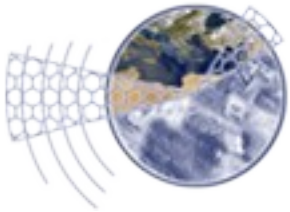
Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων



Πτυχιακή εργασία

Ψηφιακή Πλατφόρμα Ασύγχρονης Διαδικτυακής Τηλεόρασης

Άγγελος Βεγλεκτσής (1871)

Επιβλέπων καθηγητής : Κορνάρος Γιώργος

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Κορνάρο Γιώργο για τη δυνατότητα που μου έδωσε να ασχοληθώ με τη συγκεκριμένη πτυχιακή εργασία, αλλά και για την υποστήριξη και κατεύθυνση του στη διάρκεια των σπουδών μου στο ΤΕΙ Κρήτης. Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές μου, για τις απαραίτητες γνώσεις και τα κατάλληλα εφόδια που μου μετέδωσαν. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου για την αδιάκοπη υποστήριξη τους τόσο οικονομικά όσο συναισθηματικά, δίνοντας μου τη δυνατότητα να ολοκληρώσω με επιτυχία τις σπουδές μου.

Σύνοψη

Η δημιουργία της ψηφιακής πλατφόρμας ασύγχρονης τηλεόρασης ήταν αποτέλεσμα της ανάγκης για παροχή οπτικοακουστικού περιεχομένου ξεφεύγοντας από τα παραδοσιακά μέσα μετάδοσης. Προσφέροντας μια καινούργια εμπειρία στους χρήστες, δίνοντας στους μεγαλύτερη ευκολία στην πρόσβαση και παρακολούθηση του τηλεοπτικού περιεχομένου.

Για την επιλογή μιας βιώσιμης λύσης από πλευράς κόστους παροχής των υπηρεσιών, έγινε προσεκτική επιλογή των τεχνολογιών που θα έδιναν μεγάλη ευελιξία στην κατασκευή της πλατφόρμας με αρθρωτά συστήματα τα οποία θα λειτουργούσαν ανεξάρτητα.

Η Ψηφιακή Πλατφόρμα Ασύγχρονης Διαδικτυακής Τηλεόρασης είναι ένα σύνολο τεχνολογιών που μέσω του διαδικτύου δίνει τη δυνατότητα στους χρήστες να παρακολουθήσουν τηλεοπτικό υλικό με ασύγχρονο τρόπο, δηλαδή δίνεται η δυνατότητα προβολής τηλεοπτικού περιεχομένου το οποίο είχε προβληθεί σε προγενέστερο χρόνο, από τα παραδοσιακά μέσα μετάδοσης.

Για να καλύψουμε τα πιο διαδεδομένα μέσα που χρησιμοποιούν οι χρήστες, αναπτύχθηκε μια πλατφόρμα που μπορεί να δώσει πρόσβαση στο ασύγχρονο τηλεοπτικό περιεχόμενο με αρκετούς τρόπους. Η πρόσβαση στην πλατφόρμα γίνεται μέσω ειδικής συσκευής (Set-top-Box) που συνδέεται απευθείας στην τηλεόραση του χρήστη ή για λόγους κάλυψης των αναγκών των χρηστών, με την χρήση προγράμματος web browser. Η συσκευή Set-top-Box μπορεί να είναι μια συσκευή που δημιουργήθηκε εξ' ολοκλήρου για αυτόν τον σκοπό ή χρησιμοποιώντας άλλες παρόμοιες συσκευές που υπάρχουν στην αγορά.

Abstract

The creation of asynchronous digital television platform was a result of the need for offering audiovisual content apart from the traditional media transmission. Offering a new experience to users and giving ease of access and viewing of television content.

In order to create viable solution in terms of cost of performance, the technologies was carefully selected to provide great flexibility in constructing the platform with modular systems that can operate independently.

The asynchronous digital television platform is a set of technologies that through an Internet connection enables users to watch video material in asynchronous mode, i.e. to have the ability to view video content which was screened at an earlier time than traditional transmission.

To cover the most popular devices used by users,we developed a platform that can provide asynchronous access to television content in several ways. Access to the platform can be achieved using a web browser or through a device (Set-top-Box) connected directly to the TV. The Set-top-Box can be a device that was created entirely for this purpose or using other similar devices on the market.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

1 Εισαγωγή

Η πτυχιακή εργασία είναι ένα από τα σημαντικότερα μέρη του πτυχίου. Δίνει την ευκαιρία στον φοιτητή να κάνει έρευνα και να δώσει λύσεις σε πολύ σημαντικά προβλήματα. Εκτός από την εμπειρία που αποκομίζει ο φοιτητής από την εκπόνηση της πτυχιακής εργασίας, τον βοηθά να έχει διερευνητική σκέψη και να μπορεί μόνος του να βγάζει συμπεράσματα και να τα εκθέτει ορθά.

Η εκπόνηση της συγκεκριμένης πτυχιακής εργασίας έγινε με κυρίαρχη σκέψη την δημιουργία ενός πλήρους λειτουργικού πρωτοτύπου έτσι ώστε να είναι εφικτή η χρήση του σε πραγματικό περιβάλλον.

Στα κεφάλαια που ακολουθούν δίνεται πλήρη εικόνα για τον τρόπο υλοποίησης, τους λόγους επιλογής της κάθε τεχνολογίας καθώς και σύγκριση της απόδοσης του συστήματος. Η δημιουργία μιας πλατφόρμας ψηφιακής τηλεόρασης περιλαμβάνει πολλές πτυχές της λειτουργίας ενός συστήματος πληροφορικής. Για τον λόγο αυτό έχουν χρησιμοποιηθεί και αναφέρονται συστήματα και τεχνολογίες που παραδοσιακά δεν ταιριάζουν μαζί. Για την επίτευξη της δημιουργίας αυτής της πλατφόρμας έχουν χρησιμοποιηθεί πολλά διαφορετικά συστήματα και τεχνολογίες.

1.1 Περίληψη

Σκοπός της πτυχιακής εργασίας είναι η μελέτη, σχεδίαση και ανάπτυξη μιας ολοκληρωμένης πλατφόρμας ασύγχρονης ψηφιακής τηλεόρασης. Παρατηρώντας την ολοένα αυξανόμενη ανάγκη των χρηστών διαδικτύου να έχουν πρόσβαση σε τηλεοπτικό περιεχόμενο το οποίο είχε προβληθεί σε παλαιότερο χρόνο, από τα παραδοσιακά μέσα μετάδοσης, καθώς και ανεξάρτητα από την γεωγραφική προβολή του, αναπτύχθηκε η συγκεκριμένη ψηφιακή πλατφόρμα.

Μέσω αυτής της πλατφόρμας δίνεται η δυνατότητα στους χρήστες να έχουν πρόσβαση σε τηλεοπτικό περιεχόμενο οποτεδήποτε οι ίδιοι το επιθυμούν. Οι πρόσβαση στο περιεχόμενο μπορεί να γίνει από διάφορες συσκευές και με διάφορους τρόπους. Υπάρχει δυνατότητα ο χρήστης μέσω του γνωστού περιβάλλοντος της εφαρμογής web browser να επιλέξει να παρακολουθήσει το τηλεοπτικό πρόγραμμα το οποίο είναι διαθέσιμο από την πλατφόρμα. Αυτός ο τρόπος χρήσης είναι ιδιαίτερα διαδεδομένος καθώς οι χρήστες είναι πολύ εξοικειωμένη με την χρήση παρόμοιων ιστοσελίδων όπως για παράδειγμα το YouTube και το Vimeo. Οι χρήστες θα μπορούν να δουν το τηλεοπτικό περιεχόμενο μέσω διαδεδομένων προτύπων. Η αναπαραγωγή μπορεί να γίνει είτε με την χρήση του προγράμματος Flash (Adobe) ή μέσω του video tag το οποίο υπάρχει στην HTML5 και υποστηρίζεται από τους καινούργιους web browsers. Η λύση αυτή δεν προσφέρει την μεγαλύτερη εμπειρία χρήσης της πλατφόρμας, αλλά υλοποιήθηκε καθώς είναι ένας διαδεδομένος τρόπος πρόσβασης.

Ένας διαφορετικός τρόπος πρόσβασης στο τηλεοπτικό περιεχόμενο είναι μέσω μιας συσκευής γνωστής ως Set-top-Box (STB). Η χρήση του STB είναι η προτεινόμενη καθώς δίνει την μεγαλύτερη εμπειρία χρήσης της πλατφόρμας. Γι' αυτό το λόγο έχει γίνει ανάπτυξη ειδικού λογισμικού στα πλαίσια της πτυχιακής εργασίας για να μπορεί να γίνει χρήση της πλατφόρμα μέσω μιας συσκευής STB. Η συσκευή STB είναι ένα ενσωματωμένο υπολογιστικό σύστημα με πολύ συγκεκριμένες δυνατότητες. Αν και συνήθως το κόστος τους είναι αρκετά χαμηλό, αυτό δεν περιορίζει της προσφερόμενες υπηρεσίες. Οι συσκευές STB εκτός από τις εξόδους για την παροχή του τηλεοπτικού και ηχητικού σήματος, έχουν συνδέσεις για συσκευές που υποστηρίζουν το πρωτόκολλο USB καθώς και σύνδεση Ethernet ή WIFI για πρόσβαση στο διαδίκτυο,

Η πλατφόρμα ασύγχρονης ψηφιακής τηλεόρασης δίνει τη δυνατότητα στους χρήστες να επιλέξουν από μια ευρεία γκάμα τηλεοπτικού περιεχομένου. Το τηλεοπτικό περιεχόμενο καθορίζεται από τον διαχειριστή του συστήματος, ο οποίος καταχωρεί το διαθέσιμο περιεχόμενο και το κάνει διαθέσιμο στην πλατφόρμα. Η πλατφόρμα υποστηρίζει υψηλής ποιότητας εικόνα και ήχο και

συγκεκριμένα η ανάλυση του βίντεο μπορεί να φτάσει τα 1080p και ο ήχος να είναι πολυκάναλος με χρήση κωδικοποίησης Dolby Digital. Μερικές από της ευκολίες είναι η δυνατότητα ο χρήστης να επιλέξει τα τηλεοπτικά προγράμματα που τον ενδιαφέρουν και αυτόματα η πλατφόρμα να των ενημερώνει για την ύπαρξη νέου προγράμματος.

Βασικό χαρακτηριστικό είναι η υποστήριξη υποτίτλων. Αυτό δίνει την ευκαιρία να γίνει διαθέσιμο ξενόγλωσσο τηλεοπτικό πρόγραμμα σε χρήστες που δεν γνωρίζουν την γλωσσά. Ένα εξίσου μεγάλο όφελος της υποστήριξης υποτίτλων είναι για χρήστες με προβλήματα ακοής. Γνωρίζοντας τους περιορισμούς των ανθρώπων αυτών, η χρήση κατάλληλων υποτίτλων τους δίνει τη δυνατότητα να παρακολουθήσουν τηλεοπτικό πρόγραμμα χωρίς προβλήματα κατανόησης.

