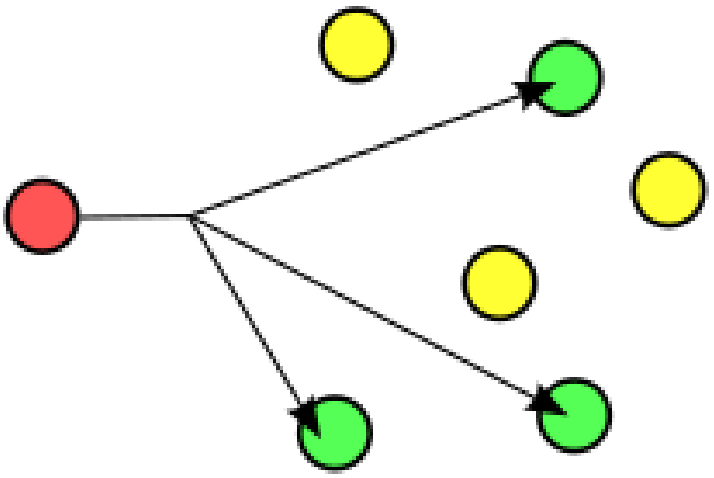
Με τη μέθοδο αυτή ο πομπός εκπέμπει ένα μήνυμα που πάει ταυτόχρονα σε όλους τους δέκτες ανεξάρτητα αν το ζήτησαν όχι γιατί ο πομπός δε γνωρίζει ποιος αιτήθηκε από το δίκτυο. Αυτό σημαίνει πως κάθε δέκτης ελέγχει αν το μήνυμα τον ενδιαφέρει και ανάλογα το κρατάει ή το απορρίπτει. Είναι σαφές ότι με αυτή την τεχνική κερδίζουμε σε χρόνο αφού το μήνυμα εκπέμπεται ταυτόχρονα προς όλους όμως χάνουμε χρόνο και υπολογιστική ισχύ γιατί κάθε δέκτης πρέπει να κάνει έλεγχο για κάθε μήνυμα που λαμβάνει κάτι όχι ιδιαίτερα συμφέρον όταν πρόκειται για εκπομπή μεγάλου και συνεχούς όγκου μηνυμάτων.



Broadcast Μετάδοση

1.4.3 Multicast τρόπος μετάδοσης δεδομένων

Με αυτό τη μέθοδο ο πομπός στέλνει στοχευμένα την πληροφορία ταυτόχρονα όχι σε όλους τους δέκτες του δικτύου αλλά σε συγκεκριμένα group κάθε φορά. Έτσι γίνεται συνδυασμός των δύο παραπάνω μεθόδων που είναι αλήθεια ότι παρουσιάζουν πολλά προβλήματα.



Multicast Μετάδοση

1.4.4 Εφαρμογές στην καθημερινότητα που εφαρμόζει την κάθε μέθοδο

Ανάλογα με την τοπολογία αλλά και το είδος της σύνδεσης το κάθε δίκτυο χρησιμοποιεί ανάλογα και αντίστοιχη μέθοδο μετάδοσης. Οι οπτικές ίνες , τα καλώδια χαλκού, οι Ethernet διασυνδέσεις, χρησιμοποιούν τη μέθοδο unicast. Κόμβοι που χρησιμοποιούν για την αναμετάδοση χρησιμοποιούν μέθοδο multicast. Ένα ασύρματο δίκτυο είναι multicast διασύνδεσης κατά κανόνα γιατί όλοι οι κόμβοι συμμετέχουν στη λήψη πληροφοριών.
Όταν υπάρχει κόμβος με πολλές διαθέσιμες εξερχόμενες συνδέσεις η απόφαση για το ποια θα χρησιμοποιηθεί στην ερώτηση για το ποια τεχνική θα χρησιμοποιηθεί απαντά ένα πολύπλοκος αλγόριθμος που κρίνει κάθε στιγμή τι είναι καλύτερο. Στις μέρες μας βέβαια οι δρομολογητές είναι αρκετά έξυπνοι ώστε να αξιοποιούν με τον καλύτερο δυνατό τρόπο όλες τις τεχνικές και να επιβαρύνεται το δίκτυο όσο το δυνατό λιγότερο.
Υπάρχουν δύο τεχνικές δρομολόγησης αυτή που το πακέτο της πληροφορίας περιέχει και τη διεύθυνση του κόμβου που θα λάβει την πληροφορία και στη λογική να κάνει αναζήτηση σε όλο το δίκτυο με ερωτήματα σε κάθε μέλος του για το αν ανταποκρίνεται σε αυτόν η πληροφορία.
Η τελευταία μέθοδος είναι αρκετά δαπανηρή και χρονοβόρα όπως μπορεί να αντιληφθεί οποιοσδήποτε για αυτό και όταν έχουμε μεγάλα δίκτυα,

χωρίζονται σε μικρότερα ομαδοποιημένα υποδίκτυα ώστε η δρομολόγηση των πακέτων να γίνεται ευκολότερα και γρηγορότερα.

Δύο δημοφιλή πρωτόκολλα δρομολόγησης είναι το RIP και το OSPF
Μία από τις βασικές κατηγορίες αλγόριθμων δρομολόγησης είναι αυτή που βασίζεται στον αλγόριθμο distanse-vector (διάνυσμα-απόσταση). Κάθε κόμβος κατασκευάζει ένα διάνυσμα που περιέχει τις αποστάσεις (κόστος) για όλους τους υπόλοιπους κόμβους και κατανέμει το διάνυσμα αυτό στους άμεσα γειτονικούς κόμβους. Το πρωτόκολλο RIP έχει κατασκευαστεί με βάση τον αλγόριθμο distanse-vector. Οι δρομολογητές που λειτουργούν με RIP στέλνουν ανακοίνωση των διανυσμάτων τους συχνά (π.χ κάθε 30 δευτερόλεπτα). Επίσης ένας δρομολογητής στέλνει ενημέρωση όταν δέχεται και αυτός κάποια αλλαγή στον πίνακα δρομολόγησης του.

Το OSPF (Open Shortest Path First) είναι ένα open standard πρωτόκολλο δρομολόγησης το οποίο χρησιμοποιείται από πολλούς κατασκευαστές δικτυακών προιόντων. Η δρομολόγηση με βάση αυτόν τον αλγόριθμο ανήκει στην δεύτερη βασική κατηγορία των intra-domain πρωτοκόλλων δρομολόγησης.

Η βασική ιδέα του αλγόριθμου δρομολόγησης κατάστασης συνδέσμου είναι απλή. Κάθε κόμβος γνωρίζει τους άμεσα συνδεδεμένους γειτονικούς του κόμβους και εάν το σύνολο της γνώσης αυτής μεταδοθεί σε κάθε κόμβο, τότε κάθε κόμβος θα γνωρίζει αρκετές πληροφορίες ώστε να μπορεί να κατασκευάσει τον χάρτη του δικτύου.

Από την στιγμή που κάθε κόμβος διαθέτει τον χάρτη του δικτύου μπορεί να επιλέξει την βέλτιστη διαδρομή για κάθε προορισμό. Ο υπολογισμός των βέλτιστων διαδρομών

βασίζεται σε έναν αλγόριθμο από την θεωρία των γράφων – στον αλγόριθμο του μικρότερου μονοπατιού του Dijkstra.

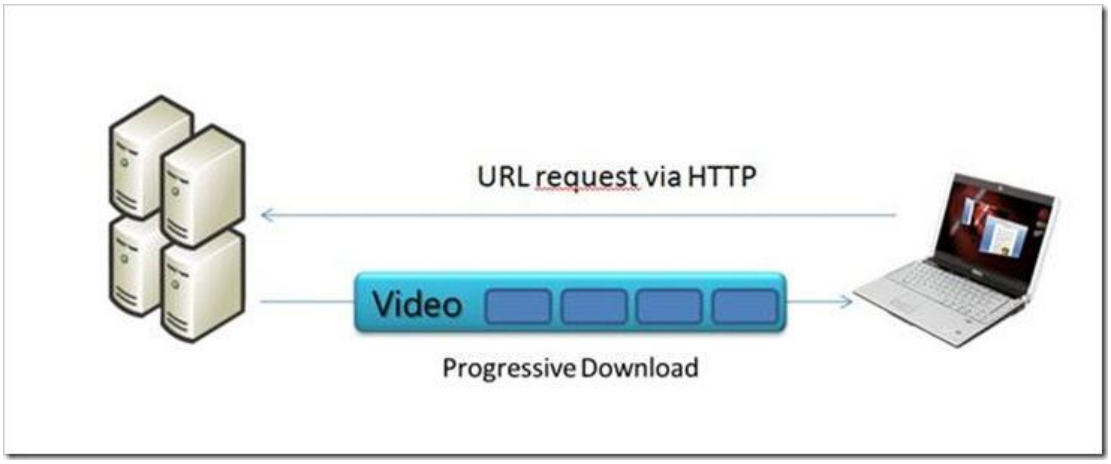
Το OSPF εισάγει ένα πρόσθετο επίπεδο στην δρομολόγηση με το να επιτρέπει ένα domain να τμηματοποιείται σε περιοχές. Αυτό σημαίνει ότι ένας δρομολογητής δεν απαιτείται να γνωρίζει πως να προσπελάσει το κάθε δίκτυο στο domain αυτό, του είναι αρκετό να γνωρίζει το πως θα στείλει πληροφορίες προς την σωστή περιοχή. Με αυτόν τον τρόπο υπάρχει σημαντική μείωση στον όγκο των πληροφοριών που πρέπει να μεταδίδεται και να αποθηκεύεται σε κάθε δρομολογητή. Επιπλέον το OSPF επιτρέπει την ύπαρξη πολλαπλών διαδρομών για έναν προορισμό με το ίδιο κόστος, ώστε να κατανέμεται η κίνηση ομοιόμορφα σε αυτές τις διαδρομές.

Κεφάλαιο 2

Μέθοδοι on-line αναπαραγωγής βίντεο

2.1 Progressive download

Με τη μέθοδο αυτή ένα video αναπαράγεται στον υπολογιστή όταν έχει «κατέβει» ένα μέρος του τη στιγμή που συνεχίζεται η λήψη του. Το μειονέκτημά του είναι η μεγάλη σπατάλη πόρων δικτύου, υπολογιστικού συστήματος και υπολογιστικής ισχύος . Επιπρόσθετα δεν προσαρμόζεται δυναμικά στις αλλαγές του δικτύου αλλά και στις συνθήκες κάθε χρήστη και τέλος δεν υποστηρίζει σε καμία περίπτωση το live streaming.

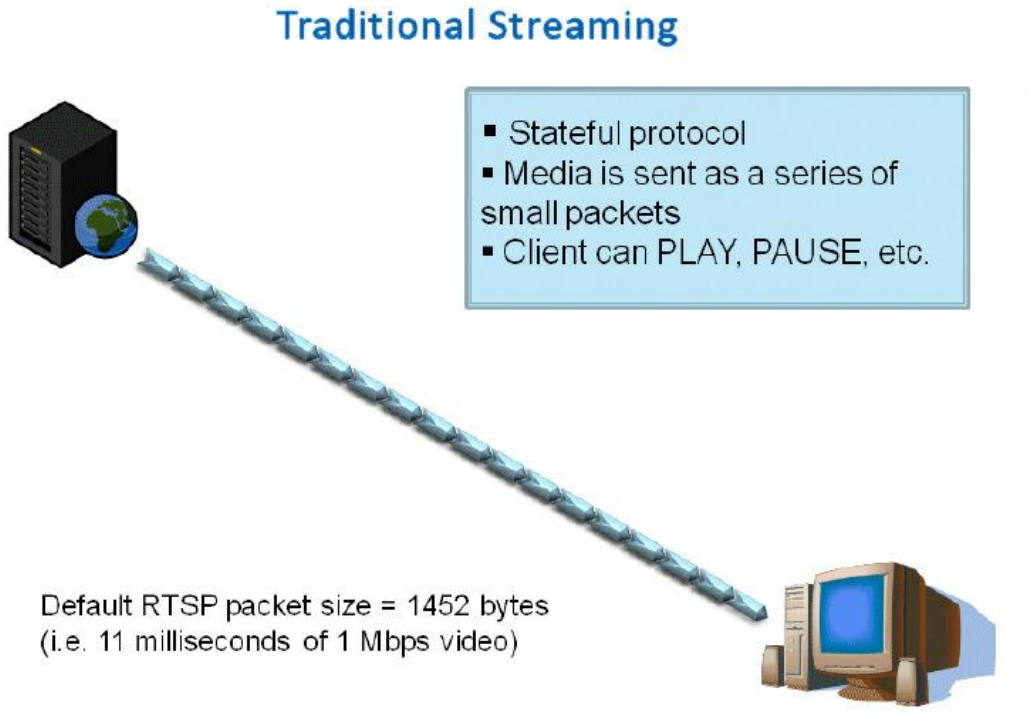


2.2 Steteful Streaming

Η συγκεκριμένη τεχνική στηρίζεται σε live steaming σύνδεση μεταξύ ενός υπολογιστή και ενός server που συνδέεται απευθείας πάνω του και μέσα από έξυπνα πρωτόκολλα γίνεται η αποστολή του βίντεο. Ο server ελέγχει συνέχεια την κατάσταση του υπολογιστή μέχρι αυτός να αποσυνδεθεί. Η αποστολή του βίντεο γίνεται με σταθερό ρυθμό σε μικρά πακετάκια που ονομάζονται RTP με τη βοήθεια του πρωτοκόλλου UDP και TCP. Ο χρήστης μέσα από μια συγκεκριμένη πλατφόρμα χρησιμοποιεί εντολές play, pause, volume, screen size κλπ για να επιλέξει το πώς θα δει το βίντεο.

Πρόκειται για μια φαινομενικά πολύ καλή λύση όπου όμως παρουσιάζει πολλά προβλήματα. Για να έχει επιτυχία η σύνδεση με το server θα πρέπει ο τελευταίος να

είναι αρκετά έξυπνος με ειδικά χαρακτηριστικά ώστε να υπάρχει σταθερότητα καθώς και ειδικά πρωτόκολλα. Ένα ακόμα μειονέκτημα της συγκεκριμένης τεχνικής είναι ότι είναι αρκετά ευαίσθητη που σημαίνει ότι αν υπάρξει ένα μικρό πρόβλημα στη σύνδεση η αναπαραγωγή, προβολή του βίντεο παγώνει.



2.3 Adaptive Bitrate Streaming

2.3.1 Εισαγωγή

Πρόκειται για μία τεχνική που αναπτύχθηκε για να εξυπηρετήσει το streaming για video και πολυμέσα. Βασίζεται στο πιο διαδεδομένο πρωτόκολλο που χρησιμοποιείται στο διαδίκτυο το HTTP. Το συγκεκριμένο πρωτόκολλο δε χρειάζεται μόνιμη σύνδεση αλλά στιγμιαίες αιτήσεις-συνεδρίες και λειτουργεί σε κατανεμημένα δίκτυα.