1.2 Κίνητρο για την Διεξαγωγή της Εργασίας

Η ανάπτυξη του διαδικτύου και τον προγραμμάτων ανταλλαγής αρχείων γνωστά ως peer-to-peer έδωσαν τη δυνατότητα σε αρκετούς χρήστες να έχουν πρόσβαση σε πολλά τηλεοπτικά προγράμματα, κυρίως της Αμερικής. Λόγο της ιδιαιτερότητας της Αμερικανικής Τηλεοπτικής βιομηχανίας, οι τηλεοπτικές σειρές χαρακτηρίζονται ως μικρές κινηματογραφικές παραγωγές. Χαρακτηριστικό παράδειγμα είναι το πρώτο επεισόδιο της τηλεοπτικής σειράς Lost που στοίχισε περίπου $11.000.000. Η διαφορά με την εγχώρια τηλεοπτική βιομηχανία είναι πολύ μεγάλη όσων άφορα τον προϋπολογισμό. Έτσι λοιπόν, υπάρχει μεγάλη διαφορά στην ποιότητα της παραγωγής του τηλεοπτικού περιεχομένου από τα σκηνικά μέχρι και τα special effects.

Οι μεγάλες τηλεοπτικές παραγωγές της Αμερικής έγιναν γρήγορα πολύ δημοφιλής και σε άλλες χώρες. Αν και πολλά χρόνια συνηθίζετε να γίνεται αγοραπωλησία των δικαιωμάτων μιας τηλεοπτικής παραγωγής και να αναπαραγάγετε σε άλλες χώρες, η διείσδυση του διαδικτύου έδωσε τη δυνατότητα στους χρήστες να έχουν πρόσβαση απευθείας στο τηλεοπτικό προϊόν λίγες ώρες μετά την πρώτη προβολή του.

Το κίνητρο για την διεξαγωγή της πτυχιακής εργασίας είναι να δώσει τη δυνατότητα στους χρήστες να έχουν πρόσβαση σε υψηλής ποιότητας τηλεοπτικό περιεχόμενο από ολόκληρο των κόσμο. Όπως παλιότερα η πειρατεία της μουσικής οδήγησε στην δημιουργία του iTunes με τεράστια κέρδη, το ίδιο πιστεύουμε ότι μπορεί να συμβεί και στην περίπτωση του τηλεοπτικού προγράμματος.

1.3 Σκοπός και Στόχοι Εργασίας

Παρατηρώντας ένα κενό στην διανομή του τηλεοπτικού προϊόντος καθώς και την ανάγκη για καλύτερη χρήση της τεχνολογίας στον τομέα της ψυχαγωγίας, γεννήθηκε η ιδέα της υλοποίησης μιας ενιαίας ψηφιακής πλατφόρμας παροχής τηλεοπτικού προγράμματος που θα άλλαζε τον τρόπο που παραδοσιακά αλληλεπιδρούμε με την τηλεόραση. Η πτυχιακή εργασία αυτή ασχολείται με την δημιουργία ενός λειτουργικού πρωτοτύπου που θα δίνει τη δυνατότητα να δούμε στην πράξη την αποτελεσματικότητα αυτής της ιδέας. Κύριος σκοπός είναι να δώσει τη δυνατότητα στους χρήστες να παρακολουθήσουν οποιαδήποτε στιγμή το επιθυμούν κάποιο τηλεοπτικό πρόγραμμα. Το τηλεοπτικό πρόγραμμα, μπορεί να είχε μεταδοθεί με τους παραδοσιακούς τρόπους πριν από λίγες ώρες μέχρι και ολόκληρα χρόνια. Η ψηφιακή αρχειοθέτηση των τηλεοπτικών παραγωγών μας δίνει τη δυνατότητα να έχουμε πρόσβαση στο υλικό αυτό και μέσω αυτής της πλατφόρμας να το κάνουμε διαθέσιμο στους χρήστες όταν οι ίδιοι το επιθυμούν.

Ένας από τους σημαντικούς στόχους για την δημιουργία της ψηφιακής πλατφόρμας είναι η βιωσιμότητα της. Λαμβάνοντας υπόψιν τον μεγάλο όγκο δεδομένων που πρέπει να μεταδίδεται καθώς και το κόστος του bandwidth από τις εταιρίες παροχής διαδικτύου, πρέπει να γίνει προσεκτική μελέτη για τους τρόπους και της τεχνικές για να κρατηθεί η μεταφορά δεδομένων σε τέτοια πλαίσια που να θεωρείται συμφέρουσα.

Ένας σημαντικός στόχος είναι η δυνατότητα χρήσης της πλατφόρμας με διαφορετικούς τρόπους. Έχει δοθεί ιδιαίτερη προσοχή στην χρήση αρθρωτής αρχιτεκτονικής έτσι ώστε να είναι δυνατή η χρήση της πλατφόρμας από αρκετές διαφορετικές συσκευές, αλλά ακόμα και από εφαρμογή web browser.

- Υψηλή ποιότητα εικόνας και ήχου
- Αυτόματη ενημέρωση για νέο τηλεοπτικό πρόγραμμα (εγγραφή σε τηλεοπτικά προγράμματα)
- Υποστήριξη Υποτίτλων (ξενόγλωσσων & για άτομα με ειδικές ανάγκες)
- Ελκυστική και Εύχρηστη διεπαφή
- Υψηλές ταχύτητες μεταφοράς δεδομένων
- Εξοικονόμηση εύρους ζώνης
- Μικρο κόστος απόκτησης
- Βιωσιμότητα

2 Μέθοδος Ανάλυσης & Ανάπτυξης

Πρώτιστο μέλημα μας είναι η δημιουργία ενός αρθρωτού συστήματος το οποίο θα δίνει μεγάλη ευελιξία στον τρόπο λειτουργίας της πλατφόρμας. Δημιουργώντας υποσυστήματα τα οποία το καθένα είναι ανεξάρτητο το ένα από το άλλο, υπάρχει μεγάλη ευκολία για αλλαγές και προσθήκες νέων χαρακτηριστικών. Η δημιουργία μιας πλατφόρμας μπορεί να χρησιμοποιεί πολλές και διαφορετικές τεχνολογίες. Για το λόγο αυτό ήταν καίριας σημασίας όλα τα υποσυστήματα να λειτουργούν ανεξάρτητα το ένα από το άλλο.

Χρησιμοποιώντας κατάλληλες τεχνολογίες και τεχνικές ανάπτυξης θα μας προστατεύσουν από προβλήματα τα οποία μπορεί να εμφανιστούν όχι απαραίτητα στη φάση της ανάπτυξης αλλά πολύ αργότερα. Ένα παράδειγμα είναι η χρήση του πρωτοκόλλου μεταφοράς δεδομένων BitTorrent το οποίο και περιγράφετε παρακάτω.

Για την σωστή οργάνωση του πηγαίου κώδικα χρησιμοποιούμε το πρόγραμμα Git το οποίο είναι ένα κατανεμημένο εργαλείο έλεγχου εκδόσεων. Μεσώ αυτού μπορούμε να έχουμε πρόσβαση στον πηγαίο κώδικα από το διαδίκτυο, να δούμε προηγούμενες εκδόσεις του πηγαίου κώδικα, δυνατότητα πολλοί προγραμματιστές να συνεισφέρουν πηγαίο κώδικα και άλλα χαρακτηριστικά που κάνουν την ανάπτυξη ασφαλέστερη και πιο οργανωμένη.

2.1 Αλγόριθμοι

2.1.1 BitTorrent

Το BitTorrent είναι ένα πρωτόκολλο μεταφοράς δεδομένων μέσω του διαδικτύου. Προσφέρει μια αξιόπιστη λύση για τη μεταφορά πολύ μεγάλων αρχείων, όπως για παράδειγμα τα αρχεία ήχου ή βίντεο, μετατρέποντας τον υπολογιστή κάθε χρήστη σε σημείο αναδιανομής.

Με βάση αυτό το πρωτόκολλο, ο διανομέας ή κάτοχος του αρχείου, αντί να το διανέμει σε κάθε αιτητή ξεχωριστά, το αποστέλλει σε έναν, ο οποίος με τη σειρά του το αναδιανέμει σε άλλους αιτητές. Οι αιτητές διαμοιράζονται μεταξύ τους κομμάτια του αρχείου, τα οποία είναι μικρότερα σε μέγεθος από το αρχικό αρχείο, μέχρις ότου όλοι τους να ολοκληρώσουν τη λήψη του. Με αυτήν την τεχνική, καθίσταται δυνατή η μεταφορά τεράστιου όγκου δεδομένων μέσω του διαδικτύου, χωρίς να είναι απαραίτητη η χρήση εξυπηρετητών (servers).

Το BitTorrent γράφτηκε από τον προγραμματιστή Μπραμ Κόχεν (Bram Cohen) στην γλώσσα προγραμματισμού Python και έγινε διαθέσιμο το 2001.

Η χρήση του πρωτοκόλλου BitTorrent στην ψηφιακή πλατφόρμα ασύγχρονης τηλεόρασης είναι ένα σημαντικό κομμάτι καθώς χρησιμοποιείτε ως το μέσω για την μεταφορά του τηλεοπτικού περιεχομένου. Μέσω του πρωτοκόλλου BitTorrent, έχουμε μεγάλη απόδοση στην ταυτόχρονη διανομή του βίντεο, και μειώνουμε το κόστος για την συντήρηση του κεντρικού εξυπηρετητή.

2.1.2 Κατανεμημένο σύστημα ελέγχου εκδόσεων

Στον προγραμματισμό λογισμικού το κατανεμημένο σύστημα ελέγχου εκδόσεων (Distributed revision control) είναι λογισμικό οποίο παρακολουθεί την πορεία των αναθεωρήσεων λογισμικού και επιτρέπει σε πολλούς προγραμματιστές να εργάζονται ταυτόχρονα σε ένα κοινό πηγαίο κώδικα χωρίς να είναι απαραίτητο να είναι συνδεδεμένοι σε ένα κοινό δίκτυο.

Το Git δημιουργήθηκε από τον Linus Torvalds, προγραμματιστή του Linux, για να καλύψει της ανάγκες του πυρήνα. Πλέον χρησιμοποιείται από πολύ γνωστά και μεγάλα projects όπως το

Android, Facebook, Gnome, Twitter, Qt, Netflix και πολλά άλλα.

Στην διάρκεια της ανάπτυξης της ψηφιακής πλατφόρμας έγινε η χρήση του Git μέσω της πλατφόρμας Bitbucket. Έτσι είχαμε όλες της δυνατότητες που μας δίνει η χρήση του Git καθώς και ένα εύχρηστο περιβάλλον που θα μπορούσαμε να δούμε αλλαγές και να διαχειριστούμε το project μέσω της ιστοσελίδας bitbucket.org.

2.1.3 HTTP Live Streaming

Το HTTP Live Streaming (γνωστό επίσης ως HLS) είναι ένα πρωτόκολλο βασισμένο στο HTTP για media streaming. Λειτουργεί με το να χωρίζει την ροή δεδομένων σε μικρότερα κομμάτια τα οποία στέλνονται μέσω του πρωτοκόλλου HTTP. Κάθε πακέτο HTTP έχει ένα μικρο μέρος από την ροή, το λογισμικό πελάτη μπορεί να επιλέξει από πολλές διαφορετικές ροές ανάλογα με το διαθέσιμο εύρος ζώνης. Αυτό δίνει τη δυνατότητα να παρακολουθούμε βίντεο υψηλής ποιότητας, αλλά στην περίπτωση που η ταχύτητα του δικτύου μειωθεί, να αλλάξουμε αυτόματα σε χαμηλότερης ποιότητας βίντεο, χωρίς να υπάρχει διακοπή στην αναπαραγωγή.

Λόγο του ότι η μετάδοση των δεδομένων γίνετε μέσω του πρωτοκόλλου HTTP, το HTTP Live Streaming έχει τη δυνατότητα να λειτουργήσει πίσω από firewall ή proxy servers, σε αντίθεση με τα πρωτόκολλα που βασίζονται στο UDP όπως για παράδειγμα το RTP. Επίσης μπορεί να γίνει η χρήση content delivery network(CDN) για την αύξηση της απόδοσης μεταφοράς δεδομένων.

2.2 Μοντέλα

2.2.1 Event-driven Programming

Ο προγραμματισμός με βάση τα συμβάντα (Event-driven programming) είναι ένα προγραμματιστικό παράδειγμα όπου η ροή της εκτέλεσης του προγράμματος καθορίζεται από κάποια συμβάντα (σήμα από sensor, κλικ του ποντικιού, μήνυμα από άλλο πρόγραμμα ή thread).

Ο προγραμματισμός με βάση τα events μπορεί να ορίζει και ολόκληρη την αρχιτεκτονική του λογισμικού. Σε αυτήν την περίπτωση το πρόγραμμα έχει μια main loop (ή event-loop) και χωρίζετε σε δυο διακριτές καταστάσεις. Η πρώτη είναι η ανίχνευση του event (detection) και η δεύτερη είναι ο χειρισμός του event (handling).

Στα ενσωματωμένα συστήματα, το ίδιο μπορεί να επιτευχθεί με χρήση interrupts αντί για την συνεχή εκτέλεση της main loop. Στην περίπτωση αυτή, η ανίχνευση (detection) του event είναι ευθύνη του υλικού.

Event-driven προγράμματα μπορούν να γραφτούν με χρήση οποιασδήποτε γλώσσας προγραμματισμού. Η χρήση μιας πιο high-level abstracted γλώσσας που προσφέρει closures αυξάνει την ευκολία γραφής του κώδικα με το συγκεκριμένη τακτική.

2.2.2 Asynchronous IO

Asynchronous IO ή non-blocking IO είναι μια μορφή επεξεργασίας της εισόδου / εξόδου που επιτρέπει σε άλλες διεργασίες να συνεχίσουν την επεξεργασία πριν να τελειώσει η μετάδοση.

Οι διεργασίες εισόδου / εξόδου σε έναν υπολογιστή μπορεί να είναι μια εξαιρετικά αργή διαδικασία σε σχέση με την επεξεργασία των δεδομένων. Μια συσκευή IO μπορεί να ενσωματώνει μηχανικά μέρη που θα πρέπει να μετακινηθούν για να ολοκληρωθεί μια λειτουργία, αυτό αποτελεί πολύ μεγάλη καθυστέρηση σε σχέση με την λειτουργία της μεταγωγής ηλεκτρικού ρεύματος σε κάποια συσκευή που χρησιμοποιεί μόνο ηλεκτρονικά κυκλώματα για την λειτουργία της.

Μια απλή προσέγγιση στο IO είναι να ξεκινήσει η πρόσβαση και στη συνέχεια να περιμένουμε μέχρι να ολοκληρωθεί. Μια τέτοια προσέγγιση (γνωστή ως synchronous IO ή blocking IO) θα εμποδίσει την εκτέλεση του προγράμματος μειώνοντας έτσι δραματικά την απόδοση και αφήνοντας τους πόρους του συστήματος ανεκμετάλλευτους. Όταν ένα πρόγραμμα κάνει πολλές IO διεργασίες, αυτό αυτόματα σημαίνει ότι τον περισσότερο χρόνο η διεργασία θα περιμένει να ολοκληρωθεί το IO πριν συνεχίσει.

Χρησιμοποιώντας την μεθοδολογία ανάπτυξης με ασύγχρονο IO, η εκτέλεση του προγράμματος συνεχίζετε μόνο για ενέργειες που δεν έχουν άμεση σχέση με τα την ενέργεια του IO. Αυτό έχει ως αποτέλεσμα την αύξηση της απόδοσης και την αποδοτικότερη χρήση των πόρων του συστήματος. Χρειάζεται ιδιαίτερη προσοχή η συγκεκριμένη μεθοδολογία διότι αυξάνει την πολυπλοκότητα του λογισμικού και θα πρέπει να διασφαλιστεί ότι η ενέργειες που έχουν σχέση με το IO θα εκτελεστούν μόνο μετά την ολοκλήρωση του. Τα οφέλη χρήσης της συγκεκριμένης προσέγγισης είναι η καλύτερη αξιοποίηση των πόρων του συστήματος και σε πολλές περιπτώσεις η αύξηση της απόδοσης του λογισμικού.

3 Τεχνολογία Αιχμής (State of the Art)

Το κεφάλαιο αυτό ασχολείται με την βιβλιογραφική αναζήτηση της τεχνολογίας αιχμής για την εκπόνηση της πτυχιακής εργασίας. Οτιδήποτε έχει χρησιμοποιηθεί για την ψηφιακή πλατφόρμα είναι ανοικτό λογισμικό. Οι λόγοι που επιλέξαμε ανοικτό λογισμικό για την δημιουργία της ψηφιακής πλατφόρμας είναι η ποιότητα, το κόστος και η ελεύθερη πρόσβαση στον πηγαίο κώδικα.

3.1 Clutter

Εικόνα 1: Clutter Toolkit

Το Clutter είναι μια βιβλιοθήκη ανοιχτού κώδικα για την δημιουργία hardware-accelerated γραφικού περιβάλλοντος. Χρησιμοποιεί το OpenGL για rendering ή OpenGL ES σε mobile και embedded συσκευές, υποστηρίζει όλα τα δημοφιλή λειτουργικά συστήματα και δίνει ένα απλό και ευέλικτο API κρύβοντας την πολυπλοκότητα του OpenGL. Μπορεί να χρησιμοποιηθεί από αρκετές γλώσσες όπως η Perl, Python, Vala κτλ. Επιπρόσθετα χαρακτηριστικά είναι ότι μπορεί να αναπαραγάγει μέσω GStreamer αρχεία πολυμέσων, καθώς και να σχεδιάσει 2D γραφικά μέσω της βιβλιοθήκης Cairo.

Αρχιτεκτονική

Το Clutter βασίζετε στη λογική ότι κάθε αντικείμενο είναι μια 2D επιφάνεια σε 3D χώρο, τα αντικείμενα αυτά ονομάζονται Actors. Κάθε αντικείμενο Actor τοποθετείτε σε μια κεντρική σκηνή. Σε αντίθεση με το OpenGL που χρησιμοποιεί matrices για να υπολογίσει την κατάσταση του 3D αντικειμένου, το Clutter χρησιμοποιεί τα properties των Actors και υπολογίζει μόνο του πότε και πως θα κάνει render το αντικείμενο στην σκηνή.

Animation

Σε κάθε Actor πάνω στην σκηνή μπορεί να εφαρμοστεί ένα ειδικό αντικείμενο τύπου behaviour το οποίο αλλάζει τα χαρακτηριστικά του Actor σε ένα διάστημα χρόνου, δημιουργώντας κίνηση (animation) για το αντικείμενο. Κάθε αντικείμενο behaviour μπορεί να εφαρμοστεί σε πολλά διαφορετικά αντικείμενα Actors και πολλά διαφορετικά behaviour μπορούν να εφαρμοστούν σε ένα Actor. Το αντικείμενο behaviour δέχεται τιμές για την αρχική και τελική κατάσταση του αντικειμένου Actor, τον χρόνο ή τα καρέ για να ολοκληρωθεί το animation και την συνάρτηση χρόνου που θα χρησιμοποιηθεί (linear, sine wave, exponential κτλ). Το λεγόμενο tweening, δηλαδή η κίνηση στις ενδιάμεσες θέσεις από την αρχική μέχρι την τελική κατάσταση του Actor την αναλαμβάνει το αντικείμενο behaviour.

Το Clutter δίνει τη δυνατότητα στον προγραμματιστή να επεκτείνει το αντικείμενο behaviour δίνοντας του τη δυνατότητα για μεγαλύτερη ευελιξία στο animation το οποίο θα εφαρμοστεί στο αντικείμενο Actor. Οι αλλαγές που μπορεί να υποστεί ένα αντικείμενο Actor στην διαδικασία του animation μπορεί να είναι από απλές αλλαγές στην διαφάνεια(opacity), την τοποθεσία (X, Y axis), το βάθος (Z axis), την περιστροφή, μέχρι μεγαλύτερης πολυπλοκότητας αλλαγές που μπορεί να επεμβαίνουν άμεσα στην διαδικασία του rendering σε OpenGL. Για πολύ απλές αλλαγές στα αντικείμενα μπορεί να χρησιμοποιηθεί το Animation API.

Δημιουργία διεπαφής

Για την δημιουργία της διεπαφής εκτός από την κλασσική προσέγγιση μέσω προγραμματισμού του κάθε αντικειμένου, μπορεί να χρησιμοποιηθεί η διάλεκτο JSON για τον ορισμό των αντικειμένων της σκηνής. Κατά το runtime το Clutter διαβάζει τον ορισμό της διεπαφής και το δημιουργεί δυναμικά. Έτσι ο προγραμματιστής μπορεί να κάνει αλλαγές στην διεπαφή με μεγαλύτερη ευκολία και χωρίς μεγάλες αλλαγές στον πηγαίο κώδικα.

Χαρακτηριστικά

Μερικά από τα χαρακτηριστικά του Clutter είναι:

- Βασισμένο στην λογική του Scene-graph με 2D αντικείμενα σε 3D χώρο.
- Animation framework
- Ορισμός διεπαφής σε JSON
- Input event handling με υποστήριξη ταυτόχρονων pointing συσκευών
- Υποστήριξη UTF-8 κειμένου μέσω της βιβλιοθήκης Pango
- Υποστήριξη προχωρημένων OpenGL δυνατοτήτων όπως Shaders, FBO, VBO και PBO μέσω ενός low-level αντικειμενοστραφές abstracted API.
- Υποστήριξη αναπαραγωγής βίντεο μέσω του συστήματος Gstreamer.
- Αντικειμενοστραφής σχεδιασμός μέσω της βιβλιοθήκης Gobject
- Υποστήριξη σε Linux, Windows, OSX με backend σε GLX, EGL, WGL και Cocoa.
- Υποστήριξη mobile συσκευών με fixed point internals και OpenGL ES 1.1 ή 2.0
- Γραμμένο στην γλώσσα προγραμματισμού C με language bindings για Perl, Python, C#, C++, Vala και Ruby. Υποστήριξη του GObject introspection API.

Χρήση του Clutter

Από την έκδοση 3.0 και έπειτα του Desktop Enviroment GNOME γίνεται η χρήση του Clutter ως κύριου χαρακτηριστικού του συστήματος. Το Mutter – το οποίο κάνει χρήση του Clutter – αναλαμβάνει να τοποθετήσει κάθε παράθυρο στην επιφάνεια του χρήστη. Όλο το γραφικό περιβάλλον πλέον υλοποιείται μέσω OpenGL. Αυτό έχει ως αποτέλεσμα την αύξηση των επιδόσεων καθώς μεγάλο μέρος του υπολογιστικού φόρτου γίνεται στην κάρτα γραφικών.

Εικόνα 2: GNOME 3.0

Το Clutter χρησιμοποιείτε επίσης στο λειτουργικό σύστημα Google's Chrome OS. Το Chrome OS είναι ένα λειτουργικό σύστημα που προορίζεται για μικρούς φορητούς ηλεκτρονικούς υπολογιστές, αποθηκεύοντας κατά κύριο λόγο τα δεδομένα του συστήματος στο διαδίκτυο (Cloud computing).

Η ευρεία διάδοση και χρήση του Clutter το έχει καταστήσει ως ένα σταθερό περιβάλλον ανάπτυξης για γραφικά περιβάλλοντα. Η ανάπτυξη του γίνεται από την εταιρία Intel και από άλλους προγραμματιστές.