Έχει την ικανότητα να "παίζει με την ποιότητα" του βίντεο ανάλογα με τις δυνατότητες του επεξεργαστή αλλά και τη διαθεσιμότητά του κάθε στιγμή. Απαιτεί τη χρήση ενός κωδικοποιητή που μπορεί να κωδικοποιήσει το βίντεο σε πολλαπλούς ρυθμούς μετάδοσης. Η ποιότητα της αναπαραγωγής για κάθε χρήστη ποικίλει ανάλογα με τους διαθέσιμους πόρους. Πολύ λίγο buffering, γρήγορος χρόνος εκκίνησης και ένα καλό αποτέλεσμα για high-end και low-end συνδέσεις είναι τα αποτελέσματα του adaptive bitrate streming.

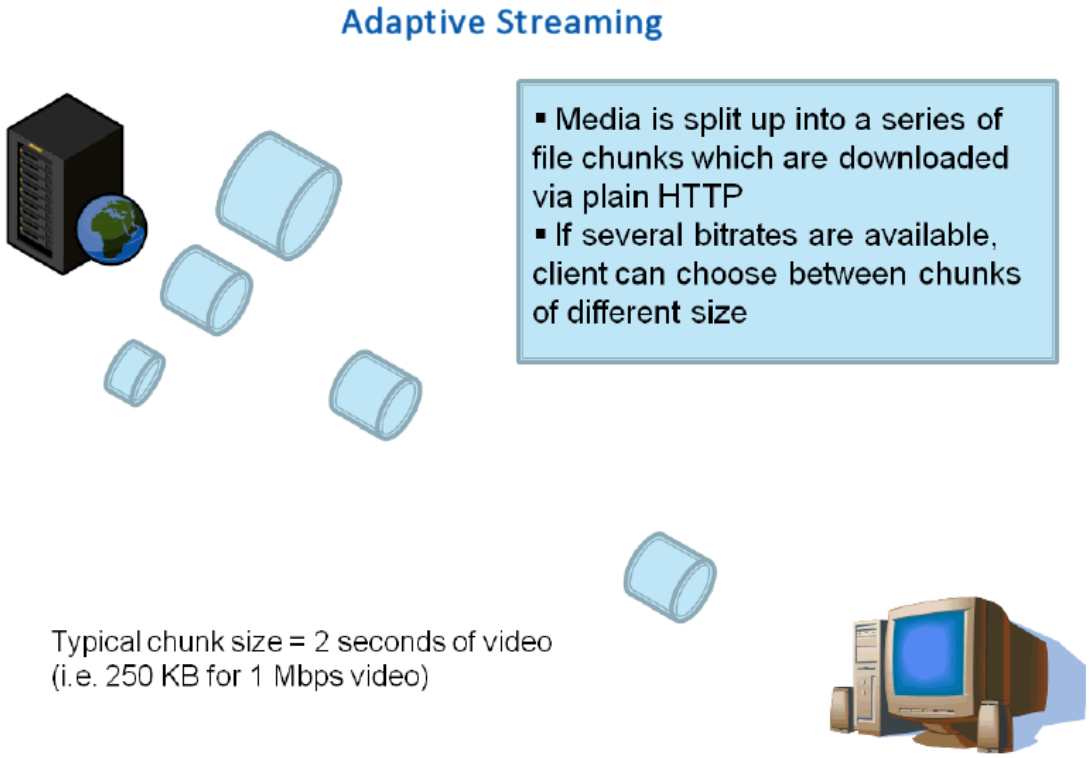
2.3.2 Τρέχουσα χρήση του adaptive bitrate streaming.

Μεγάλες εταιρείες παραγωγής ήχου, βίντεο, πολυμέσων αλλά και διαδικτυακά κανάλια ή τηλεόρασης που εκπέμπουν και στο διαδίκτυο, προσφέρουν στους τελικούς χρήστες μιας υψηλής ποιότητας βίντεο που ταυτόχρονα έχει μικρό διαχειριστικό κόστος σε προσωπικό καθώς χρησιμοποιεί τεχνολογία αιχμής. Μια

λύση απλή και ταυτόχρονα προσβάσιμη από όλους αντιμετωπίζει το χρόνιο πρόβλημα του streaming όπως αναφέρθηκε παραπάνω με επιτυχία.

2.3.3 Πλεονεκτήματα του adaptive bitrate streaming.

Όταν μιλάμε για την τεχνολογία adaptive streaming μιλάμε για βίντεο πολύ υψηλής ποιότητας σύμφωνα πάντα με τις δυνατότητες που υπάρχουν κάθε στιγμή στο δίκτυο αλλά και τους πόρους του υπολογιστή, επεξεργαστής, μνήμη, πόροι των οποίων οι διαθεσιμότητα αλλάζει από στιγμή σε στιγμή. Με δεδομένο ότι ο χώρος του βίντεο αυξάνεται εκθετικά τα δίκτυα διανομής και οι πάροχοι έχουν τη δυνατότητα να παράσχουν μια ανώτερης ποιότητας υπηρεσία. Το adaptive streaming απαιτεί λιγότερη κωδικοποίηση απλοποιώντας πολλές διαδικασίες που με τις άλλες τεχνολογίες φάνταζαν πολύπλοκες και δαπανηρές.



(Εικόνα: adaptive bitrate streaming)

2.3.4 Προκλήσεις και Quality of Experience (QoE)

Όταν γίνεται ένας έλεγχος κατά τη σύνδεση σε ένα δίκτυο οικιακό, ή πολύ μεγαλύτερο μη ελεγχόμενο πάντως είναι άγνωστο, τι ασφάλεια έχει ο router, αν διαθέτει firewall . Σε ένα ασύρματο δίκτυο συνήθως τα μέτρα ασφαλείας που ενεργοποιεί ο διαχειριστής του είναι ισχυρότερα λόγω του ότι η πληροφορία ταξιδεύει στο αέρα και είναι πολύ εύκολο να υποκλαπεί ή να διαστρεβλωθεί. Αυτό το εμπόδιο μπορεί να ξεπεραστεί με τα πρωτόκολλα HTTP και HTTPS που χρησιμοποιούνται για την επικοινωνία. Το HTTP χρησιμοποιεί τη θύρα 80 για τις αιτήσεις. Οι αιτήσεις προς αυτή τη θύρα είναι γνωστές ώστε οποιοδήποτε τείχος προστασίας ή router να επιτρέπουν την πρόσβαση.

2.3.5 Έλεγχος εύρους ζώνης

Ένα κρίσιμο ερώτημα που έχει βασανίσει έτη πολλά τους σχεδιαστές δικτύων, των πρωτοκόλλων δρομολόγησης αλλά και των δικτυακών λειτουργικών συστημάτων είναι αυτό του ελέγχου εύρους ζώνης. Έστω ότι σε ένα δίκτυο σε ένα φοιτητικό campus με Ν χρήστες κάποιος παρακολουθεί με live streaming την αγαπημένη του ποδοσφαιρική ομάδα να αγωνίζεται. Ένας άλλος χρήστης εκείνη τη στιγμή αποφασίζει ότι θέλει να μεταφέρει ένα μεγάλο όγκο αρχείων που έχουν να κάνουν με την πτυχιακή του εργασία από έναν υπολογιστή σε έναν άλλον. Ένας τρίτος αποφασίζει να πάρει backup τη μία βάση δεδομένων που τηρεί στοιχεία για τη σίτιση, στέγαση των φοιτητών. Το αποτέλεσμα για αυτό που παρακολουθεί ποδόσφαιρο; Καταστροφικό! Αν το βίντεο δεν έχει την ευελιξία να κωδικοποιηθεί σε διαφορετικές ποιότητες και το πρωτόκολλο μετάδοσης να την αλλάζει δυναμικά, χωρίς διακοπή της αναπαραγωγής ή ενέργεια από το χρήστη τότε η παρακολούθηση του βίντεο θα είναι από δύσκολη ως αδύνατη.

2.3.6 Πολλαπλοί χρήστες και πολλαπλές αναλύσεις

Στην εποχή που οι συσκευές ασύρματες και κινητές έχουν πάρει ένα σημαντικό κομμάτι των συσκευών που έχουν πρόσβαση στο διαδίκτυο όπου συνεπάγεται μεγάλης αύξησης των απαιτήσεων σε ανάλυση αλλά και ρυθμό μετάδοσης. Κάθε συσκευή έχει τα δικά της χαρακτηριστικά αλλά δεν είναι θεμιτό να υπάρχουν ξεχωριστοί κωδικοποιητές, ξεχωριστά συστήματα και ειδικοί servers για κάθε συσκευή που πρέπει να υποστηριχθεί.

2.3.7 Ασφάλεια Περιεχομένου

Η πιο κλασική μέθοδος ασφάλειας μιας πληροφορίας που ταξιδεύει στο διαδίκτυο είναι αυτή της κρυπτογράφησης/αποκρυπτογράφησης από πομπό σε δέκτη με τη χρήση κατάλληλων κλειδιών. Η μέθοδος αυτή έχει χρησιμοποιηθεί με επιτυχία στη μετάδοση ζωντανού βίντεο μέσω διαδικτύου. Το περιεχόμενο είναι κρυπτογραφημένο μεταξύ κωδικοποιητή και server και πάλι μεταξύ server και χρήστη. Ωστόσο ο κίνδυνος με αυτή την προσέγγιση είναι ότι το περιεχόμενο είναι προσωρινά γραμμένο στη 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.

Κεφάλαιο 3

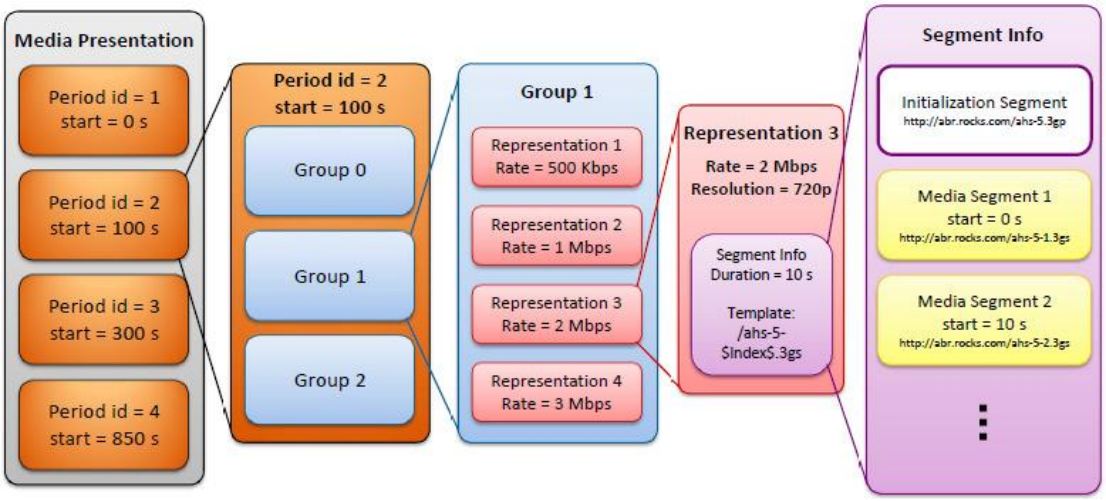
Dynamic Adaptive Streaming over HTTP (DASH).

3.1 Εισαγωγή

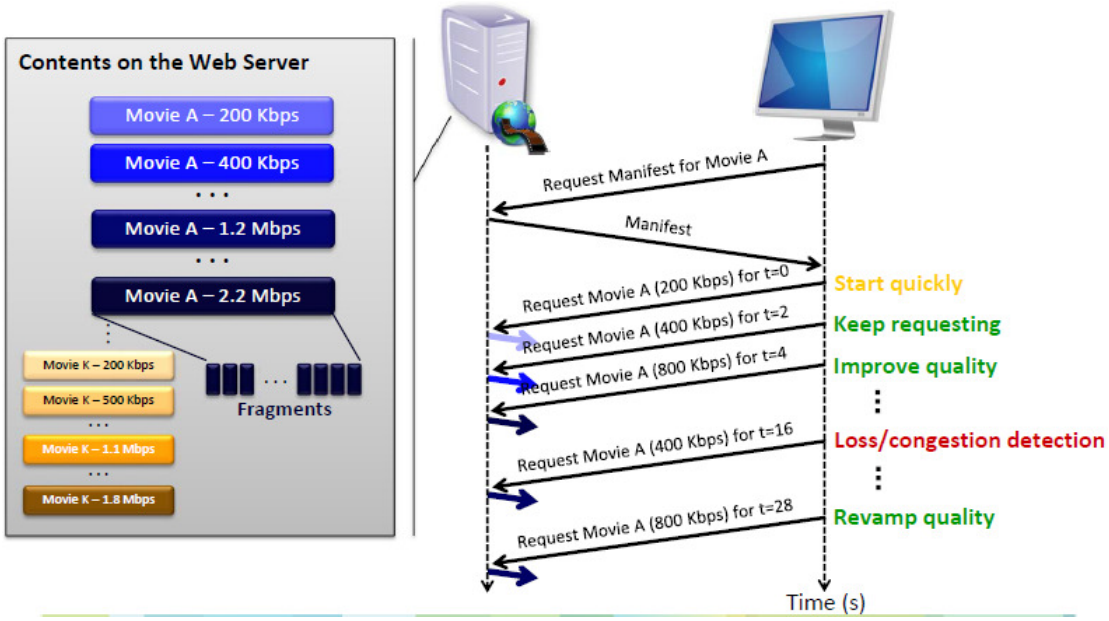
Πρόκειται για μια multimedia streaming τεχνολογία που βασίζεται στο MPEG. Οι εργασίες πάνω στο DASH ξεκίνησαν το 2010 και τον Ιανουάριο του 2011 ορίστηκε ως σχέδιο για ένα διεθνές πρότυπο που υλοποιήθηκε το μέχρι το τέλος του 2011.