Παραδείγματα

Προσθήκη ετικέτας στο κέντρο της σκηνής

```
ClutterActor *stage = clutter_stage_get_default ();
ClutterActor *label = clutter_text_new_with_text ("Sans 32px",
"Hello, world");
clutter_container_add_actor (CLUTTER_CONTAINER (stage), label);

float x, y;

x = (clutter_actor_get_width (stage) - clutter_actor_get_width
(label)) / 2;
y = (clutter_actor_get_height (stage) - clutter_actor_get_height
(label)) / 2;
clutter_actor_set_position (label, x, y);
```

Διπλασιασμός του μεγέθους της ετικέτας σε χρόνο 2 δευτερολέπτων με χρήση γραμμικής συνάρτησης

```
ClutterTimeline *timeline = clutter_timeline_new (2000);
ClutterAlpha    *alpha    =    clutter_alpha_new_full    (timeline,
CLUTTER_LINEAR);
ClutterBehaviour *behaviour = clutter_behaviour_scale_new (alpha,
                                                           1.0,
1.0, /* initial scaling factors */
                                                           2.0,
2.0  /* final scaling factors */ );
clutter_behaviour_apply (behaviour, label);
```

Εφάμιλλο αποτέλεσμα με χρήση του Animation API

```
clutter_actor_animate (label,          /* the actor to animate */
                       CLUTTER_LINEAR, /* the easing mode */
                       2000,           /* duration  */
                       "scale-x", 2.0, /
                       "scale-y", 2.0,
                       NULL);
```

Ορισμός της διεπαφής σε JSON

```
{
  "id" : "main-stage",
  "type" : "ClutterStage",
  "color" : "white",
```

```
  "width" : 800,
  "height" : 600,
  "title" : "Script demo",
  "children" : [
    {
      "id" : "hello-label",
      "type" : "ClutterText",
      "x" : 400,
      "y" : 300,
      "text" : "Hello, world!",
      "color" : "black",
      "font-name" : "Sans 48px"
    }
  ],
  "signals" : [
    { "name" : "destroy", "handler" : "clutter_main_quit" }
  ]
}
```

Δημιουργία της διεπαφής μέσω του παραπάνω αρχείου JSON

```
ClutterScript *script = clutter_script_new ();
GError *error = NULL;
clutter_script_load_from_data (script, description, -1, &error);
if (error)
  {
      g_warning ("Unable to load UI description: %s", error-
>message);
    g_error_free (error);
  }else{
    GObject *stage;

    clutter_script_connect_signals (script, NULL);
    stage = clutter_script_get_object (script, "main-stage");
    clutter_actor_show (CLUTTER_ACTOR (stage));
  }
```

3.2 JavaScript

Η JavaScript είναι γλώσσα προγραμματισμού η οποία έχει σαν σκοπό την παραγωγή δυναμικού περιεχομένου και την εκτέλεση κώδικα στην πλευρά του πελάτη (client-side) σε ιστοσελίδες. Το πρότυπο της γλώσσας κατά τον οργανισμό τυποποίησης ECMA ονομάζεται ECMAscript. Η αρχική έκδοση της JavaScript βασίστηκε στη σύνταξη στη γλώσσα προγραμματισμού C, αν και έχει εξελιχθεί, ενσωματώνοντας πια χαρακτηριστικά από νεότερες γλώσσες.

Παρά την ευρεία χρήση της Javascript για συγγραφή προγραμμάτων σε περιβάλλον φυλλομετρητή, χρησιμοποιήθηκε και για τη συγγραφή κώδικα από την πλευρά του διακομιστή, στο προϊόν LiveWire της Netscape, με μικρή επιτυχία. Η χρήση της Javascript στο διακομιστή εμφανίζεται πάλι σήμερα, με τη διάδοση του Node.js, ενός μοντέλου προγραμματισμού βασισμένο στα γεγονότα (events).

Η Javascript δεν θα πρέπει να συγχέεται με τη Java, που είναι διαφορετική γλώσσα προγραμματισμού και με διαφορετικές εφαρμογές. Η χρήση της λέξης "Java" στο όνομα της γλώσσας έχει περισσότερη σχέση με το προφίλ του προϊόντος που έπρεπε να έχει και λιγότερο με κάποια πιθανή συμβατότητα ή άλλη στενή σχέση με τη Java. Ρόλο σε αυτήν τη σύγχυση έπαιξε και ότι η Java και η Javascript έχουν δεχτεί σημαντικές επιρροές από τη γλώσσα C, ειδικά στο συντακτικό, ενώ είναι και οι δύο αντικειμενοστρεφείς γλώσσες. Τονίζεται ότι ο σωστός τρόπος γραφής της είναι "JavaScript" και όχι 'Java script' σαν δύο λέξεις, όπως λανθασμένα γράφεται ορισμένες φορές.

Μοντέλο εκτέλεσης

Η αρχική έκδοση της Javascript βασίστηκε στη σύνταξη στη γλώσσα προγραμματισμού C, αν και έχει εξελιχθεί, ενσωματώνοντας πια χαρακτηριστικά από νεότερες γλώσσες.

Αρχικά χρησιμοποιήθηκε για προγραμματισμό από την πλευρά του πελάτη (client), που ήταν ο φυλλομετρητής (browser) του χρήστη, και χαρακτηρίστηκε σαν client-side γλώσσα προγραμματισμού. Αυτό σημαίνει ότι η επεξεργασία του κώδικα Javascript και η παραγωγή του τελικού περιεχομένου HTML δεν πραγματοποιείται στο διακομιστή, αλλά στο πρόγραμμα περιήγησης των επισκεπτών, ενώ μπορεί να ενσωματωθεί σε στατικές σελίδες HTML. Αντίθετα, άλλες γλώσσες όπως η PHP εκτελούνται στο διακομιστή (server-side γλώσσες προγραμματισμού).

Παρά την ευρεία χρήση της Javascript για συγγραφή προγραμμάτων σε περιβάλλον φυλλομετρητή, αξίζει να σημειωθεί ότι από την αρχή χρησιμοποιήθηκε και για τη συγγραφή κώδικα από την πλευρά του διακομιστή, από την ίδια τη Netscape στο προϊόν LiveWire, με μικρή επιτυχία. Η χρήση της Javascript στο διακομιστή εμφανίζεται πάλι σήμερα, με τη διάδοση του Node.js, ενός μοντέλου προγραμματισμού βασισμένο στα γεγονότα (events).

Παραδείγματα

Μια απλή Recursive συνάρτηση

```
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Σύνταξη μιας Anonymous συνάρτησης και υλοποίηση closure

```
function displayClosure() {
    var count = 0;
    return function () {
        return ++count;
    };
}
var inc = displayClosure();
inc(); // returns 1
inc(); // returns 2
inc(); // returns 3
```

Επίδειξη μιας Variadic συνάρτησης

```
function sum() {
    var i, x = 0;
    for (i = 0; i < arguments.length; ++i) {
        x += arguments[i];
    }
    return x;
}
sum(1, 2, 3); // returns 6
```

3.3 Node.js

Εικόνα 3: Node.js

Το Node.js είναι μια πλατφόρμα για την κατασκευή scalable δικτυακών προγραμμάτων, κατά κύριο λόγο web servers. Τα προγράμματα είναι γραμμένα στην γλώσσα προγραμματισμού JavaScript, χρησιμοποιώντας event-driven αρχιτεκτονική και ασύγχρονο I/O ελαχιστοποιώντας το overhead και αυξάνοντας την επεκτασιμότητα. Το Node.js αποτελείτε από την μηχανή εκτέλεσης JavaScript V8 της Google καθώς και αρκετές built-in βιβλιοθήκες που επεκτείνουν το πρότυπο της JavaScript.

Δημιουργήθηκε το 2009 από τον Ryan Dahl και υποστηρίζετε από την εταιρία Joyent. Αν και είναι μια καινούργια πλατφόρμα, ήδη χρησιμοποιείτε από μεγάλες εταιρίες όπως eBay, Yahoo και Linkedin υποστηρίζοντας κρίσιμα σημεία των συστημάτων τους.

Παρόμοια περιβάλλοντα χρησιμοποιώντας άλλες γλώσσες προγραμματισμού είναι το Twisted για την Python, libevent για την C και EventMachine για την Ruby.

Σε αντίθεση με τα περισσότερα προγράμματα γραμμένα σε JavaScript, δεν εκτελείτε στον web browser, αλλά η εκτέλεση του προγράμματος γίνεται server-side. Το Node.js υλοποιεί κάποιες από τις προδιαγραφές του CommonJS. Διαθέτει ένα REPL(read–eval–print loop) περιβάλλον για διαδραστικές δοκιμές.

Παραδείγματα

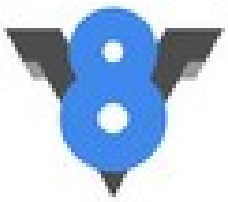
Μια πλήρη εφαρμογή hello world ως HTTP server σε JavaScript τρέχοντας σε Node.js

```
var http = require('http');

http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World\n');
}).listen(8000);

console.log('Server running at http://localhost:8000/');
```

Ο κώδικας που ακολουθεί είναι ένας απλός διακομιστής TCP που ακούει στη θύρα 7000 και απαντάει «hello» κατά τη σύνδεση:

```
var net = require('net');

net.createServer(function (stream) {
    stream.write('hello\r\n');

    stream.on('end', function () {
        stream.end('goodbye\r\n');
    });