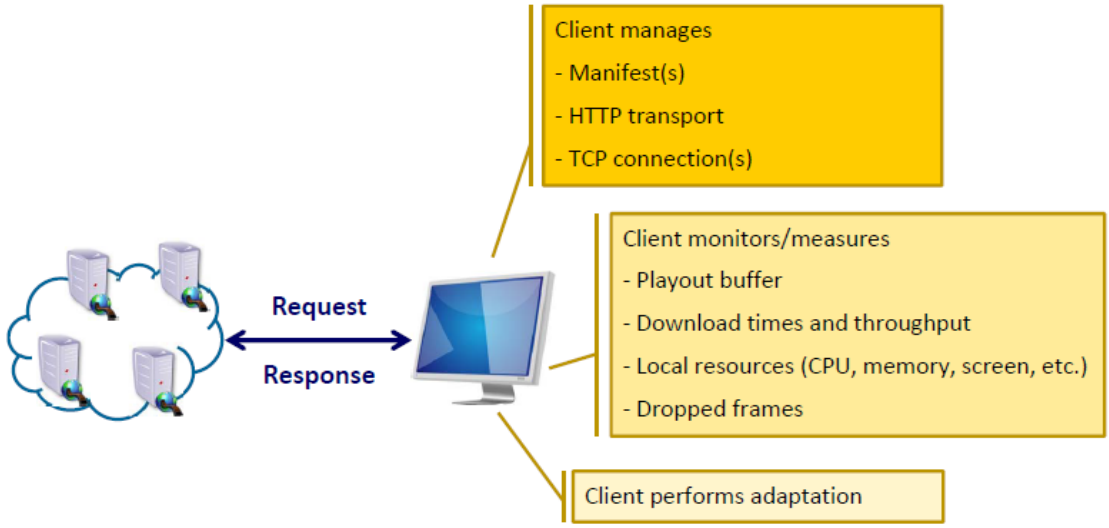
3.2 Λειτουργία

Το DASH ως μια multimedia streaming τεχνολογία αποθηκεύει ένα βίντεο σε ένα server χωρισμένο σε ένα ή περισσότερα τμήματα διαφορετικής ποιότητας και ανάλυσης και ανάλογα στέλνεται σε ένα client με το πρωτόκολλο HTTP. Το media presentation description (MPD) αποθηκεύεται σε ένα αρχείο xml μορφής και περιγράφει τη δομή και το περιεχόμενο παρουσίασης του βίντεο. Στο παρακάτω σχήμα περιέχονται διάφορες πληροφορίες όπως χρονοδιάγραμμα χαρακτηριστικά, αναλύσεις, ρυθμοί μετάδοσης, URL διευθύνσεις κτλ. Κατά τη διάρκεια αναπαραγωγής του βίντεο ανάλογα από μια σειράς παραμέτρους όπως οι συνθήκες του δικτύου, τις δυνατότητες του συστήματος και τις προτιμήσεις του χρήστη επιτρέπεται το bitrate streaming.



Πιο ειδικά η λειτουργία του DASH αρχίζει με τον Client να αιτείτε από το Server το MPD και στη συνέχεια ο client κάνει τους απαραίτητους υπολογισμούς όπως throughput, εύρος ζώνης, ικανότητες υλικού κ.τ.λ. και ύστερα παίρνει την καλύτερη απόφαση για τη βέλτιστη ικανή αναπαραγωγή του βίντεο όπως φαίνετε στις παρακάτω εικόνες





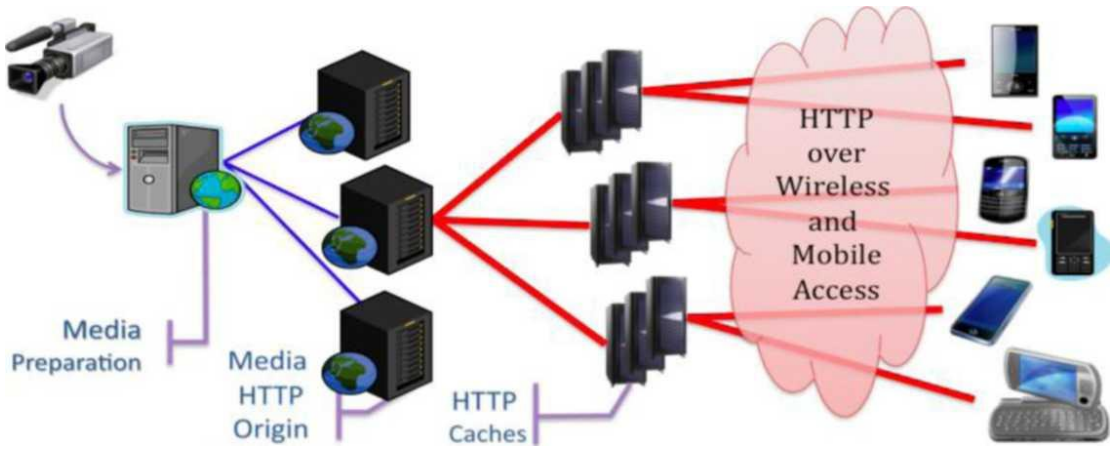
3.3 Πλεονεκτήματα και σύγκριση του DASH

Όπως αναφέρθηκε ήδη το παραδοσιακό streaming χρησιμοποιεί κάποιο είδος stateful πρωτοκόλλου όπως το Real-Time Streaming Protocol (RTSP) όπου όταν ο client συνδεθεί στο server αυτός έχει συνεχή εικόνα της κατάστασής του μέχρι να αποσυνδεθεί. Όταν ξεκινήσει μια τέτοια ζεύξη ο server στέλνει το περιεχόμενό του μέσω ενός συνεχούς streaming σε πακέτα πάνω από τα πρωτόκολλα TCP, UDP. Από την άλλη μεριά όμως το πρωτόκολλο HTTP είναι stateless δηλαδή δε δημιουργεί μόνιμες συνδέσεις αλλά περιστασιακές τη στιγμή που πρόκειται να εξυπηρετήσει κάποιο αίτημα.

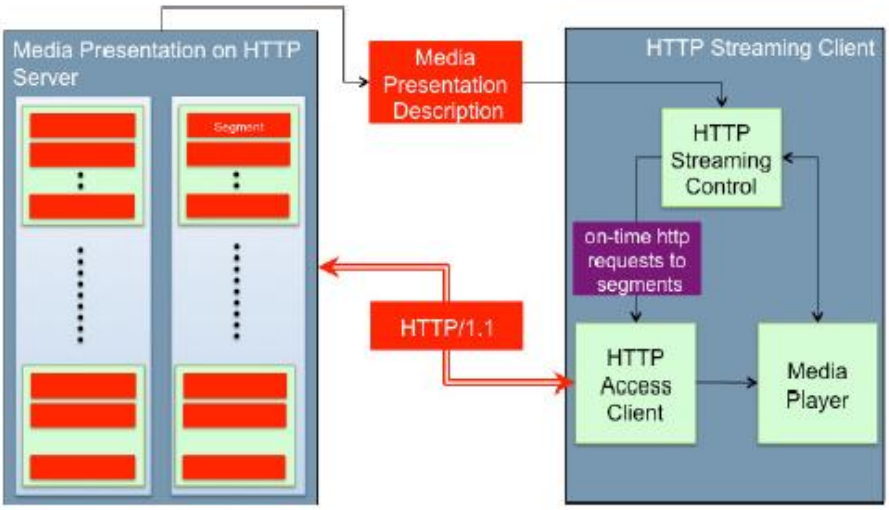
Εναλλακτικά μπορεί με το steaming να χρησιμοποιηθεί μια τεχνική που ονομάζεται προοδευτική λήψη (progressive download) από τους συνήθεις HTTP Web servers. Εφόσον ο client υποστηρίζει πρωτόκολλο HTTP/1.1 είναι δυνατόν να αποστείλει αιτήματα HTTP στον εξυπηρέτη Web server (πρέπει και ο server να υποστηρίζει το

HTTP/1.1) τα οποία ζητούν συγκεκριμένες ομάδες Bytes ενός πολυμεσικού αρχείου. Έτσι είναι δυνατό να λάβει ο χρήστης οποιοδήποτε τμήμα του βίντεο που 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 ξεκινήσει η προοδευτική λήψη, 2) δεν είναι δυνατόν να προσαρμόσει το bitrate adaptive και 3)δεν υποστηρίζει πολυμεσικές υπηρεσίες πραγματικού χρόνου (live streaming)

Αντίθετα τα παραπάνω το DASH έχει τη δυνατότητα να αντιμετωπίσει και να υπερισχύσει έναντι των παραπάνω. Επειδή στηρίζεται στο HTTP πρωτόκολλο δεν καταναλώνει πόρους του δικτύου, έχει αρκετά στοιχεία με την προοδευτική λήψη πάνω από HTTP με συνέπεια να μην είναι αναγκαία η αλλαγή εξοπλισμού από χρήστες/παρόχους για τη λειτουργία του.



Στην παραπάνω εικόνα φαίνεται μια πιθανή αρχιτεκτονική μετάδοσης πολυμέσων για streaming πάνω από HTTP. Κατά το αρχικό στάδιο δημιουργούνται διαφορετικές εκδόσεις ενός ή όλου των μερών του πολυμεσικού περιεχομένου σε τεμάχια με την κάθε έκδοση να έχει διαφορετική κωδικοποίηση. Στη συνέχεια τα τεμάχια μαζί με τα αρχεία MPD που περιγράφουν στέλνονται σε έναν η περισσότερους servers που είναι διαθέσιμοι. Συνήθως οι εν λόγο servers είναι HTTP ώστε η επικοινωνία και η όλη εν γένει διαδικασία να γίνεται πάνω από το HTTP . Επιπρόσθετα μπορεί να χρησιμοποιηθούν http caches και proxies ώστε να μειωθεί ο φόρτος των server πηγής. Ο κάθε client γνωρίζοντας στα MPD αρχεία που περιγράφουν τη 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 τον πλήρη έλεγχο της σύνδεσης, της συνόδου, κάνει τα αιτήματα στην ώρα τους και φροντίζει για την ομαλή αναπαραγωγή των τεμαχίων που λαμβάνει μέσω της προσαρμογής bitrate και των 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.



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 που είναι σχεδόν πάντα ανοιχτή οπότε δεν τίθεται θέμα firewall ή NAT.
- Το HTTP streaming διαδίδεται ταχύτητα ως τρόπος μετάδοσης βίντεο πάνω από το διαδίκτυο.
- Το HTTP θεωρείται ιδιαίτερα αξιόπιστο και απλό, επειδή τόσο αυτό όσο και τα πρωτόκολλα μεταφοράς που χρησιμοποιεί (TCP/IP) είναι ευρέως διαδεδομένα
- Ο έλεγχος της συνόδου δίνεται εξ ολοκλήρου στο χρήστη. Το τερματικό εγκαθιδρύει μια ή περισσότερες συνδέσεις TCP σε έναν ή περισσότερους server ή cache HTTP.
- Δίνει την ευκαιρία στο τερματικό να επιλέξει αυτό το bitrate που επιθυμεί χωρίς να χρειάζεται να είναι κάτι που θα ρυθμίζει το stream server.
- Δίνει τη δυνατότητα αλλαγής του bitrate όταν αυτό κρίνεται απαραίτητο χωρίς να διαταραχθεί η απρόσκοπτη λειτουργία του βίντεο. Όλα αυτά γίνονται χωρίς την παρέμβαση ενός stream server.
- Έχει τη δυνατότητα να επιταχύνει τη σύγκλιση στις προσφερόμενες υπηρεσίες ανάμεσα σε κινητές και σταθερές συσκευές, καθώς μπορεί να χρησιμοποιηθεί ως μια κοινή πλατφόρμα διανομής.

Υπηρεσίες που υποστηρίζει το DASH είναι

- Streaming κατά απαίτηση (On demand streaming)
- Εκπομπή παραδοσιακής τηλεόρασης . Βίντεο δηλαδή στο οποίο ο χρήστης δεν μπορεί να παρέμβει απλά παρακολουθεί ζωντανά όπως στην κλασική τηλεόραση

- Θέαση με δυνατότητα χρονικής ολίσθησης (time shift) με δυνατότητα επιλογής του βίντεο όποτε ο χρήστης επιθυμεί, εγγραφής βίντεο (Personal Video Recording- PVR).
- Επιπλέον επιλογές αναπαραγωγής όπως fast forward, rewind
- Απλή ενσωμάτωση διαφημίσεων στο on demand και live streaming
- Αποδοτική μετάδοση πολλαπλών γλωσσών και ηχητικών κομματιών.

Κεφάλαιο 4

Γνωστές υλοποιήσεις του DASH

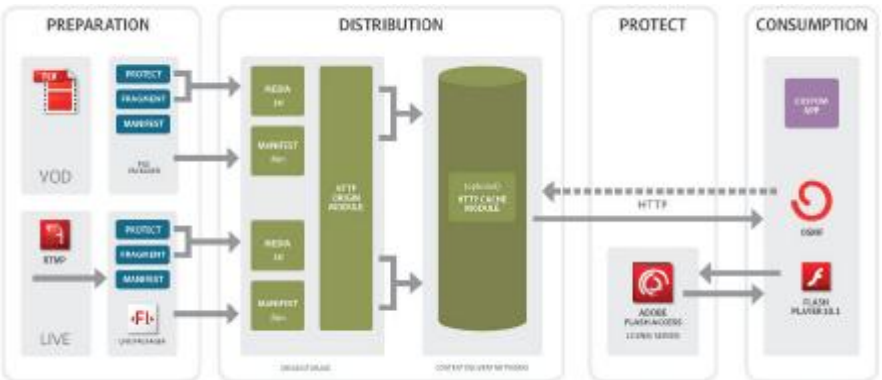
4.1 Adobe Systems's HTTP Dynamic Streaming

Η Adobe μετά την έκδοση 10.1 του Flash Player διαθέτει πλέον τη δυνατότητα του DASH χρησιμοποιώντας κοινούς HTTP servers και συσκευές δικτύου μέσα από την εφαρμογή MP4 διαμόρφωση κατακερματισμού(F4F). Υποστηρίζει ζωντανή ή κατά απαίτηση μετάδοση του βίντεο με προσαρμοσμένη ταχύτητας 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 συσκευών έχουν Flash Player
- Χρήση της γνωστής MP4 διαμόρφωσης κατακερματισμού(F4F)
- Προστασία πολυμεσικού περιεχομένου με τη χρήση Adobe 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, .f4v
- Υποστήριξη xml αρχείου δομής περιεχομένου τύπου .f4m

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

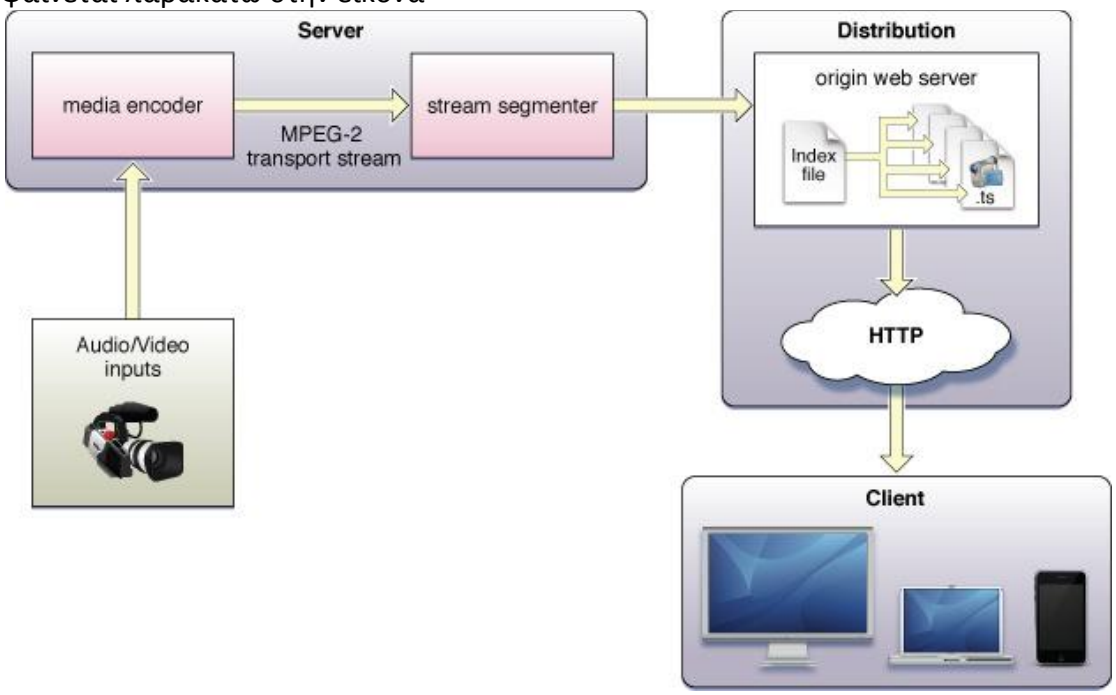


4.2 Apple HTTP Live Streaming

Το HTTP live streaming αποτελείται από τρία μέρη : το server, το σύστημα διανομής και τον client

- 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 αναγνώριση 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 και η κατάληξη σε .ts. Τα αρχεία δομής .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 παρακάτω στην εικόνα

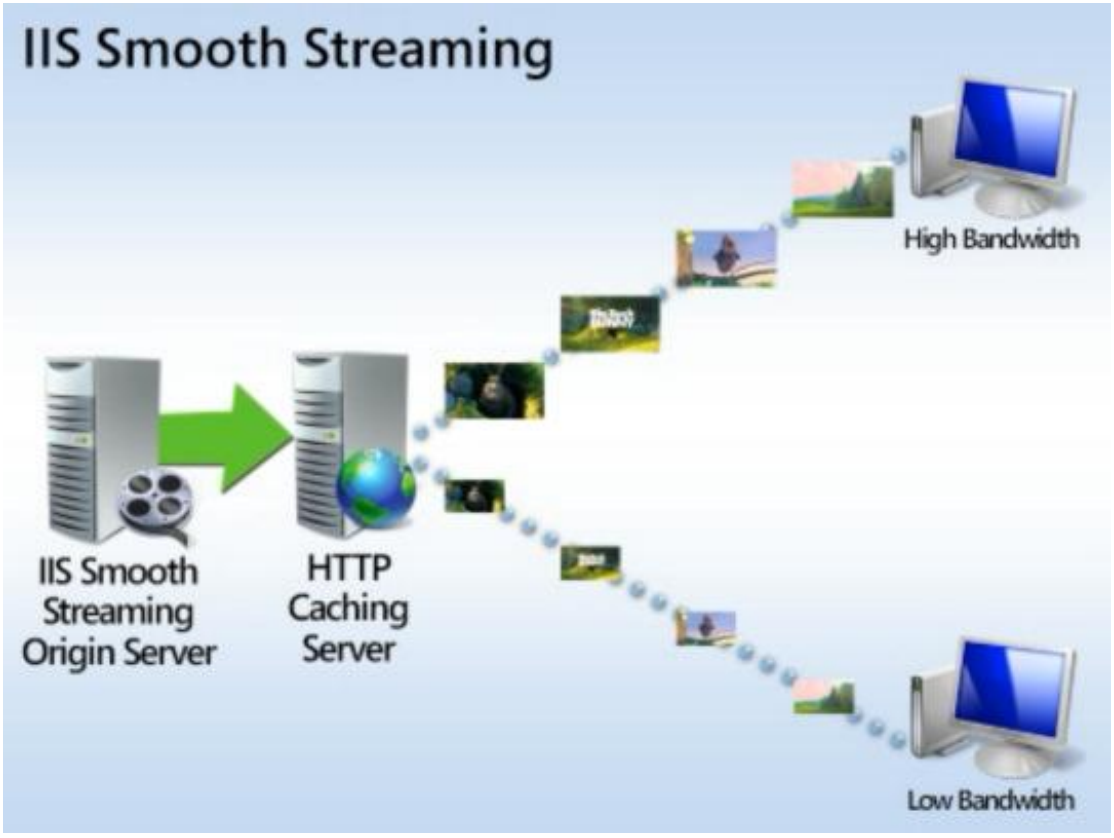


4.3 Microsoft's Smooth Streaming

Όπως όλες οι μεγάλες εταιρείες έτσι και η Microsoft ανέπτυξε τη δικιά της πλατφόρμα για το μηχανισμό DASH που της έχει δώσει το όνομα 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. Για τη διαμόρφωση των αρχείων χρησιμοποιείται το MPEG-4 part 14.

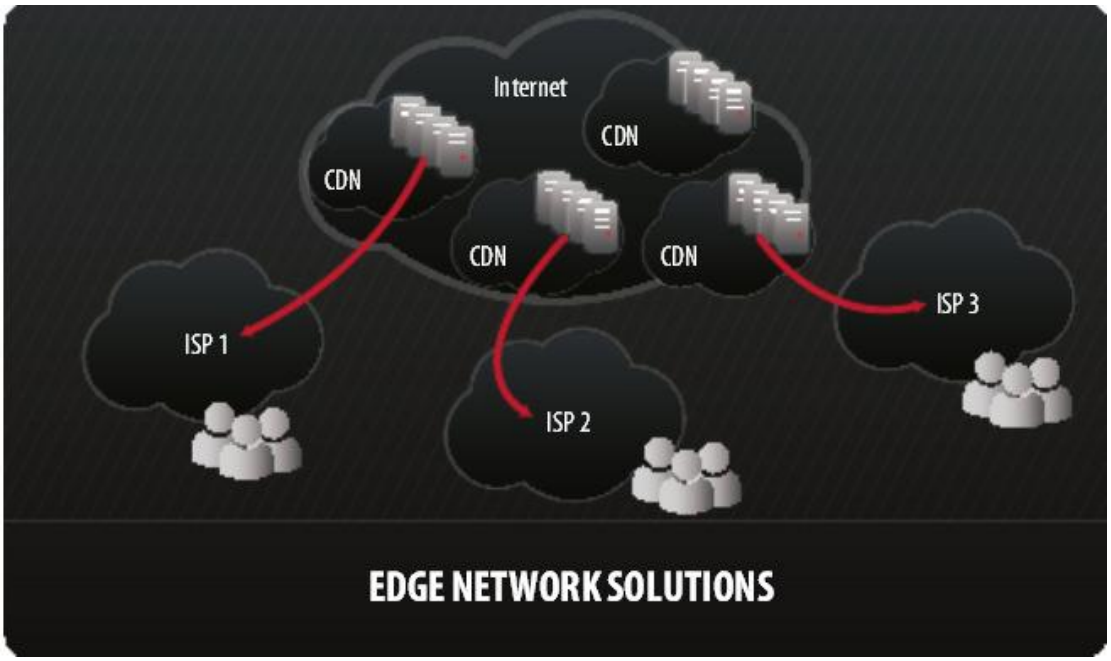
Το smooth streaming χρησιμοποιεί αρχεία πολυμεσικού περιεχομένου τύπου .ismv και δυο είδη αρχείων δομής περιεχομένου xml ένα για το server και έναν για τον client με πληροφορίες για αναλύσεις, ποιότητες, τους κωδικοποιητές κλπ.

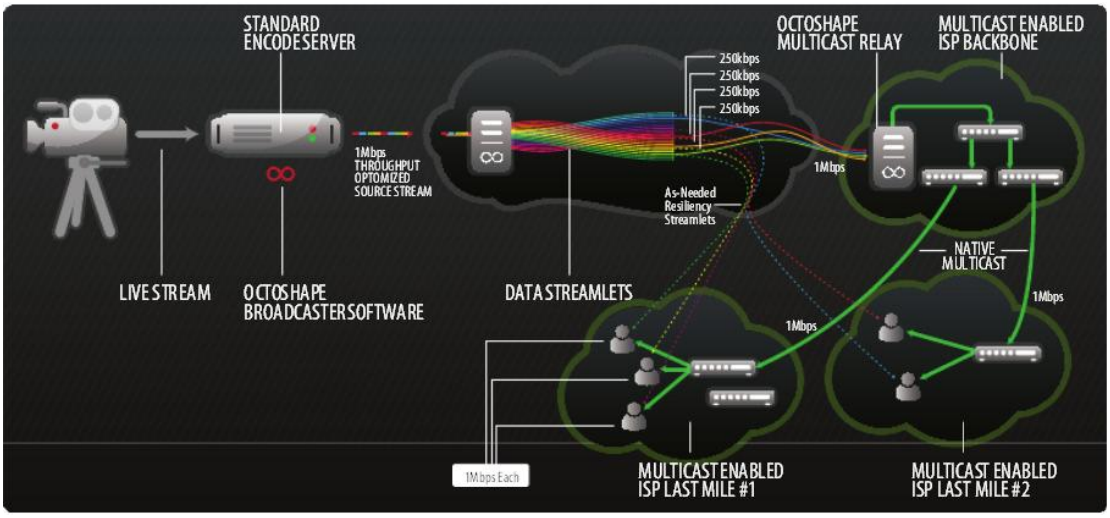
Στο παρακάτω σχήμα φαίνεται πώς γίνεται η δυναμική διαχείριση της σύνδεσης μεταξύ Client/Server



4.4 Octoshape Multi-BitRate

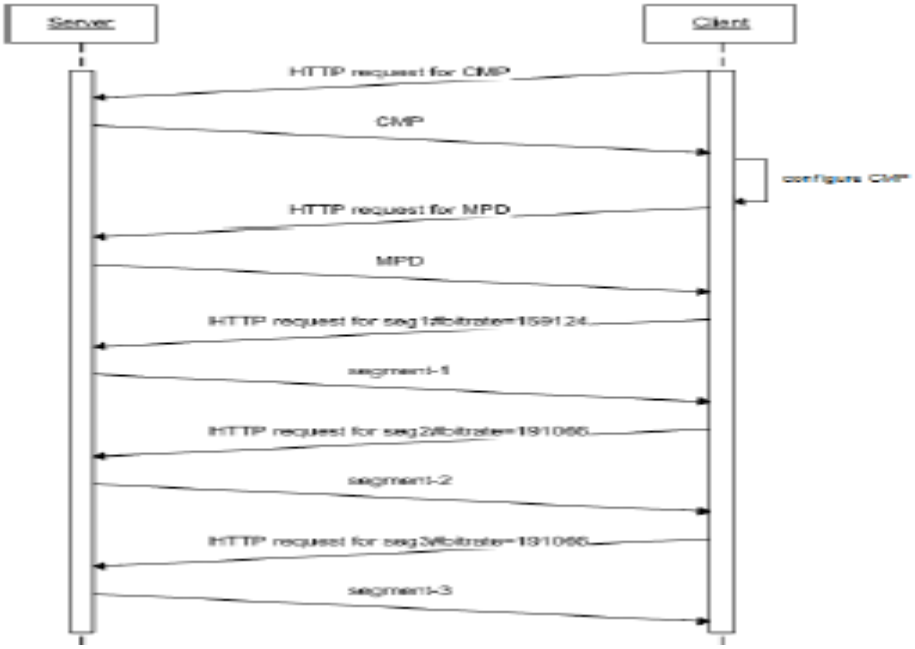
Ο Octoshape δεν παρέχει πολλές πληροφορίες για την τεχνολογία που χρησιμοποιεί ακριβώς το ίδιο σκεπτικό με τις παραπάνω εταιρείες και συνδυάζει το HTTP streaming με τα δίκτυα διανομής περιεχομένου για κατανεμημένη απόδοση στη μετάδοση πολυμεσικού υλικού δίνοντας έμφαση στην απόσταση των μερών που συνδέονται για τη μεταφορά δεδομένων και την αύξηση της ποιότητας.

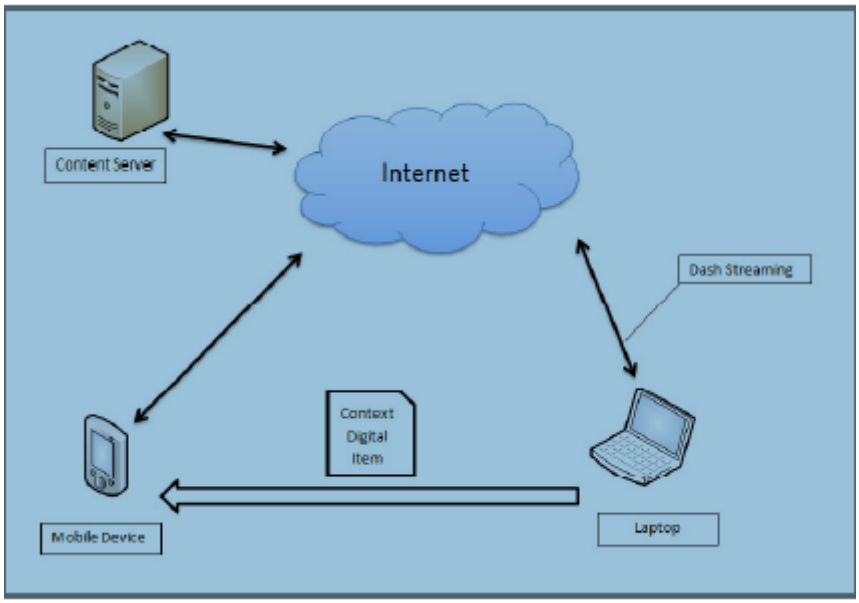




4.5 DASH VLC plugin

Το plugin σχεδιάστηκε για ερευνητικές και πειραματικές δραστηριότητες και δε χρησιμοποιείται επίσημα για τη αναπαραγωγή περιεχομένου μέσω hhtp streaming. Η λογική που ακολουθείται είναι παρόμοια με όλες τις υπόλοιπες και αναφέρεται σε έναν client που ζητάει το αρχείο ρυθμίσεων προβολής CMP και περιγραφής περιεχομένου MPD από το server και τα αρχεία είναι σε μορφή mp4





Αρχιτεκτονική του DASH vlc plugin

4.6 GPAC

Είναι ένα ανοιχτού κώδικα πολυεργαλείο πολυμέσων για ακαδημαϊκούς και ερευνητικούς σκοπούς. Καλύπτει διάφορους τομείς των πολυμέσων όπως τεχνολογίες παρουσίασης, γραφικά, κινούμενα σχέδια. Τρέχει σε όλα τα λειτουργικά συστήματα και είναι γραμμένη σε C

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 τη μορφή του αρχείου αντίστοιχα.

4.7 GStreamer

Ένα παρόμοιο πολυεργαλείο όπως το GPAC προσφέρει βιβλιοθήκη εργαλείων για ερευνητικούς, ακαδημαϊκούς σκοπούς.