    stream.pipe(stream);
}).listen(7000);
```

3.4 V8



Εικόνα 4: V8

Η V8 είναι μια ανοικτού κώδικα μηχανή εκτέλεσης εντολών JavaScript αναπτυγμένη από την εταιρία Google και χρησιμοποιείται από των γνωστό web browser Google Chrome. Η V8 αυξάνει την απόδοση εκτέλεσης JavaScript με το να μεταγλωττίζει την γλώσσα JavaScript σε γλώσσα μηχανής πριν την εκτέλεση σε αντιδιαστολή με την εκτέλεση bytecode ή μέσω interpreter. Για ακόμα μεγαλύτερη αύξηση των επιδόσεων χρησιμοποιείτε η τεχνική του inline caching. Η V8 χρησιμοποιεί garbage collector για την αυτόματη διαχείριση μνήμης και συγκεκριμένα τύπου generational incremental collector.

Η V8 μηχανή JavaScript βρίσκεται στην καρδιά του Node.js και είναι ο κύριος λόγος για τα χαρακτηριστικά του Node.js.

Η δημιουργία της V8 JavaScript μηχανής έγινε με κύριο σκοπό την γρήγορη εκτέλεση εφαρμογών σε JavaScript. Σε πολλά benchmarks tests, η V8 είναι κατά πολύ ταχύτερη από άλλες υλοποιήσεις όπως η JScript του Internet Explorer, SpiderMonkey του Firefox, JavaScriptCore του Safari. Η χρήση της V8 μπορεί να αυξήσει δραματικά την εκτέλεση των εφαρμογών JavaScript. Η αύξηση των επιδόσεων σχετίζεται από την φύση της εφαρμογής. Για παράδειγμα εάν μια εφαρμογή εκτελεί πολλές φόρες συγκεκριμένες συναρτήσεις συνεχώς, η αύξηση των επιδόσεων θα είναι πολύ μεγαλύτερες από την περίπτωση της εφαρμογής που εκτελεί πολλές διαφορετικές συναρτήσεις. Ο λόγος που συμβαίνει αυτό εξηγείτε παρακάτω.

Υπάρχουν τρεις συνιστώσες αναφορικά με της επιδόσεις της V8.

- Fast Property Access
- Dynamic Machine Code Generation
- Efficient Garbage Collection

Fast Property Access

Η JavaScript είναι μια δυναμική γλώσσα, γνωρίσματα (properties) μπορούν να προστεθούν ή να αφαιρεθούν την ώρα εκτέλεσης (on-the-fly). Αυτό σημαίνει ότι τα γνωρίσματα του αντικειμένου είναι πιθανό να αλλάξουν. Οι περισσότερες μηχανές JavaScript χρησιμοποιούν dictionary-like data structure για την αποθήκευση των γνωρισμάτων των αντικειμένων. Κάθε πρόσβαση στα γνωρίσματα ενός αντικειμένου χρειάζεται να γίνει μια δυναμική αναζήτηση για να βρεθεί η η τοποθεσία που είναι αποθηκευμένη στην μνήμη. Αυτή η προσέγγιση κάνει την πρόσβαση στα γνωρίσματα των αντικειμένων ιδιαίτερα χρονοβόρα σε σχέση με άλλες γλώσσες όπως η Java και η Smalltalk. Σε αυτές

της γλώσσες τα γνωρίσματα του κάθε αντικειμένου βρίσκονται σε σταθερό offset το οποίο βρίσκεται από τον compiler λόγο της σταθερής δήλωσης του object class. Η πρόσβαση είναι θέμα μόνο ανάγνωσης και έγγραφης στην μνήμη, που στης περισσότερες περιπτώσεις είναι μόνο μια εντολή.

Για να μείωση του χρόνου που χρειάζεται για να γίνει πρόσβαση στα γνωρίσματα των αντικειμένων, η V8 δεν χρησιμοποιεί δυναμική αναζήτηση για να βρει τη διεύθυνση της μνήμης που βρίσκεται το γνώρισμα. Αντιθέτως η V8 δημιουργεί δυναμικά hidden classes (κρυφές κλάσης). Η βασική ιδέα δεν είναι καινούργια και χρησιμοποιείται και από την γλώσσα Self. Στην V8, ένα αντικείμενο αλλάζει hidden class όταν ένα καινούργιο γνώρισμα προστεθεί στο αντικείμενο.

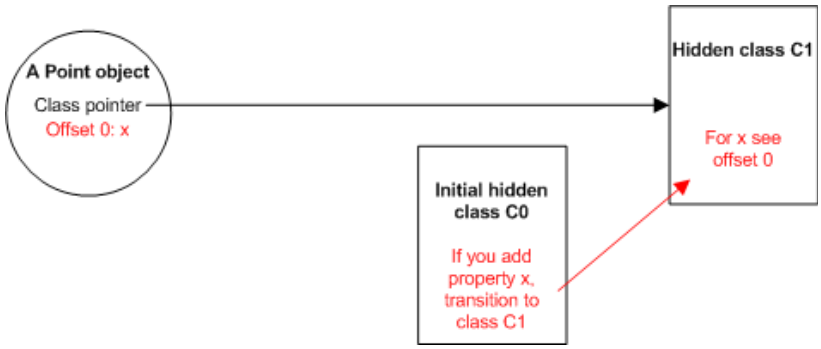
Για να γίνει πιο αντιληπτό, θα χρησιμοποιήσουμε το παρακάτω παράδειγμα. Ας φανταστούμε μια απλή συνάρτηση σε JavaScript όπως αυτή:

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
```

Όταν η *new Point(x, y)* εκτελεστεί, ένα νέο *Point* αντικείμενο θα δημιουργηθεί. Όταν η V8 το κάνει αυτό για την πρώτη φορά, δημιουργεί μια αρχική κρυμμένη κλάση του Point, θα την ονομάσουμε C0 στο παράδειγμα μας. Καθώς το αντικείμενο μας δεν έχει κάποια γνωρίσματα (properties) ακόμα η αρχική κλάση είναι άδεια. Σε αυτό το σημείο η κρυφή κλάση του αντικειμένου Point είναι η C0.

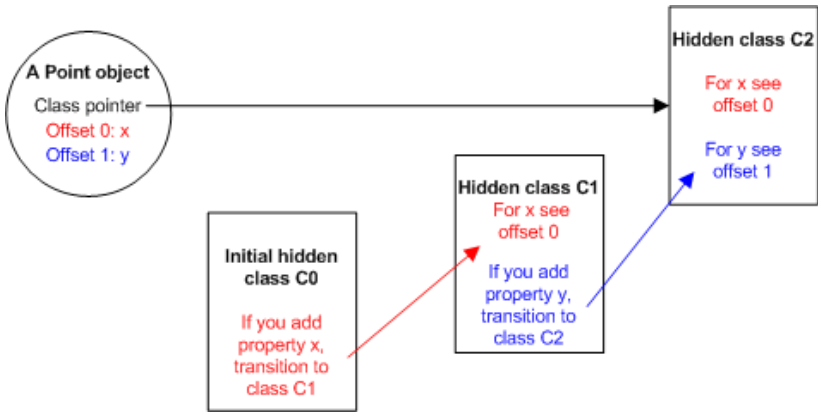
Εκτελώντας την πρώτη δήλωση του Point (this.x = x;) δημιουργείται ένα καινούργιο γνώρισμα, με όνομα x, στο αντικείμενο Point. Σε αυτήν την περίπτωση, η V8:

- δημιουργεί μια καινούργια κρυφή κλάση C1, βασισμένη στην C0, έπειτα προσθέτει πληροφορίες στην C1 που περιγράφουν ότι το αντικείμενο έχει ένα γνώρισμα, με όνομα x, που το περιεχόμενο του βρίσκεται στο offset 0 (μηδέν) στο αντικείμενο Point
- ανανεώνει την C0 με μια μεταβατική περιγραφή, δηλώνοντας ότι εάν ένα γνώρισμα x προστεθεί σε ένα αντικείμενο που περιγράφετε από την C0, τότε η κρυφή κλάση C1 θα πρέπει να χρησιμοποιηθεί αντί για την C0. Σε αυτήν την φάση το αντικείμενο Point αντιστοιχεί στην κρυφή κλάση C1.



Εκτελώντας την δεύτερη δήλωση του Point (this.y = y;) δημιουργείται ένα καινούργιο γνώρισμα, με όνομα y, στο αντικείμενο Point. Σε αυτήν την περίπτωση, η V8:

- δημιουργεί ακόμα μια καινούργια κρυφή κλάση C2, βασισμένη στην C1, έπειτα προσθέτει πληροφορίες στην C2 που περιγράφουν ότι το αντικείμενο έχει ακόμα ένα γνώρισμα, με όνομα y, που το περιεχόμενο του βρίσκεται στο offset 1 (ένα) στο αντικείμενο Point
- ανανεώνει την C1 με μια μεταβατική περιγραφή, δηλώνοντας ότι εάν ένα γνώρισμα y προστεθεί σε ένα αντικείμενο που περιγράφετε από την C1, τότε η κρυφή κλάση C2 θα πρέπει να χρησιμοποιηθεί αντί για την C1. Σε αυτήν την φάση το αντικείμενο Point αντιστοιχεί στην κρυφή κλάση C2.



Ίσως να φαίνεται αναποτελεσματικό να δημιουργείται μια κρυφή κλάση με κάθε γνώρισμα που προστίθετο στο αντικείμενο. Ωστόσο, επειδή η κρυφές κλάσης μπορούν να ξαναχρησιμοποιηθούν, την επόμενη φορά που θα χρειαστεί να δημιουργηθεί ένα αντικείμενο τύπου Point, δεν δημιουργούνται καινούργιες κρυφές κλάσης, αλλά χρησιμοποιούνται αυτές που είχαν δημιουργηθεί πρωτύτερα. Για παράδειγμα, όταν ακόμα ένα αντικείμενο Point δημιουργείται:

- αρχικά το αντικείμενο Point είναι άδειο, οπότε αντιστοιχεί στην κρυφή κλαση C0
- έπειτα προστίθετο το γνώρισμα x, η V8 ακολουθεί την μεταβατική περιγραφή από την C0 προς την C1 και γράφει το περιεχόμενο του x στο offset που ορίζεται από την C1.
- έπειτα προστίθετο το γνώρισμα y, η V8 ακολουθεί την μεταβατική περιγραφή από την C1 προς την C2 και γράφει το περιεχόμενο του y στο offset που ορίζεται από την C2.

Ακόμα κι αν η JavaScript είναι πιο δυναμική από ότι άλλες αντικειμενοστραφής γλώσσες, η συμπεριφορά κατά την εκτέλεση των προγραμμάτων με την συγκεκριμένη τεχνική έχει ως αποτέλεσμα να μοιράζονται πολλές δομές δεδομένων μέσω των κρυφών κλάσεων. Υπάρχουν δυο πλεονεκτήματα χρησιμοποιώντας κρυφές κλάσης: η πρόσβαση στα γνωρίσματα των αντικειμένων δεν χρειάζεται αναζήτηση σε κάποιο dictionary (offset) και δίνουν τη δυνατότητα στην V8 να χρησιμοποιήσει κλασσικές τεχνικές βελτιστοποιήσεις όπως το inline caching.

Dynamic Machine Code Generation

Η V8 compiles των πηγαίο κώδικα JavaScript απευθείας σε γλώσσα μηχανής κατά την πρώτη εκτέλεση. Δεν υπάρχει ενδιάμεσο byte code ή interpreter. Η πρόσβαση στα γνωρίσματα των αντικειμένων γίνεται μέσω inline cache κώδικα και μπορεί να προστεθεί και άλλος κώδικας γλώσσας μηχανής καθώς η V8 εκτελεί τις εντολές.

Κατά την αρχική εκτέλεση του κώδικα για την πρόσβαση στα γνωρίσματα των αντικειμένων, η V8 καθορίζει την hidden class, βελτιστοποιεί την πρόσβαση στα γνωρίσματα με να τα προβλέπει το ποια κλάση μπορεί να χρησιμοποιηθεί στο μέλλον στο ίδιο σημείο του κώδικα και χρησιμοποιεί αυτές της πληροφορίες για να προσθέσει inline cache κώδικα για την χρήση κάποιων συγκεκριμένων κρυφών κλάσεων. Εάν η V8 έχει προβλέψει ορθά, η πρόσβαση στο περιεχομένου του γνωρίσματος θα γίνει με μια εντολή. Εάν η πρόβλεψη είναι λανθασμένη, η V8 αφαιρεί την inline cache βελτιστοποίηση.

Για παράδειγμα, στην JavaScript ο κώδικας για πρόσβαση στο γνώρισμα x του αντικειμένου Point είναι:

```
point.x
```

Στην V8, η γλώσσα μηχανής που δημιουργείτε για την πρόσβαση στο γνώρισμα x είναι;

```
# ebx = the point object
cmp [ebx,<hidden class offset>],<cached hidden class>
jne <inline cache miss>
mov eax,[ebx, <cached x offset>]
```

Εάν η κρυφή κλάση του αντικειμένου δεν ταιριάζει με την cached κρυφή κλάση, η V8 εκτελεί το υποσύστημα που διαχειρίζεται τις αστοχίες της inline cache και αλλάζει την inline cache σύμφωνα με τα νέα δεδομένα. Σε περίπτωση όμως που η κρυφή κλάση ταιριάζει με την cached κρυφή κλάση, που είναι και η πιο συνήθης περίπτωση, τότε το περιεχόμενο του γνωρίσματος x απλά ανακτάτε.

Όταν υπάρχουν πολλά αντικείμενα με την ίδιο κρυμμένη κλάση, τα ίδια οφέλη απορρέουν όπως και στις περισσότερες στατικές γλώσσες. Ο συνδυασμός της χρήσης κρυφών κλάσεων για την πρόσβαση στα γνωρίσματα των αντικειμένων με inline caching και δημιουργία κώδικα μηχανής βελτιστοποιεί τις περιπτώσεις όπου το ίδιο είδος των αντικειμένων συχνά δημιουργείται η προσπελάζεται. Αυτό βελτιώνει σημαντικά την ταχύτητα με την οποία τα περισσότεροι προγράμματα JavaScript εκτελούνται.

Garbage Collection

Η V8 διεκδικεί μνήμη από τα αντικείμενα που δεν χρειάζονται πλέον σε μια διαδικασία γνωστή ως συλλογή των απορριμμάτων (garbage collection). Για να εξασφαλιστεί η γρήγορη δημιουργία των αντικειμένων, μικρές παύσεις του garbage collector και τον μη κατακερματισμό της

μνήμης, η V8 χρησιμοποιεί μια stop-the-world προσέγγιση. Αυτό σημαίνει ότι V8:

- σταμάτα την εκτέλεση του προγράμματος και εκτελεί έναν κύκλο garbage collection
- γνωρίζει πάντοτε της διευθύνσεις μνήμης κάθε αντικειμένου. Αυτό αποτρέπει την λανθασμένη αναγνώριση αντικειμένων ως pointers κάτι που θα οδηγούσε σε memory leaks.

Στην V8, το heap των αντικειμένων χωρίζεται σε δυο μέρη: μια περιοχή όπου αποθηκεύονται τα καινούργια αντικείμενα, και μια περιοχή που βρίσκονται τα αντικείμενα τα οποία διασώζονται και έχουν προαχθεί από τον garbage collector. Εάν ένα αντικείμενο μετακινηθεί σε μια διαδικασία του garbage collectore, η V8 ανανεώνει όλες στης διευθύνσεις μνήμης που έδειχναν στο αντικείμενο.

Embedding V8

Ένα από τα πολύ ενδιαφέροντα χαρακτηριστικά της V8 JavaScript Engine είναι η δυνατότητα να γίνεται embeded μέσα σε άλλες εφαρμογές. Αυτό μπορεί να δώσει τη δυνατότητα σε άλλους προγραμματιστές να δημιουργήσουν plug-ins για την εφαρμογή μας χρησιμοποιώντας την γλώσσα προγραμματισμού JavaScript. Το Node.js χρησιμοποιεί την V8 JavaScript Engine με αυτόν τον τρόπο.

Παράδειγμα εφαρμογής σε C++ που εκτελεί JavaScript κώδικα

```
#include <v8.h>

using namespace v8;

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

  // Create a stack-allocated handle scope.
  HandleScope handle_scope;

  // Create a new context.
  Persistent<Context> context = Context::New();

  // Enter the created context for compiling and
  // running the hello world script.
  Context::Scope context_scope(context);

  // Create a string containing the JavaScript source code.
  Handle<String> source = String::New("'Hello' + ', World!'");

  // Compile the source code.
  Handle<Script> script = Script::Compile(source);

  // Run the script to get the result.
  Handle<Value> result = script->Run();

  // Dispose the persistent context.
  context.Dispose();

  // Convert the result to an ASCII string and print it.
  String::AsciiValue ascii(result);
  printf("%s\n", *ascii);
```

```
  return 0;
}
```

3.5 libtorrent

Η βιβλιοθήκη libtorrent είναι μια ανοικτού κώδικα υλοποίηση του πρωτοκόλλου BitTorrent. Είναι γραμμένη εξολοκλήρου χρησιμοποιώντας την γλώσσα προγραμματισμού C++ και υποστηρίζει πολλά από τα χαρακτηριστικά του πρωτοκόλλου με σημαντικότερα το DHT, IPv6, HTTP seeds και μTorrent peer exchange.

Η libtorrent υποστηρίζει τα λειτουργικά συστήματα Windows, Mac OS X, Linux και FreeBSD, χρησιμοποιώντας για να πετύχει αυτήν την ευρεία υποστήριξη λειτουργικών συστημάτων την βιβλιοθήκη Boost. Η βιβλιοθήκη Boost επεκτείνει την λειτουργικότητα της γλώσσας προγραμματισμού C++.

Η βιβλιοθήκη συνεχώς ανανεώνετε με καινούργιες εκδόσεις υποστηρίζοντας καινούργια χαρακτηριστικά του πρωτοκόλλου. Πολλά από τα χαρακτηριστικά μπορεί να απενεργοποιηθούν κατά τη μεταγλώττιση έτσι ώστε να μην περιλαμβάνετε κώδικας που δεν θα χρησιμοποιηθεί στην συγκεκριμένη περίπτωση χρήσης.

Στόχος της βιβλιοθήκης είναι να μπορεί να υποστηρίζει συστήματα τα οποία δεν έχουν τα ίδια χαρακτηριστικά και δυνατότητες. Μπορεί να υποστηρίζει ενσωματωμένα συστήματα τα οποία η δυνατότητες και τα χαρακτηριστικά τους είναι συγκεκριμένα, μέχρι την υποστήριξη εφαρμογών για επιτραπέζιους υπολογιστές ή seed-servers.

Παράδειγμα ενός απλού BitTorrent client

```
int main(int argc, char* argv[])
{
        using namespace libtorrent;
#if BOOST_VERSION < 103400
        namespace fs = boost::filesystem;
        fs::path::default_name_check(fs::no_check);
#endif

if (argc != 2)
{
        std::cerr << "usage: ./simple_client torrent-file\n"
                "to stop the client, press return.\n";
        return 1;
}

#ifndef BOOST_NO_EXCEPTIONS
        try
#endif
        {
                session s;
                s.listen_on(std::make_pair(6881, 6889));
                add_torrent_params p;
                p.save_path = "./";
```

```
                p.ti = new torrent_info(argv[1]);
                s.add_torrent(p);

                // wait for the user to end
                char a;
                std::cin.unsetf(std::ios_base::skipws);
                std::cin >> a;
        }
#ifndef BOOST_NO_EXCEPTIONS
        catch (std::exception& e)
        {
                std::cout << e.what() << "\n";
        }
#endif
        return 0;
}
```

4 Ψηφιακή Πλατφόρμα Ασύγχρονης Τηλεόρασης

4.1 Ανάλυση Προβλήματος

Για την ανάπτυξη ενός επιτυχημένου υπολογιστικού συστήματος, είναι καίριας σημασίας η προσεκτική ανάλυση του συστήματος και η θεσμοθέτηση των απαιτήσεων που πρέπει να καλύπτει. Κατά την ανάλυση και τον σχεδιασμό της ψηφιακής πλατφόρμας ασύγχρονης τηλεόρασης, θέσαμε αυστηρές απαιτήσεις όσον αφορά την βιωσιμότητα της πλατφόρμας. Η δημιουργία ενός συστήματος που δεν θα είχε πρακτική εφαρμογή, δεν θα ήταν κάτι το οποίο θα μπορούσε να χρησιμοποιηθεί στην αγορά.

Ένα από τα προβλήματα τα οποία πρέπει να αντιμετωπίσει ένα σύστημα το οποίο έχει ως κύριο αντικείμενο λειτουργίας του το διαδίκτυο είναι η κατανάλωση εύρους ζώνης (bandwidth). Λόγο της ανάγκης της πλατφόρμας να κάνει διαθέσιμο τηλεοπτικό περιεχόμενο μέσω διαδικτύου, γίνετε κατανοητό ότι ένα τέτοιο σύστημα θα έχει πολύ μεγάλη κατανάλωση bandwidth. Η σπουδαιότητα του ζητήματος γίνετε κατανοητό και από το γεγονός ότι υπάρχει ολόκληρο αντικείμενο που ασχολείται με την ανάλυση του προβλήματος και την παροχή λύσεων, γνωστή ως οικονομική των δικτύων. Η οικονομική των δικτύων μελετάει και δίνει λύσεις, για την κοστολόγηση της παροχής υπηρεσιών που κάνουν χρήση δικτύων. Αν και στην πρώτη φάση δεν μας ενδιαφέρει κάποιο επιχειρηματικό σχέδιο για την αξιοποίηση της πλατφόρμας, θέτουμε ως σημαντική απαίτηση το λογικό κόστος όσον αφορά την προσφερόμενη υπηρεσία.

Εκτός από τα προβλήματα τα οποία πρέπει να αντιμετωπίσουμε όσον αφορά την βιωσιμότητα της πλατφόρμας, υπάρχουν και προβλήματα τα οποία πρέπει παρέχουμε λύσεις και αφορούν της ανάγκες των χρηστών. Ένας σημαντικός παράγοντας για την επιτυχία της πλατφόρμας είναι η αξιόπιστη παροχή των προσφερόμενων υπηρεσιών καθώς και η ευκολία χρήσης της πλατφόρμας.

4.1.1 Απαιτήσεις Συστήματος

Κατά την φάση της ανάλυσης καταγράφηκαν κάποιες σημαντικές απαιτήσεις που θα έπρεπε να πληρεί η πλατφόρμα. Οι απαιτήσεις αυτές μπορούν να χωριστούν εννοιολογικά σε αυτές που αφορούν την απόδοση του συστήματος και σε αυτές που αναφέρονται στην γενικότερη εμπειρία χρήσης της πλατφόρμας.

Αναφορικά με την απόδοση του υλικού Set-top-Box, θα πρέπει να μπορεί να αναπαράγει τηλεοπτικό περιεχόμενο υψηλής ανάλυσης χωρίς προβλήματα. Για είναι αυτό εφικτό υπάρχουν δυο λύσεις. Η πρώτη είναι η παρουσία και χρήση ειδικού κυκλώματος για την αποκωδικοποίηση του βίντεο, έτσι ώστε το υπόλοιπο σύστημα να μπορεί να ασχοληθεί με άλλες διεργασίες. Η δεύτερη λύση είναι η υπολογιστική μονάδα να είναι αρκετά δυνατή έτσι ώστε να μπορεί να αποκωδικοποιεί το βίντεο αλλά και ταυτόχρονα να είναι σε θέσει να επιτελέσει και άλλες διεργασίες χωρίς πρόβλημα.

Λόγο της φύσης της πλατφόρμας, θα πρέπει να δοθεί μεγάλη προσοχή έτσι ώστε να μπορούν να εξυπηρετούνται ταυτόχρονα ένας μεγάλος αριθμός χρηστών. Κάθε σύνδεση που δέχεται ο κεντρικός διακομιστής θα μπορούσε να μειώσει την απόδοση του συστήματος, ακόμα και να το κάνει να λειτουργεί κανονικά. Χρειάζεται ο κεντρικός διακομιστής να είναι σε θέση να ανταποκρίνεται κάτω από μεγάλο φόρτο εργασίας και να μπορεί να υποστηρίζει έναν μεγάλο αριθμό ταυτόχρονων συνδέσεων.

4.2 Περιγραφή λειτουργίας

Είναι σημαντικό να περιγράψουμε τον τρόπο λειτουργίας της πλατφόρμας για να γίνει κατανοητό πώς το κάθε υποσύστημα επιτελεί την εργασία του. Μπορούμε να χωρίσουμε την πλατφόρμα σε δυο διακριτά κομμάτια. Το πρώτο αφορά το κομμάτι εκείνο που ποτέ δεν πρόκειται να έχει επαφή με τον τελικό χρήστη και αποτελεί την καρδιά της πλατφόρμας, δηλαδή με άλλα λόγια το λογισμικό του κεντρικού διακομιστή. Το δεύτερο κομμάτι αποτελείται από το λογισμικό το οποίο χρησιμοποιεί ο χρήστης για να έχει πρόσβαση στην πλατφόρμα. Αυτό περιλαμβάνει είτε μια συσκευή Set-top-Box και το λογισμικό που περιλαμβάνει, είτε άλλους τρόπους με τους οποίους μπορεί να έχει πρόσβαση στην πλατφόρμα όπως για παράδειγμα μέσω του web browser.