Κεφάλαιο 5
Επισκόπηση του DASH με το Adobe HTTP Dynamic Streaming, πειραματική διαδικασία

5.1 Εισαγωγή

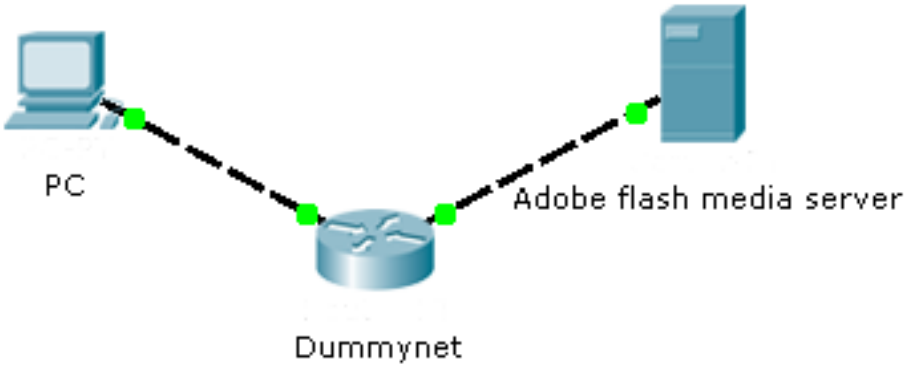
Για την πειραματική διαδικασία αλλά και τη μελέτη του DASH επί του πρακτέου αποφασίστηκε να χρησιμοποιηθεί η πλατφόρμα της adobe που είναι αρκετά δημοφιλής. Για τις ανάγκες της πτυχιακής χρειάστηκαν να γίνουν μερικές μετρήσεις επαναλαμβανόμενες και να εξαχθούν αντίστοιχα συμπεράσματα. Αυτό που επηρεάστηκε κατά κύριο λόγο ήταν η απόδοση του δικτύου αλλά και η αλλαγή πολυμεσικού περιεχομένου.

Οι λόγοι που επιλέχθηκε η πλατφόρμα της adobe σε σχέση με τις άλλες είναι οι εξής:

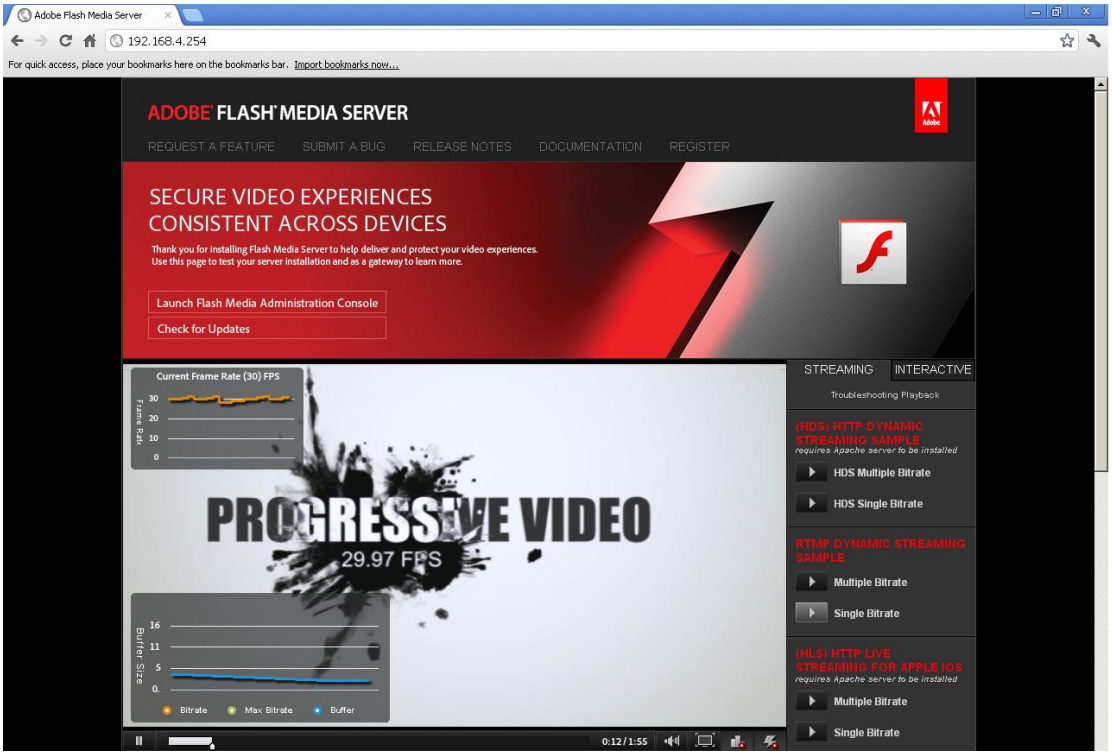
- Η λειτουργία του DASH κρίθηκε πετυχημένη και ολοκληρώθηκαν τα πειράματα χωρίς προβλήματα καθώς η συγκεκριμένη πλατφόρμα δουλεύει ήδη επαγγελματικά.
- Η συγκεκριμένη πλατφόρμα προσφέρεται από την Adobe δωρεάν για κάθε ερευνητικό, πειραματικό σκοπό.
- Όλα τα εγχειρίδια χρήσης καθώς και μια μεγάλη γκάμα από έγγραφα άλλων χρηστών βρέθηκαν στη διάθεσή μας δωρεάν και προσέφεραν μια πολύτιμη βοήθεια.
- Συμβατότητα με τα περισσότερα λειτουργικά συστήματα που τρέχουν flash players σε κινητές και σταθερές συσκευές.
- Ένα σημαντικό κομμάτι από την πλευρά του client είναι το DASH player πρόγραμμα, που είναι ανοιχτού κώδικα από την Open Source Media Framework και εκτός ότι προσφέρει έτοιμα sample players για την αναπαραγωγή πολυμεσικού περιεχομένου υπάρχει και πρόσβαση στον κώδικα έτσι ώστε να τροποποιηθεί κατάλληλα για διάφορες εφαρμογές.

5.2 Περιβάλλον

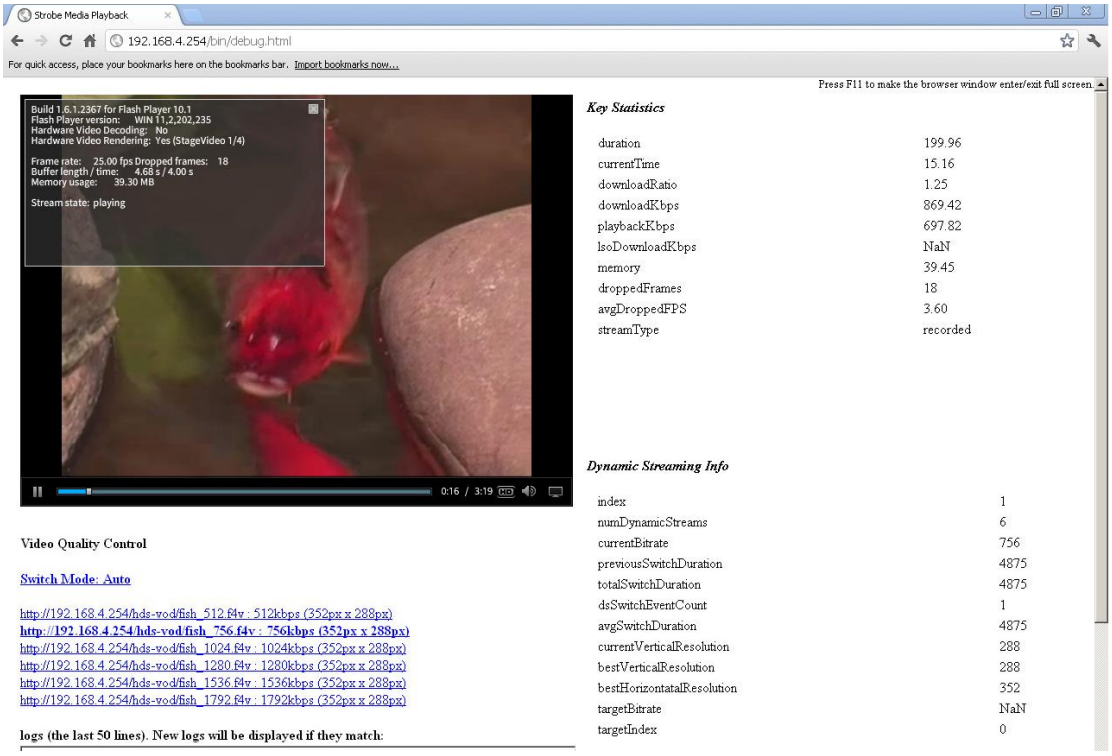
Για το περιβάλλον στο οποίο εκτελέστηκε το πείραμα χρησιμοποιήθηκε ένα windows xp pc ως client, ένα Flash Media server με λειτουργικό windows και ανάμεσά τους έναν Dummynet System βασισμένο σε λειτουργικό σύστημα Unix.

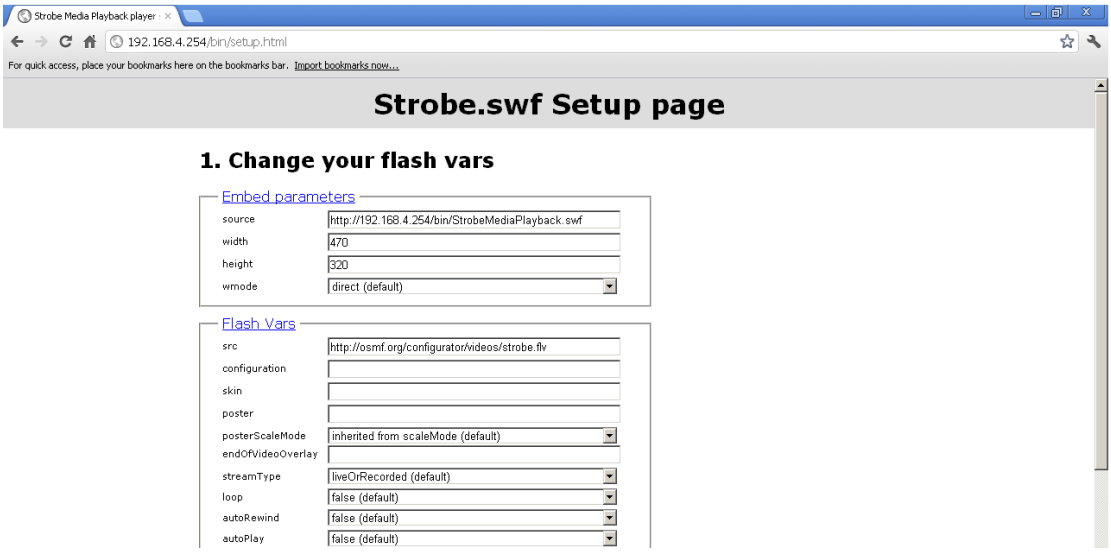


Στον client εγκαταστάθηκε η τελευταία έκδοση flash player και το google chrome χρησιμοποιήθηκε ως browser και περιήγηση στις σελίδες με το πολυμεσικό υλικό που περιέχει ο server. Στο server εγκαταστάθηκε η τελευταία έκδοση flash media server 4.5 για να υποστηρίξει τον adobe Dash

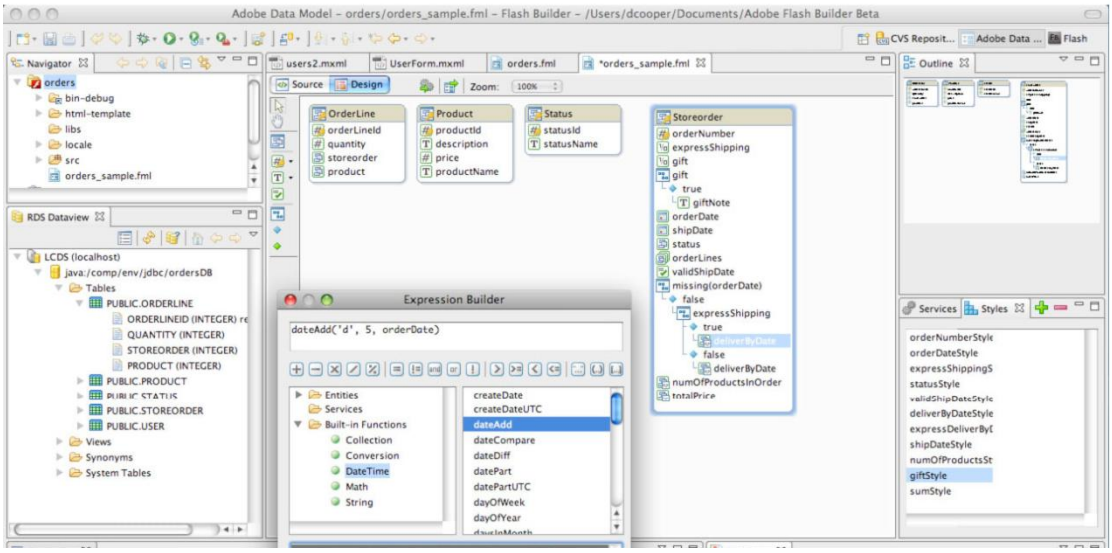


και έπειτα για την αναπαραγωγή του video streaming χρησιμοποιήθηκε το ανοιχτό OSMF 1.6.1 από το οποίο επιλέχθηκε το StrobeMediaPlayback player σε debug μορφή για πρακτικούς λόγους.