Το λογισμικό του κεντρικού διακομιστή είναι ένα σύνολο εφαρμογών το οποίο έχει δημιουργηθεί για να μπορεί να διαχειρίζεται και να κάνει διαθέσιμο το τηλεοπτικό υλικό στους χρήστες. Ένας tracker για το πρωτόκολλο BitTorrent τρέχει στον κεντρικό διακομιστή έτσι ώστε να είναι εφικτός ο διαμοιρασμός του περιεχομένου στους χρήστες αξιοποιώντας στο έπακρο το διαθέσιμο εύρος ζώνης. Επίσης ένα σύστημα το οποίο είναι υπεύθυνο για το αποδοτικό streaming του τηλεοπτικού περιεχομένου. Τέλος το σύστημα το οποίο διαχειρίζεται το τηλεοπτικό περιεχόμενο και το κάνει διαθέσιμο στους χρήστες της πλατφόρμας.

Όσον αφορά το λογισμικό το οποίο έρχεται σε επαφή με τους χρήστες, στην περίπτωση της εφαρμογής για web browsers, είναι μια σχετικά απλή υλοποίηση μιας web ιστοσελίδας που δίνει πρόσβαση στην πλατφόρμα. Το ζητούμενο ήταν δημιουργήσουμε την κατάλληλη υποδομή για την υποστήριξη αυτού του τρόπου πρόσβασης και όχι να δημιουργήσουμε μια ολοκληρωμένη web εφαρμογή. Όσον αφορά την συσκευή Set-top-Box καθώς και το λογισμικό, τα πράγματα είναι λίγο πιο πολύπλοκα καθώς αποτελεί μια ολοκληρωμένη λύση παροχής τηλεοπτικού περιεχομένου. Το λογισμικό αυτό έχει πολλά και διαφορετικά υποσυστήματα. Υποσύστημα το οποίο μεταφορτώνει το τηλεοπτικό περιεχόμενο τοπικά για να είναι διαθέσιμο προς αναπαραγωγή. Υποσύστημα το οποίο αναλαμβάνει την διεπαφή που παρουσιάζεται στον χρήστη.

Όλα τα υποσυστήματα τα οποία δημιουργήθηκαν για να υποστηρίξουν την πλατφόρμα είναι ανεξάρτητα το ένα από το άλλο. Λόγο αυτής της αρχιτεκτονικής, είναι εφικτό να αλλάξουμε τις τεχνολογίες της πλατφόρμας χωρίς να χρειάζονται μεγάλες αλλαγές στο υπόλοιπα υποσυστήματα.

4.3 Σχεδιασμός Υλοποίησης

Παρακάτω αναφέρονται όλα τα στοιχεία που χρησιμοποιήθηκαν για την υλοποίηση της ψηφιακής πλατφόρμας ασύγχρονης τηλεόρασης. Περιλαμβάνεται το λογισμικό το οποίο δημιουργήθηκε, τεχνολογίες ή προγράμματα που χρησιμοποιήθηκαν καθώς και το υλικό το οποίο βασίστηκε η πλατφόρμα.

4.3.1 Βιβλιοθήκες

Έγινε έρευνα έτσι ώστε να βρούμε της κατάλληλες βιβλιοθήκες για την χρήση τους στην πλατφόρμα. Η βιβλιοθήκες θα έπρεπε να είναι ανοικτού κώδικα, να χρησιμοποιούνται από άλλες σημαντικές εφαρμογές της πληροφορικής ή να έχουν μεγάλο εύρος χρήσης και να έχουν πολύ καλή τεκμηρίωση. Ένα επιπλέον χαρακτηριστικό το οποίο θα βοηθούσε στην επιλογή μιας βιβλιοθήκης είναι να περιλαμβάνεται στα αποθετήρια του λειτουργικού συστήματος Debian / Ubuntu.

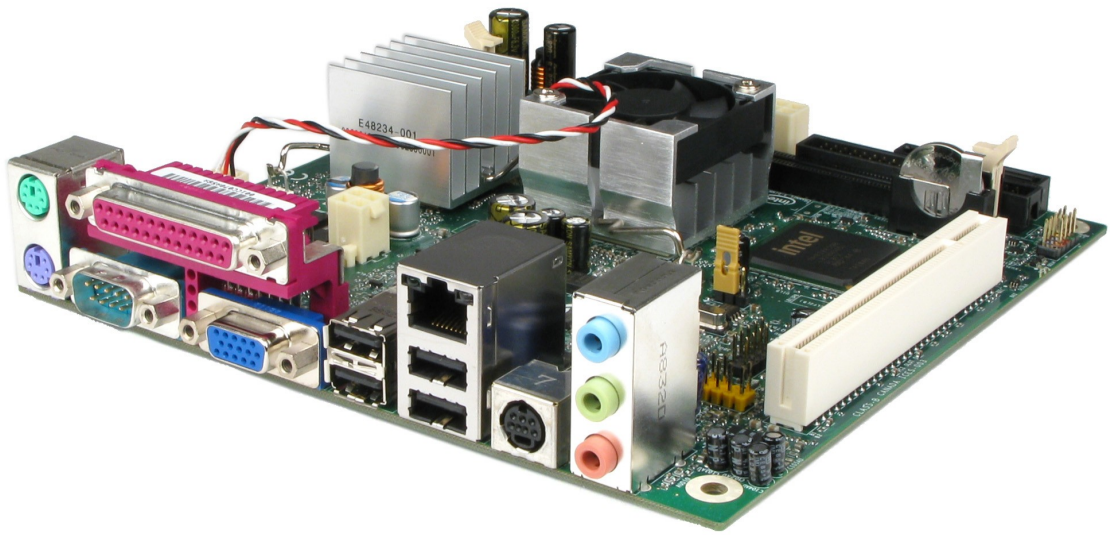
4.3.2 Hardware

4.3.2.1 Set-top-Box

Για την δημιουργία ενός πρωτότυπου Set-top-Box βασιστήκαμε στην πλακέτα D945GCLF2 της Intel. Η πλακέτα αυτή είναι εξοπλισμένη με επεξεργαστή Intel Atom Dual Core 330 1.6 Ghz, ενσωματωμένη κάρτα γραφικών Intel 945GC και υποστήριξη μνήμης RAM έως 2 GB.

Για την δημιουργία του πρωτότυπου προμηθευτήκαμε και κάποια επιπλέον εξαρτήματα τα οποία ολοκληρώνουν την πλακέτα έτσι ώστε να μπορεί να χρησιμοποιηθεί ως συσκευή set-top-box. Αυτά περιλαμβάνουν ένα Mini-ITX case, μνήμες RAM 2GB, καθώς και σκληρό δίσκο.

Η ψηφιακή πλατφόρμας ασύγχρονης τηλεόρασης βασίστηκε εξολοκλήρου σε αυτό το hardware στην φάση της ανάπτυξης. Ένα από τα πλεονεκτήματα της συσκευής είναι η πολύ χαμηλή κατανάλωση ρεύματος εν ώρα λειτουργίας. Ο επεξεργαστής Intel Atom 330 καταναλώνει από 41W έως 45W.



Εικόνα 5: Η μητρική κάρτα Intel D945GCLF2 με ενσωματωμένο επεξεργαστή Intel Atom Dual Core

Τα χαρακτηριστικά της Intel D945GCLF2 όπως εμφανίζονται στο manual του κατασκευαστή βρίσκονται παρακάτω:

- Mini-ITX form factor 6.75"x6.75"
- Dual Core Intel Atom processor
- one 240 pin DIMM socket supporting 533/667MHz single channel DDR2 up to 2GB in size
- Intel 945GC chipset: 82945GC Northbridge with integrated graphics, 82801GB ICH7 Southbridge
- GMA 950 integrated graphics

- S-Video output via Chrontel CH7021A SDTV/HDTV encoder
- RealTek ALC662 codec with HDA and 6 channel audio
- SPDIFF header on motherboard
- one PCI slot
- 8 USB2.0 ports (4 on back panel, 4 on two headers)
- one IDE interface
- two SATA2 interfaces
- one VGA connector
- one S-Video port
- one parallel port
- one serial port
- PS/2 keyboard and mouse ports
- Intel BIOS with SMBIOS support, Rapid BIOS boot, Express BIOS Update
- Gigabit Ethernet
- ACPI support, Wake on USB, PCI, PS/2, Lan
- supports Windows Vista Home Basic, Windows XP Home and Pro, 32 bit and 64 bit

Logitech diNovo Mini

Για την την πλοήγηση στην διεπαφή του Set-top-Box μπορεί να χρησιμοποιηθεί ένα οποιοδήποτε ασύρματο ή ενσύρματο πληκτρολόγιο. Ως συμπλήρωμα της συσκευής Set-top-Box θεωρούμε ως καλύτερη επιλογή το ασύρματο πληκτρολόγιο Logitech diNovi Mini.

Είναι ένα πλήρες QWERTY πληκτρολόγιο και χρησιμοποιεί την ασύρματη τεχνολογία για την σύνδεση του στην συσκευή Set-top-Box. Επίσης μπορεί να χρησιμοποιηθεί και ως συσκευή εισόδου (mouse) καθώς η περιοχή των πλήκτρων κατεύθυνσης είναι ταυτόχρονα και touchpad.

Εικόνα 6: Το ασύρματο πληκτρολόγιο Logitech diNovo Mini

4.3.2.2 Boxee Box

Το Boxee Box από την εταιρία D-Link είναι μια Set-top-Box συσκευή που βασίζεται στο λειτουργικό σύστημα Linux. Είναι σχεδιασμένο έτσι ώστε να προσφέρει με μεγάλη ευκολία πρόσβαση στο διαδίκτυο καθώς και να κάνει διαθέσιμο πολυποίκιλο multimedia περιεχόμενο μέσω του λογισμικού Boxee media center. Το υλικό βασίζετε στο CE4110 system-on-a-chip της Intel που περιλαμβάνει έναν επεξεργαστή Intel Atom χρονισμένο στα 1.2Ghz και ένα ενσωματωμένο υπολογιστή γραφικών PowerVR SGX535, 1GB Ram και 1GB NAND Flash Memory. Διαθέτει έξοδο σε HDMI (version 1.3), έξοδο για ψηφιακό ήχο μέσω οπτικής ίνας (S/PDIF), έξοδο RCA για αναλογικό στερεοφωνικό ήχο, δυο USB θύρες, μια είσοδο για κάρτα SD, μια θύρα για 100Mbps Ethernet και ενσωματωμένο δέκτη 2.4GHz 802.11n WiFi.

Εικόνα 7: D-Link Boxee Box

Λόγο του ενσωματωμένου chip γραφικών, το Boxee Box μπορεί να αναπαράγει χωρίς πρόβλημα βίντεο σε ανάλυση 1080p μέχρι και 50 καρέ το δευτερόλεπτο(fps). Ένα εξίσου σημαντικό χαρακτηριστικό είναι η δυνατότητα που προσφέρει να αναβαθμίζετε αυτόματα μέσω διαδικτύου με καινούργιες εκδόσεις του λογισμικού Boxee.

Εικόνα 8: Συνδεσιμότητα του Boxee Box

Το Boxee Box είναι κατάλληλο για την πλατφόρμα μας, καθώς έχει πολύ μεγάλες δυνατότητες για αναπαραγωγή βίντεο και δίνει επίσης μια πολύ εύχρηστη διεπαφή με πάρα πολλές δυνατότητες. Για να μπορέσουμε να εκμεταλλευτούμε το Boxee Box θα πρέπει να γράψουμε ένα πρόγραμμα ειδικά για την πλατφόρμα του Boxee. Στην ουσία θα ενθυλακώσουμε την δικιά μας ψηφιακή πλατφόρμα στην πλατφόρμα του Boxee.