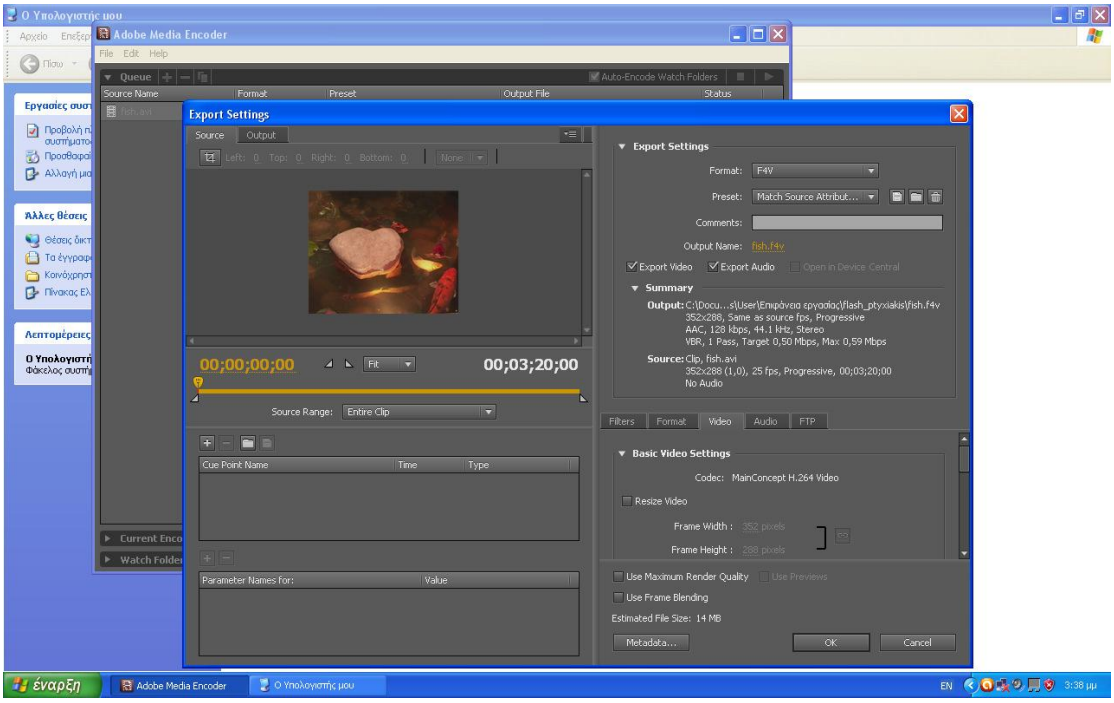




Για την επεξεργασία του κώδικα και το χτίσιμο του StrobeMediaPlayback χρησιμοποιήθηκε το Abobe Flash builder 4.5

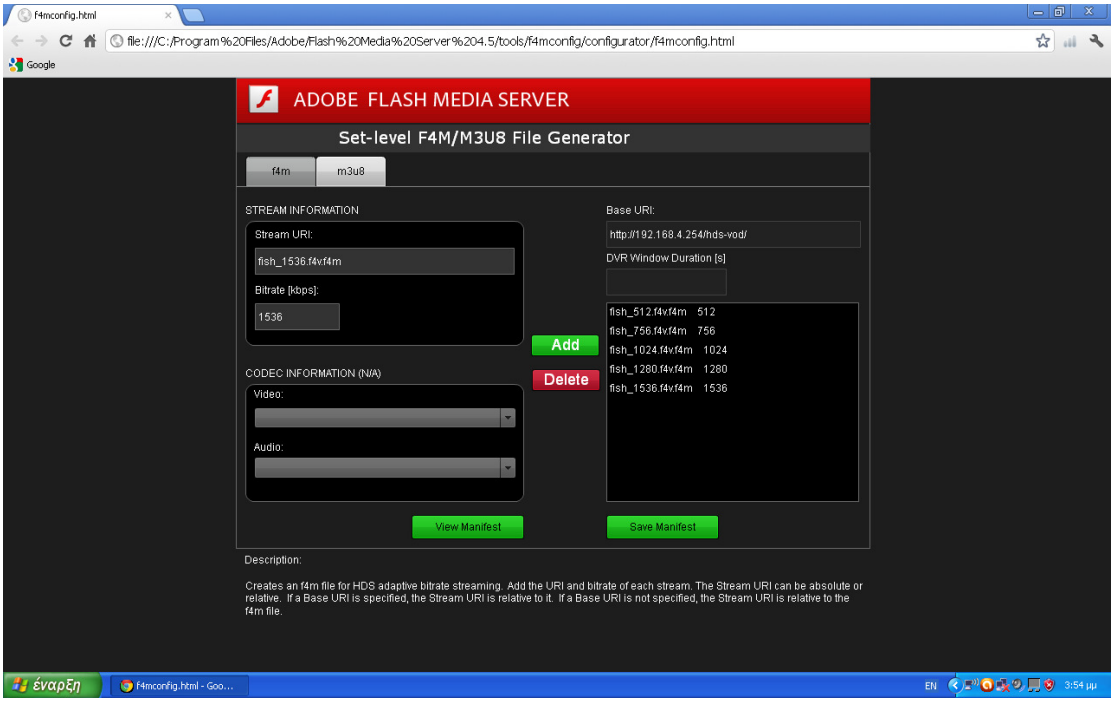


Στο πείραμα χρησιμοποιήθηκε ένα βίντεο με ψάρια τα οποία κωδικοποιήθηκε μέσω του Adobe Media Encoder CS5.5

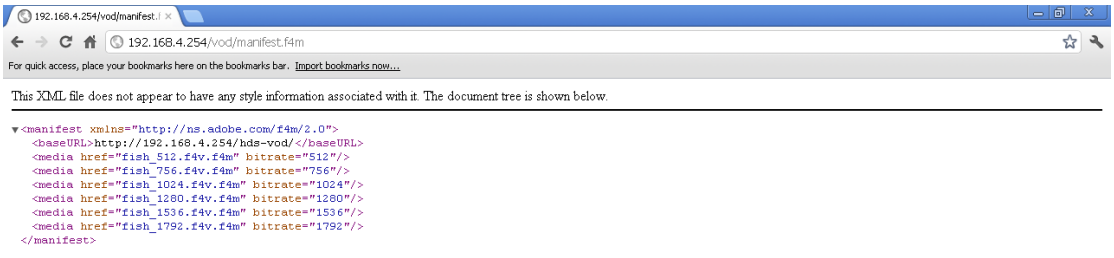


Η μορφή που κωδικοποιήθηκε το βίντεο είναι .f4v σε έξι διαφορετικές ποιότητες

512,756,1024,1280,1536 και 1792



από τη χειρότερη στην καλύτερη και με τη χρήση του εργαλείου F4M File Generator δημιουργήθηκε το αρχείο δομής περιεχομένου ή manifest file σε μορφή xml

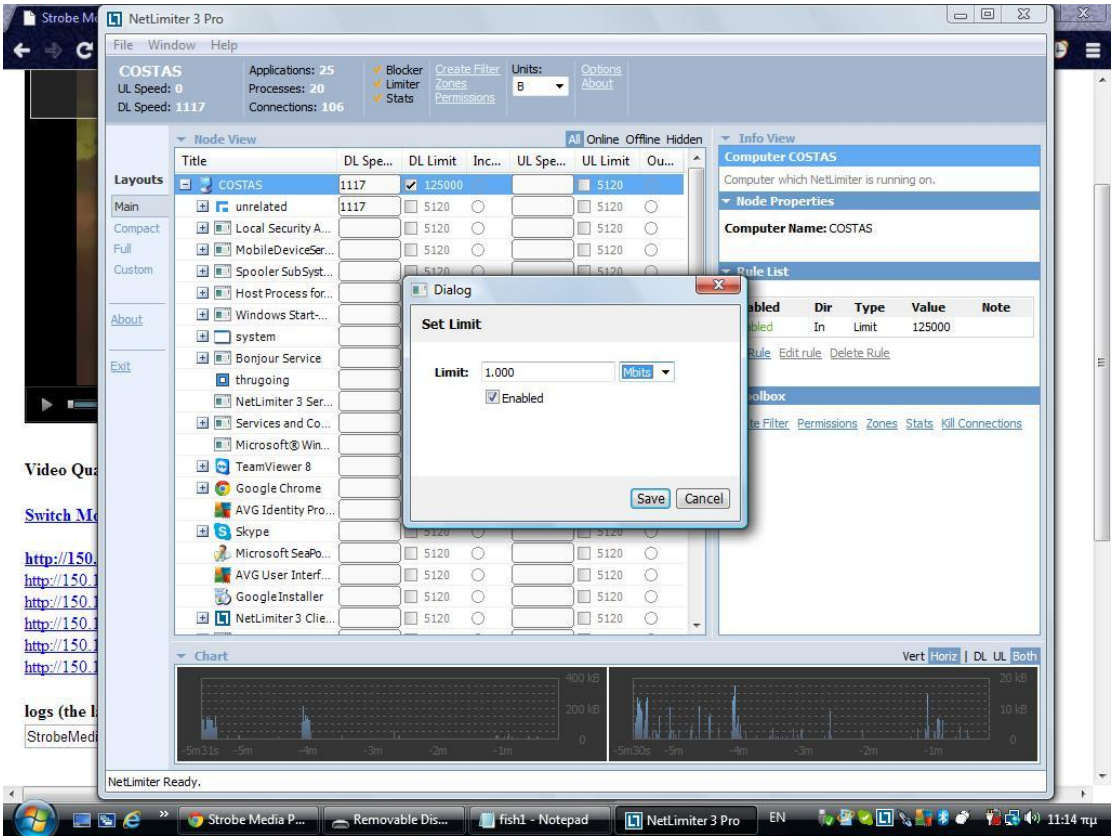


5.3 Το πείραμα

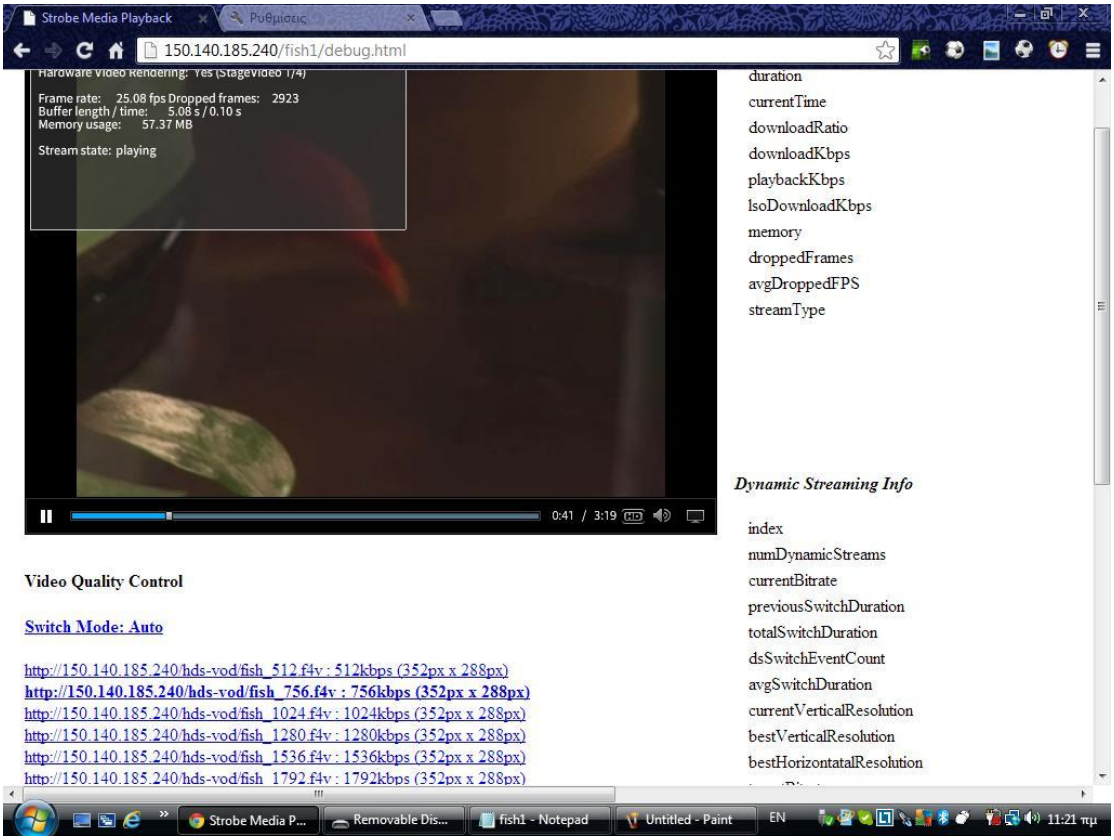
Στο πειραματικό κομμάτι αυτό που είχαμε ως στόχο ήταν να χρησιμοποιήσουμε ένα βίντεο το οποίο θα είχε πρόσβαση ένας client με συγκεκριμένη σύνδεση (2Mbs) ώστε με διάφορες τεχνιτές παρεμβολές (cross traffic) σε αυτό να δούμε πόσο μεταβάλλεται η ποιότητά του και αν αυτό γίνεται αντιληπτό από το χρήστη στην ποιότητα του βίντεο. Το βίντεο με τα ψάρια πάρθηκε από το cdvl.org.

Στόχος είναι να υπάρξει ισορροπία ανάμεσα σε δύο στοιχεία για να μην υπάρχουν μεταβολές λαμβάνοντας υπόψη index και throughput ώστε να μην έχουμε over provisioning και under provisioning.

Στην αρχή έγινε εγκατάσταση στον client του προγράμματος net limiter3 για να συνδέεται με συγκεκριμένο bandwidth ο client στο δίκτυο. Επιλέχθηκε το 1 Mbs ως ταχύτητα σε ένα πρώτο πείραμα.



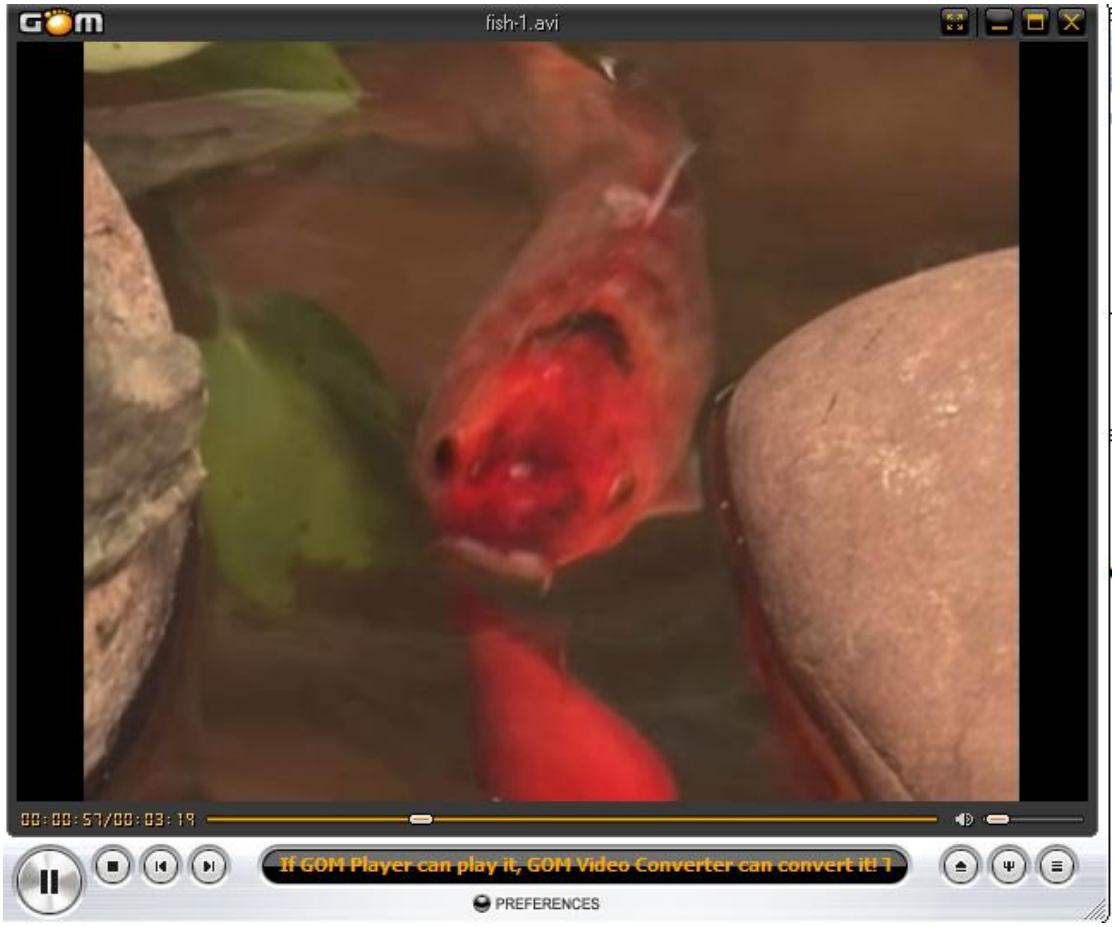
Αφού κάναμε εκκαθάριση τον cookies συνδεθήκαμε στον 150.140.185.240 και παρακολουθήσαμε το βίντεο με τα ψάρια. Αυτό που είναι παρατηρήθηκε είναι ότι το download του βίντεο δεν περνά τα 756kbs όπως φαίνεται και στην παρακάτω οθόνη



Το βίντεο είναι 5000 frames και εκπέμπονται 25frame/ second. Όλο το βίντεο είναι 200 second

Σε δεύτερη φάση αυτό που ήταν ως στόχος ήταν να κρατήσουμε σταθερή την ταχύτητα του client κάτι που επιτεύχθηκε μέσα από το πρόγραμμα net limiter αλλά και να πειραχθεί το cross traffic του δικτύου κάτι δηλαδή που θα «κλέψει bandwidth» την ώρα που εκπέμπεται το βίντεο ώστε να δούμε τη συμπεριφορά του. Αυτό επιτεύχθηκε με το πρόγραμμα iperf που τρέξαμε με προσομοίωση μέσα από το cygwin.

Στις ρυθμίσεις του iperf μπήκε κίνηση 0,6 εως 1,4 αυξανόμενο κατά 0,2 ανά δευτερόλεπτο και για κάθε Mbps είχε διάρκεια 5 seconds. Το bandwidth είναι σταθερό στα 2Mbps πλέον.



Με βάση τον τύπο **Selected Quality= w1* Calculated Bit Rate+w2*Measured Throughput/(w1+w2)**, έχουμε την πειραματική διαδικασία ώστε να επιλεγεί ένα τελικό bitrate και εν τέλει η ποιότητα που παίζει το βίντεο κάθε στιγμή.

- όπου w1, w2 είναι τα βάρος 1 και 2 και μπορεί να πάρει τιμές από 0 εως 1 το **throughput** είναι το διαθέσιμο bandwidth μια προηγούμενη χρονική στιγμή και το index είναι το bandwidth τώρα.
- Ανάλογα λοιπόν τα βάρη επιλέγεται ποια ποιότητα τελικά θα παίξει το βίντεο κάθε χρονική στιγμή

Έγιναν μετρήσεις με τα εξής ζευγάρια από βάρη με βάση τον παραπάνω τύπο

w1	w2
0,2	0,8
0,5	0,5
0,8	0,2

και εξάχθηκαν μετρήσεις στατιστικά και συμπεράσματα.

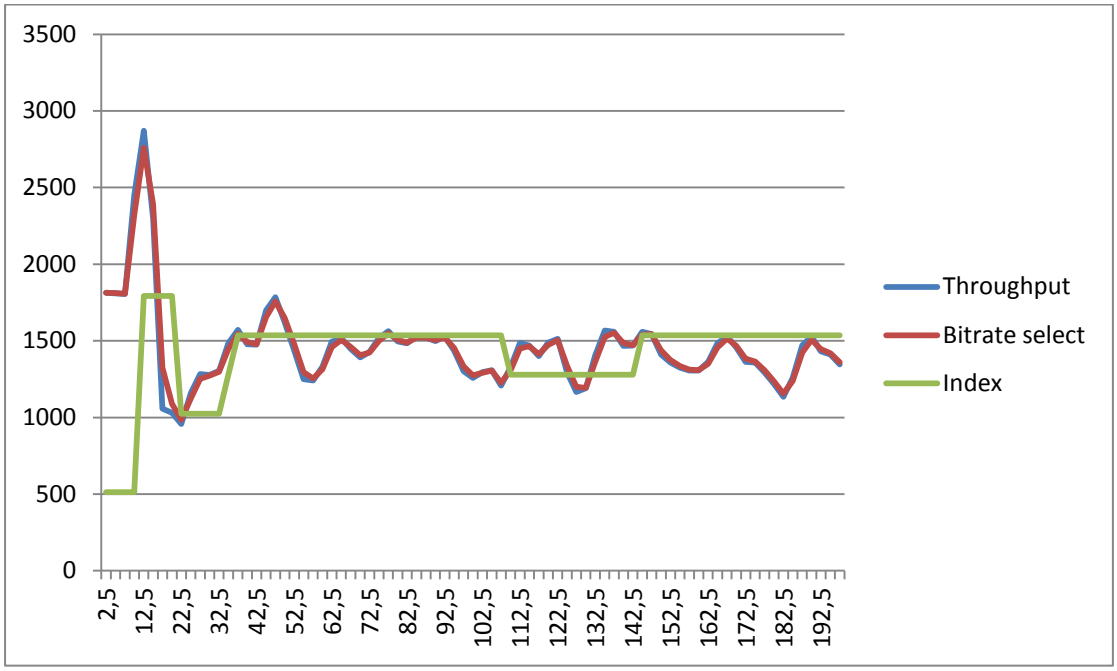
Ποιο συγκεκριμένα στη πρώτη περίπτωση έχουμε το βίντεο με τα εξής χαρακτηριστικά όπως αυτά ομαδοποιήθηκαν από τα logs :

fish w1=0,2 w2=0,8

droppedFrames 2
Index duration= 0:8, 1:0, 2:9, 3:30, 4:99, 5:10
Index switch time=0:0, 5:10.44, 2:22.198, 3:33.96, 4:37.92, 3:109.914, 4:145.195
start 3:42:33 πμ

round bw		round bit		cross		index		time
1813		1813		1000		512		2,5
1810		1811		1200		512		5
1806		1807		1400		512		7,5
2448		2320		1400		512		10
2870		2760		600		1792		12,5
2299		2391		600		1792		15
1057		1324		800		1792		17,5
1030		1088		800		1792		20
958		984		1000		1024		22,5
1161		1125		1000		1024		25
1283		1252		1200		1024		27,5
1277		1272		1200		1024		30
1305		1298		1400		1024		32,5
1483		1446		1400		1280		35
1572		1547		600		1536		37,5
1477		1491		600		1536		40
1475		1478		800		1536		42,5
1699		1655		800		1536		45
1784		1758		1000		1536		47,5
1622		1649		1000		1536		50
1435		1478		1200		1536		52,5
1249		1295		1200		1536		55
1242		1253		1400		1536		57,5
1331		1315		1400		1536		60
1494		1458		600		1536		62,5
1518		1506		600		1536		65
1449		1460		800		1536		67,5
1392		1406		800		1536		70
1427		1423		1000		1536		72,5
1518		1499		1000		1536		75
1564		1551		1200		1536		77,5
1496		1507		1200		1536		80
1483		1488		1400		1536		82,5
1529		1521		1400		1536		85
1522		1522		600		1536		87,5
1499		1504		600		1536		90
1527		1522		800		1536		92,5
1437		1454		800		1536		95

1303		1333		1000		1536		97,5
1259		1274		1000		1536		100
1296		1291		1200		1536		102,5
1309		1306		1200		1536		105
1207		1226		1400		1536		107,5
1330		1309		1400		1280		110
1486		1450		600		1280		112,5
1469		1465		600		1280		115
1400		1413		800		1280		117,5
1488		1473		800		1280		120
1512		1504		1000		1280		122,5
1296		1337		1000		1280		125
1166		1200		1200		1280		127,5
1192		1193		1200		1280		130
1405		1363		1400		1280		132,5
1567		1526		1400		1280		135
1559		1552		600		1280		137,5
1468		1485		600		1280		140
1469		1472		800		1280		142,5
1558		1541		800		1536		145
1545		1544		1000		1536		147,5
1411		1437		1000		1536		150
1359		1375		1200		1536		152,5
1324		1335		1200		1536		155
1307		1313		1400		1536		157,5
1307		1308		1400		1536		160
1362		1351		600		1536		162,5
1483		1457		600		1536		165
1529		1514		800		1536		167,5
1457		1468		800		1536		170
1363		1384		1000		1536		172,5
1359		1364		1000		1536		175
1294		1308		1200		1536		177,5
1220		1238		1200		1536		180
1134		1155		1400		1536		182,5
1259		1238		1400		1536		185
1471		1424		600		1536		187,5
1528		1507		600		1536		190
1432		1447		800		1536		192,5
1411		1418		1000		1536		195
1345		1360		1000		1536		197,5



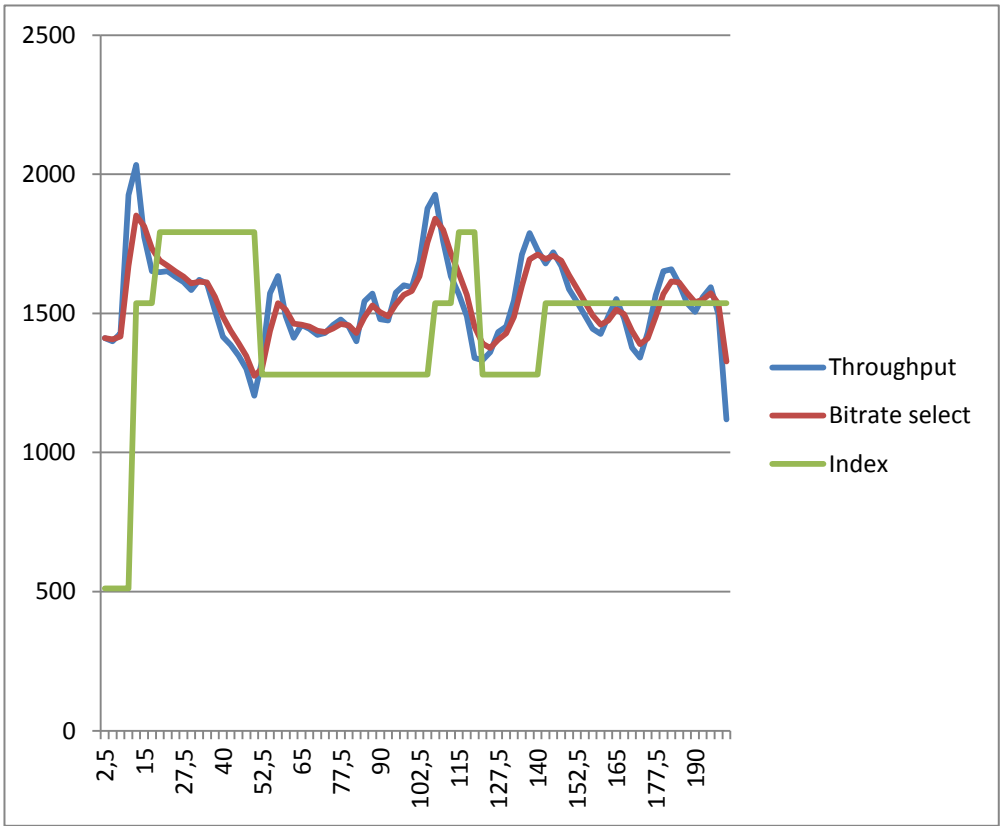
Στη δεύτερη περίπτωση έχουμε :

fish w1=0,5 w2=0,5
droppedFrames 3
Index duration= 0:8, 1:0, 2:0, 3:59, 4:59, 5:31
Index switch time=0:0, 4:10.999, 5:18.795, 3:50.231, 4:106.514, 5:114.437, 3:122.231, 4:141.91
start 05:15:27

round bw		round bit		cross		index		time
1411		1411		1200		512		2,5
1400		1406		1400		512		5
1428		1417		1400		512		7,5
1925		1671		600		512		10
2033		1852		600		1536		12,5
1775		1813		800		1536		15
1652		1733		800		1536		17,5
1648		1690		1000		1792		20
1652		1671		1000		1792		22,5
1631		1651		1200		1792		25
1613		1632		1200		1792		27,5
1584		1608		1400		1792		30
1620		1614		1400		1792		32,5
1608		1611		600		1792		35
1510		1560		600		1792		37,5
1416		1488		800		1792		40
1387		1438		800		1792		42,5
1348		1393		1000		1792		45
1301		1347		1000		1792		47,5
1204		1275		1200		1792		50
1329		1302		1200		1280		52,5
1572		1437		1400		1280		55
1634		1536		1400		1280		57,5

1490		1513		600		1280		60
1412		1463		600		1280		62,5
1458		1460		800		1280		65
1445		1452		800		1280		67,5
1423		1438		1000		1280		70
1429		1433		1000		1280		72,5
1458		1446		1200		1280		75
1478		1462		1200		1280		77,5
1453		1457		1400		1280		80
1400		1429		1400		1280		82,5
1543		1486		600		1280		85
1571		1528		600		1280		87,5
1479		1504		800		1280		90
1474		1489		800		1280		92,5
1575		1532		1000		1280		95
1601		1566		1000		1280		97,5
1594		1580		1200		1280		100
1686		1633		1200		1280		102,5
1877		1755		1400		1280		105
1926		1840		1400		1536		107,5
1759		1800		600		1536		110
1633		1716		600		1536		112,5
1569		1643		800		1792		115
1489		1566		800		1792		117,5
1340		1453		1000		1792		120
1332		1392		1000		1280		122,5
1360		1376		1200		1280		125
1433		1405		1200		1280		127,5
1452		1428		1400		1280		130
1546		1487		1400		1280		132,5
1711		1599		600		1280		135
1789		1694		600		1280		137,5
1728		1711		800		1280		140
1679		1695		800		1536		142,5
1719		1707		1000		1536		145
1670		1689		1000		1536		147,5
1587		1638		1200		1536		150
1541		1589		1200		1536		152,5
1494		1542		1400		1536		155
1445		1493		1400		1536		157,5
1426		1460		600		1536		160
1489		1474		600		1536		162,5
1552		1513		800		1536		165
1483		1498		800		1536		167,5
1377		1437		1000		1536		170
1341		1389		1000		1536		172,5
1431		1410		1200		1536		175
1565		1488		1200		1536		177,5

1652		1570		1400		1536		180
1659		1615		1400		1536		182,5
1607		1611		600		1536		185
1536		1573		600		1536		187,5
1506		1540		800		1536		190
1558		1549		1000		1536		192,5
1594		1572		1000		1536		195
1496		1534		1200		1536		197,5
1119		1327		1200		1536		200



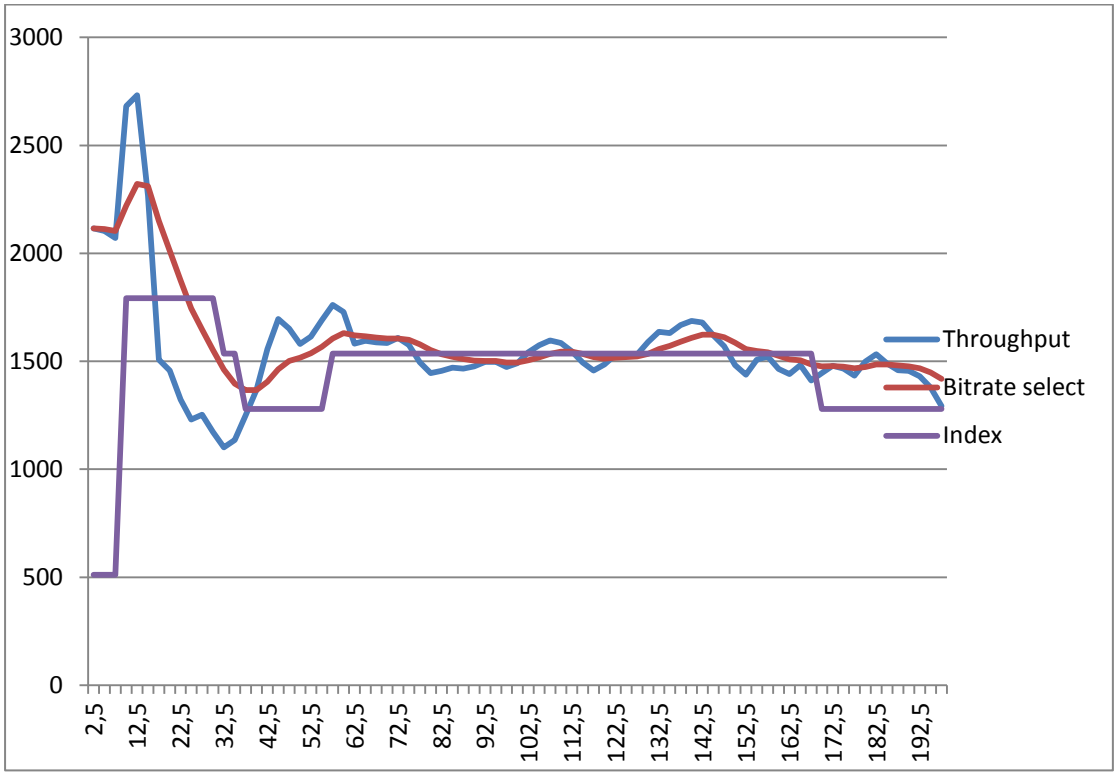
Και στη τελευταία περίπτωση έχουμε

fish w1=0,8 w2=0,2
droppedFrames 4
Index duration= 0:8, 1:0, 2:0, 3:39, 4:95, 5:16
Index switch time=0:0, 5:10.507, 4:30.063, 3:37.98, 4:57.66, 3:170.22
start 06:54:48

round bw		round bit		cross		index		time
2115		2115		1000		512		2,5

2104		2113		1200		512		5
2071		2104		1200		512		7,5
2681		2220		1400		1792		10
2732		2322		1400		1792		12,5
2266		2311		600		1792		15
1507		2150		600		1792		17,5
1458		2012		800		1792		20
1322		1874		800		1792		22,5
1231		1745		1000		1792		25
1252		1647		1000		1792		27,5
1173		1552		1200		1792		30
1101		1462		1200		1536		32,5
1136		1397		1400		1536		35
1250		1367		1400		1280		37,5
1368		1367		600		1280		40
1556		1405		600		1280		42,5
1696		1463		800		1280		45
1652		1501		800		1280		47,5
1581		1517		1000		1280		50
1615		1537		1000		1280		52,5
1690		1567		1200		1280		55
1761		1606		1200		1536		57,5
1729		1631		1400		1536		60
1582		1621		1400		1536		62,5
1595		1616		600		1536		65
1587		1610		600		1536		67,5
1585		1605		800		1536		70
1608		1605		800		1536		72,5
1576		1600		1000		1536		75
1496		1579		1000		1536		77,5
1445		1552		1200		1536		80
1456		1533		1200		1536		82,5
1471		1521		1400		1536		85
1466		1510		1400		1536		87,5
1476		1503		600		1536		90
1498		1502		600		1536		92,5
1497		1501		800		1536		95
1473		1495		800		1536		97,5
1491		1494		1000		1536		100
1542		1504		1000		1536		102,5
1575		1518		1200		1536		105
1596		1534		1200		1536		107,5
1585		1544		1400		1536		110
1546		1544		1400		1536		112,5
1494		1534		600		1536		115
1458		1519		600		1536		117,5
1485		1512		800		1536		120
1530		1516		800		1536		122,5

1533		1519		1000		1536		125
1532		1522		1000		1536		127,5
1589		1535		1200		1536		130
1637		1556		1200		1536		132,5
1630		1571		1400		1536		135
1668		1590		1400		1536		137,5
1687		1609		600		1536		140
1680		1623		600		1536		142,5
1620		1623		800		1536		145
1570		1612		800		1536		147,5
1483		1586		1000		1536		150
1438		1557		1000		1536		152,5
1507		1547		1200		1536		155
1522		1542		1200		1536		157,5
1464		1526		1400		1536		160
1441		1509		1400		1536		162,5
1482		1504		600		1536		165
1411		1485		600		1536		167,5
1445		1477		800		1280		170
1480		1478		800		1280		172,5
1466		1475		1000		1280		175
1433		1467		1000		1280		177,5
1499		1473		1200		1280		180
1533		1485		1200		1280		182,5
1490		1486		1400		1280		185
1459		1481		1400		1280		187,5
1456		1476		600		1280		190
1430		1467		600		1280		192,5
1378		1449		800		1280		195
1293		1418		800		1280		197,5



Συμπεράσματα

Αυτό που παρατηρούμε είναι ότι οι τιμές του throughput, Bitrate και του Index μετριόνται κάθε 2,5 δευτερόλεπτα και βλέπουμε ότι η ισορροπία throughput/index που είχαμε ως στόχο να μελετηθεί εξαρτάται κατά κύριο λόγο από τις τιμές βάρη w1 και w2. Η ποιότητα του βίντεο και αυτό είναι κάτι που έχει σχεδιαστεί ο αλγόριθμος να κάνει, έχει διάρκεια chunk ίση με 4 seconds . Το βίντεο μπορεί να μεταβληθεί οποιαδήποτε άλλη στιγμή ή και καθόλου. Αν δεν αλλάξει το throughput τότε το index παραμένει σταθερό καθ' όλη τη διάρκεια του βίντεο. Όπως απεικονίζεται και στις γραφικές παραστάσεις όταν τα βάρη είναι ισότιμα το βίντεο προβάλλεται χωρίς ιδιαίτερες μεταβολές. Πρόκειται για μια ενδιάμεση περίπτωση όπου το bitrate ακολουθεί αποκλειστικά το throughput ούτε όμως δεν το ακολουθεί και καθόλου . Η επιλογή του bitrate της φόρμουλας επηρεάζεται και από το παλιό και το τωρινό throughput και ίσως και η τιμή του index που είναι ακόμα πιο αργή απόφαση. Όταν το w1>w2 τότε το bitrate αργεί να ακολουθήσει το throughput και οι μεταβολές γίνονται πιο ομαλά. Όταν το w1<w2 τότε οι μεταβολές γίνονται πιο απότομα σε σχέση με το bitrate που ακολουθεί το throughput.. Σε κάθε περίπτωση δεν είναι κάτι που γίνεται αντιληπτό με γυμνό μάτι παρά μόνο μέσα από μετρήσεις. Ο DASH έχει σχεδιαστεί έτσι ώστε σε κάθε περίπτωση η ποιότητα να επιλέγεται από το throughput . Για χάρη του πειράματος το bitrate να ακολουθεί την τιμή του throughput και με εξακολούθηση η τιμή index ακολουθεί αυτή της bitrate. Η τελική επιλογή είναι να έχει την κοντινότερη τιμή, που συνήθως είναι η μικρότερη δυνατή γιατί αν πάει μεγαλύτερη τότε το βίντεο θα πιέζεται και θα κολλάει. Ο player πέρα από τις μεταβολές throughput φροντίζει να υπάρχει ένα ομαλό playback με πολύ λιγότερες μεταβολές από αυτές του throughput. Η φόρμουλα παίζει έμμεσο ρόλο στο playback πχ εκεί που θα είχαμε 50 αλλαγές index τώρα έχουμε 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/+__"_____(υ_____)
[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-bugzillagnome-
org
[24] http://gstreamer.freedesktop.org/conference/speakers.html
[25] http://noc.auth.gr/services/voice-video/mbone/index.html

[26] Luca De Cicco, Saverio Mascolo, Vittorio Palmisano, “**Feedback Control for Adaptive Live Video Streaming**”, Proceedings of USAB WIMA, 2010

[27] Ali C. Begen. **Watching Video over the Web : Adaptive Streaming over HTTP**

[28] Yago Sánchez, Thomas Schierl, Cornelius Hellge, Thomas Wiegand, Fraunhofer HHI, Germany

Dohy Hong N2N Soft, France. Danny De Vleeschauwer, Werner Van Leekwijck Bell Labs - Alcatel

Lucent, Belgium. Yannick Le Louédec Orange-FT, France “iDASH: Improved Dynamic Adaptive

Streaming over HTTP using Scalable Video Coding”

Παράρτημα Α

Παρατίθεται ο κώδικας της κλάσης που αναπτύσσει και εκτελεί τον αλγόριθμο που κάθε φορά ρυθμίζει την ποιότητα στην οποία εκπέμπεται τελικά το βίντεο με όλες τις παραμέτρους που αναπτύχθηκαν στην εργασία

```
/**********************************************************

 * Copyright 2010 Adobe Systems Incorporated.  All Rights Reserved.

 *

 * *********************************************************

 * The contents of this file are subject to the Berkeley Software Distribution (BSD) Licence

 * (the "License"); you may not use this file except in

 * compliance with the License.

 *

 * Software distributed under the License is distributed on an "AS IS"

 * basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the

 * License for the specific language governing rights and limitations

 * under the License.

 *

 *

 * The Initial Developer of the Original Code is Adobe Systems Incorporated.

 * Portions created by Adobe Systems Incorporated are Copyright (C) 2010 Adobe Systems

 * Incorporated. All Rights Reserved.

 **********************************************************/


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.net.SharedObject;
import flash.system.System;
import flash.utils.Dictionary;
import flash.utils.Timer;
import flash.utils.getTimer;

import org.osmf.player.utils.StrobeUtils;
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);
                m= new PlaybackOptimizationMetrics(netStream);
                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;

                indexDur[pcurrentIndex]+=((getTimer()-start)/1000);
                start=getTimer();
                str=" "+0+":"+indexDur[0];
                for (var j:int = 1; j < dsResource.streamItems.length; j++)
                {
                        str=str+(", "+j+":"+indexDur[j]);
                }

                CONFIG::LOGGING
                {
                        debug("------Index duration="+str);


                }
                CONFIG::LOGGING
                {
                        debug("------Index time="+indexPlay);
```

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

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

                bw=m.averageDownloadKbps;

        if((isNaN(bw))||(bw==Number.NEGATIVE_INFINITY)||(bw==Number.POSITIV
E_INFINITY)){
                        bw=dsResource.streamItems[0].bitrate;
                }
                if(flag==1)
                {
                        previousb=bw;
                        time=getTimer();
                        flag=0;
                }
                bitrate=(w1*previousb+w2*bw)/(w1+w2);
                var i:int = 1;
```

```
        while((dsResource.streamItems[dsResource.streamItems.length-
i].bitrate>bitrate)&&(i!=dsResource.streamItems.length))
                        {
                                i++;
                        }

                        newIndex=dsResource.streamItems.length-i;
                        if(pindex!=newIndex){
                                switchcount++;
                                switchCounts[newIndex]++;
                        }
                        CONFIG::LOGGING
                        {
                                debug("------SumSwitches: "+switchcount);
                        }



                        if((getTimer()-time)>=60000){

                                sumswitchcount+=(switchcount-pindexcount);
                                CONFIG::LOGGING
                                {
                                        debug("------1minSwitches: "+(switchcount-
pindexcount)+" Sum: "+sumswitchcount);
                                }

                                pindexcount=switchcount;
                                time=getTimer();
```

```
}



str=" "+0+":"+switchCounts[0];
for ( j = 1; j < dsResource.streamItems.length; j++)
        {
        str=str+(", "+j+":"+switchCounts[j]);
        }

CONFIG::LOGGING
{
        debug("------Switch Indexes="+str);



}
pindex=newIndex;
//newIndex = 4;
previousb=bitrate;
CONFIG::LOGGING
{
        debug("----BW: "+bw+" BITRATE: " + bitrate+" INDEX:
"+dsResource.streamItems[dsResource.streamItems.length-i].bitrate);
}




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:
                    start=getTimer();
                    start2=start;
                    indexPlay="0:0";
                    CONFIG::LOGGING
                {
                    debug("---------START=" + start);
                }
                    if (metrics.resource == null)
                    {
                        prepareForSwitching();
                        indexDur[pcurrentIndex]+=((getTimer()-
start)/1000);
                        start=getTimer();
                    }
                    else if (autoSwitch && checkRulesTimer.running
== false)
                    {
                        start=getTimer();
                        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;
                                case NetStreamCodes.NETSTREAM_PAUSE_NOTIFY:
                                        checkRulesTimer.stop();
                                        break;
                                case NetStreamCodes.NETSTREAM_PLAY_COMPLETE:
```

```
                    checkRulesTimer.stop();
                    break;
                case NetStreamCodes.NETSTREAM_UNPAUSE_NOTIFY:
                    checkRulesTimer.start();
                    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");
                                        }

                                        indexPlay=indexPlay+(",
"+currentIndex+":"+((getTimer()-start2)/1000));
                                        indexDur[pcurrentIndex]+=((getTimer()-
start)/1000);
                                        start=getTimer();
                                        pcurrentIndex=currentIndex;
                                        break;
                        }
                }

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

        {

                if (switchCounts != null)

                {

                        switchCounts.length = 0;

                        switchCounts = null;

                }
```

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

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

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

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);

                }

                }


                /**

                 * 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 averageDownloadBytesPerSecond;
        }

        public function set
averageDownloadBytesPerSecond(value:Number):void
        {
                averageDownloadBytesPerSecond = value;
                //return averageDownloadBytesPerSecond;
        }

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




        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 index:int = 0;///giorgos

private var time:int = 0;///giorgos

private var start:int = 0;///giorgos

private var start2:int = 0;///giorgos

private var finish:int = 0;///giorgos

private var pcurrentIndex:int = 0;///giorgos

private var pindex:int = 0;///giorgos

private var switchCounts:Vector.<int>;///giorgos

private var indexDur:Vector.<int>;///giorgos

private var switchcount:int = 0;///giorgos

private var sumswitchcount:int = 0;///giorgos

private var pindexcount:int = 0;///giorgos

private var w1:Number = 0.8;///giorgos

private var str:String;///giorgos

private var indexPlay:String;///giorgos

private var w2:Number = 0.2;///giorgos

private var bitrate:Number = 0;///giorgos

private var previousb:Number = 0;///giorgos

private var flag:int = 1;///giorgos

private var m:PlaybackOptimizationMetrics;///giorgos

private var bw:Number = 0;///giorgos

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;

                }

        }

}
```