4.3.2.3 Raspberry Pi

Το Raspberry Pi είναι μια πλακέτα που με πολύ μικρό κόστος προσπαθεί να ενσωματώσει κομμάτια ενός υπολογιστή. Αν και δημιουργήθηκε για καθαρά εκπαιδευτικούς σκοπούς, μπορεί να αποτελέσει την βάση για να χρησιμοποιηθεί και πέρα από τον σκοπό αυτό. Βασίζεται στο system-on-a-chip BCM2835 της Broadcom με επεξεργαστή ARM1176JZF-S χρονισμένο στα 700 Mhz, επεξεργαστή γραφικών VideoCore IV και 256MB μνήμη RAM. Δεν περιλαμβάνετε κάποιο αποθηκευτικό μέσο και για την εκκίνηση του συστήματος θα πρέπει να χρησιμοποιηθεί κάποιος σκληρός δίσκος ή κάποια κάρτα SD. Το κόστος για την πλακέτα είναι 35$, καθιστώντας το ιδιαίτερα ελκυστικό για την χρήση του στην ψηφιακή μας πλατφόρμα.

Τα πρώτα κομμάτια της πλακέτας έγιναν διαθέσιμα τον Μάρτιο του 2012. Κατά την συγγραφή της πτυχιακής εγκάρσιας, δεν μπορέσαμε να προμηθευτούμε κάποιο Raspberry Pi διότι όλα τα κομμάτια είχαν αμέσως εξαντληθεί λόγο του μεγάλου ενδιαφέροντος για το συγκεκριμένο hardware. Αν και δεν μπορούμε να τρέξουμε την εφαρμογή set-top-box στο συγκεκριμένο hardware για να μετρήσουμε την απόδοση του συστήματος και να δούμε κατά πόσο θα ήταν κατάλληλο για την πλατφόρμα μας, σύμφωνα με τα χαρακτηριστικά και τα benchmarks τα οποία υπάρχουν στο διαδίκτυο είναι μια αρκετά καλή επιλογή για την χρήση του στην πλατφόρμα.

Εικόνα 9: Raspberry Pi

4.4 Υλοποίηση

Έπειτα από αρκετή μελέτη των υπαρχόντων τεχνολογιών, έγινε η επιλογή του επιμέρους λογισμικού που θα χρησιμοποιηθούν στην πτυχιακή εργασία. Κύριο κριτήριο για την επιλογή των υποσυστημάτων που θα υποστήριζαν την πλατφόρμα ήταν όχι μόνο η δωρεάν διάθεση των προγραμμάτων αλλά η ποιότητα τους.

Πρώτα έγιναν ορισμένες δοκιμές έτσι ώστε να δούμε την καταλληλότητα των προγραμμάτων που θα χρησιμοποιηθούν στην πλατφόρμα. Εκτός από τα χαρακτηριστικά τα οποία έδινε ο δημιουργός των εφαρμογών, θα έπρεπε να δοκιμαστούν για την καταλληλότητα τους κάτω από της δικές μας ανάγκες και συνθήκες χρήσης. Σημαντικός παράγοντας για την επιλογή των προγραμμάτων ήταν η ευκολία χρήσης, η διαθεσιμότητα του πηγαίου κώδικα, η ύπαρξη ολοκληρωμένης τεκμηρίωσης, η ευρεία χρήση των προγραμμάτων από άλλα διαδεδομένα projects και τέλος η απόδοση και η αξιοπιστία τους.

Έπειτα από την επιλογή των προγραμμάτων που θα αποτελούσαν την ψηφιακή πλατφόρμα ασύγχρονης τηλεόρασης, άρχισε σταδιακά ο προγραμματισμός των υποσυστημάτων.

4.4.1 Προετοιμασία λογισμικού Set-top-Box

Για την εγκατάσταση της πλατφόρμας θα χρησιμοποιήσουμε το λειτουργικό σύστημα Ubuntu 12.04 LTS το οποίο είναι ένα δωρεάν λειτουργικό σύστημα βασισμένο στον πυρήνα του Linux. Η εγκατάσταση του λειτουργικού συστήματος αν και είναι μια πολύ απλή διαδικασία είναι πέραν από το σκοπό της πτυχιακής εργασίας, Η χρήση του Ubuntu θα γίνει και για την μεταγλώττιση του πηγαίου κώδικα αλλά και ως λειτουργικού για το Set-top-Box.

4.4.1.1 Βιβλιοθήκες

Για την μεταγλώττιση του πηγαίου κώδικα του προγράμματος για το Set-top-Box πρέπει να εγκατασταθούν ορισμένες βιβλιοθήκες ανοιχτού κώδικα. Θα πρέπει να συμπεριλαμβάνονται κατά την εγκατάσταση και η βιβλιοθήκη αλλά και τα header files. Η βιβλιοθήκη χρειάζεται κατά την εκτέλεση του προγράμματος και περιλαμβάνει την υλοποίηση ρουτινών ενώ τα header files χρειάζονται κατά την μεταγλώττιση και ορίζουν της ρουτίνες της βιβλιοθήκης. Οι εγκατάσταση τους μπορεί να γίνει είτε κατεβάζοντας τον πηγαίο κώδικα, είτε για μεγαλύτερη ευκολία κατεβάζοντας κάποιο έτοιμο πακέτο τύπου deb μέσω κάποιου προγράμματος εγκατάστασης πακέτων (Synaptic).

- Clutter (Gstreamer extension)
- GTK+
- Glib
- libtorrent (Rasterbar)
- Glade
- SQLite
- libcurl
- mRss
- Gnet

4.4.1.2 Εγκατάσταση

Για την εγκατάσταση των απαραίτητων βιβλιοθηκών θα χρησιμοποιήσουμε την εντολή *apt-get* η οποία κατεβάζει από το διαδίκτυο έτοιμα πακέτα και τα εγκαθιστά στο σύστημά μας. Λόγο ότι στα αποθετήρια του λειτουργικού συστήματος η έκδοση της βιβλιοθήκης *libtorrent* είναι παλιότερη από αυτήν που χρειαζόμαστε, θα χρειαστεί να την κατεβάσουμε, να μεταγλωττίσουμε και να την εγκαταστήσουμε μόνοι μας. Η παρακάτω εντολές εκτελούνται σε τερματικό. (Ctrl + Alt + T)

Εγκατάσταση απαραίτητων βιβλιοθηκών

```
sudo apt-get install build-essential
sudo apt-get install git-all
sudo apt-get install libglade2-dev
sudo apt-get install libgnet-dev
sudo apt-get install libclutter-1.0-dev libclutter-gst-dev
sudo apt-get install libmrss0-dev
sudo apt-get install libsqlite3-dev

wget  http://libtorrent.googlecode.com/files/libtorrent-rasterbar-0.16.0.tar.gz
tar xvf libtorrent-rasterbar-0.16.0.tar.gz
cd libtorrent-rasterbar-0.16.0
./configure –prefix=/usr
# Small fix for old BOOST_VERSION
echo "//if BOOST_VERSION >= 104610" > src/asio_ssl.cpp
make
sudo make install
```

Πλέον το σύστημα έχει όλες της βιβλιοθήκες που χρειάζεται για να γίνει η μεταγλώττιση και η εκτέλεση του λογισμικού για το Set-top-Box.

4.4.1.3 Μεταγλώττιση

Για την μεταγλώττιση των προγραμμάτων που απαρτίζουν την πλατφόρμα θα χρειαστεί αφού κάνουμε λήψη του πηγαίου κώδικα από τα αποθετήρια, να το μεταγλωττίσουμε χρησιμοποιώντας το σύστημα *automake*. Το αποθετήριο που χρησιμοποιούμε για τον πηγαίο κώδικα φιλοξενείται στο *bitbucket.org*

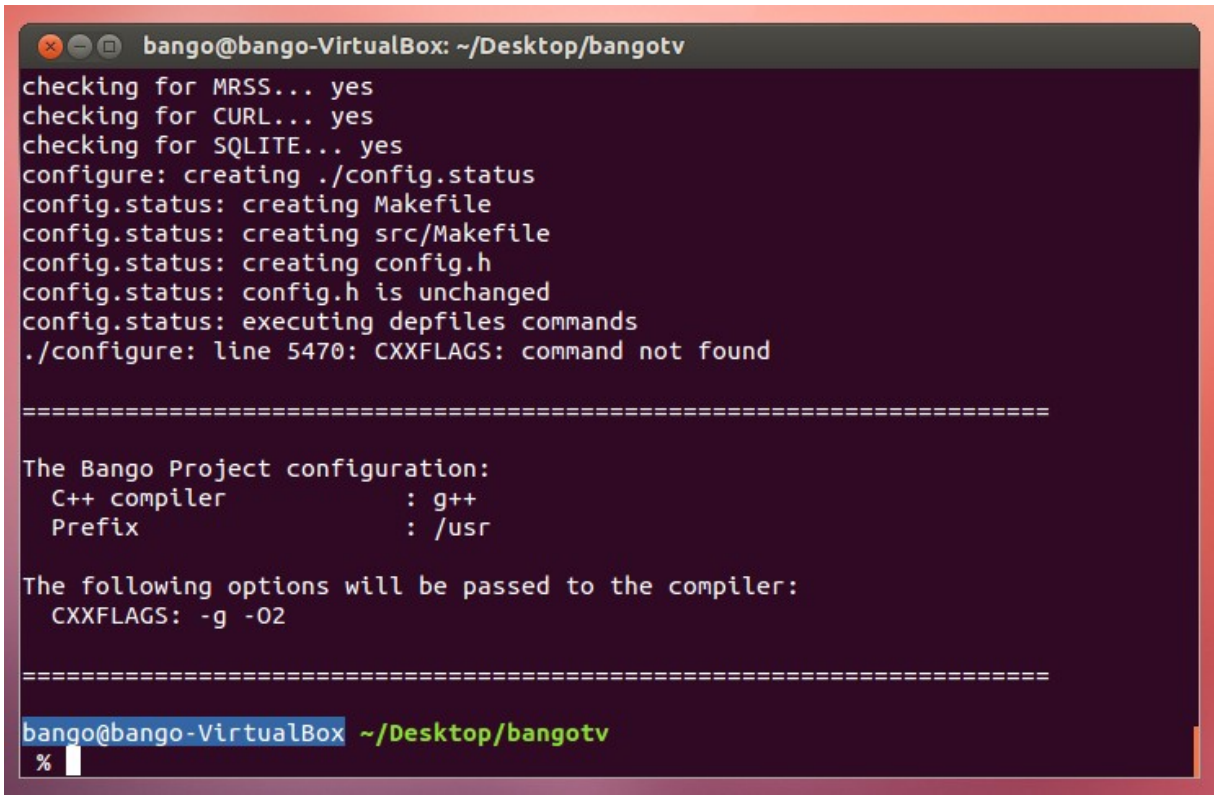
Configure & Compile

```
git clone https://angelix@bitbucket.org/angelix/bangotv.git
cd bangotv/
./configure –prefix=/usr
make
sudo make install
```

Το λογισμικό έχει εγκατασταθεί στον υπολογιστή. Εκτελώντας το για πρώτη φορά θα δημιουργήσει τα default αρχεία ρυθμίσεων. Επεξεργάζοντας το αρχείο ρυθμίσεων μπορούμε να αλλάξουμε της ρυθμίσεις του προγράμματος.

```
vim ~/.config/bango/bangod.conf
```

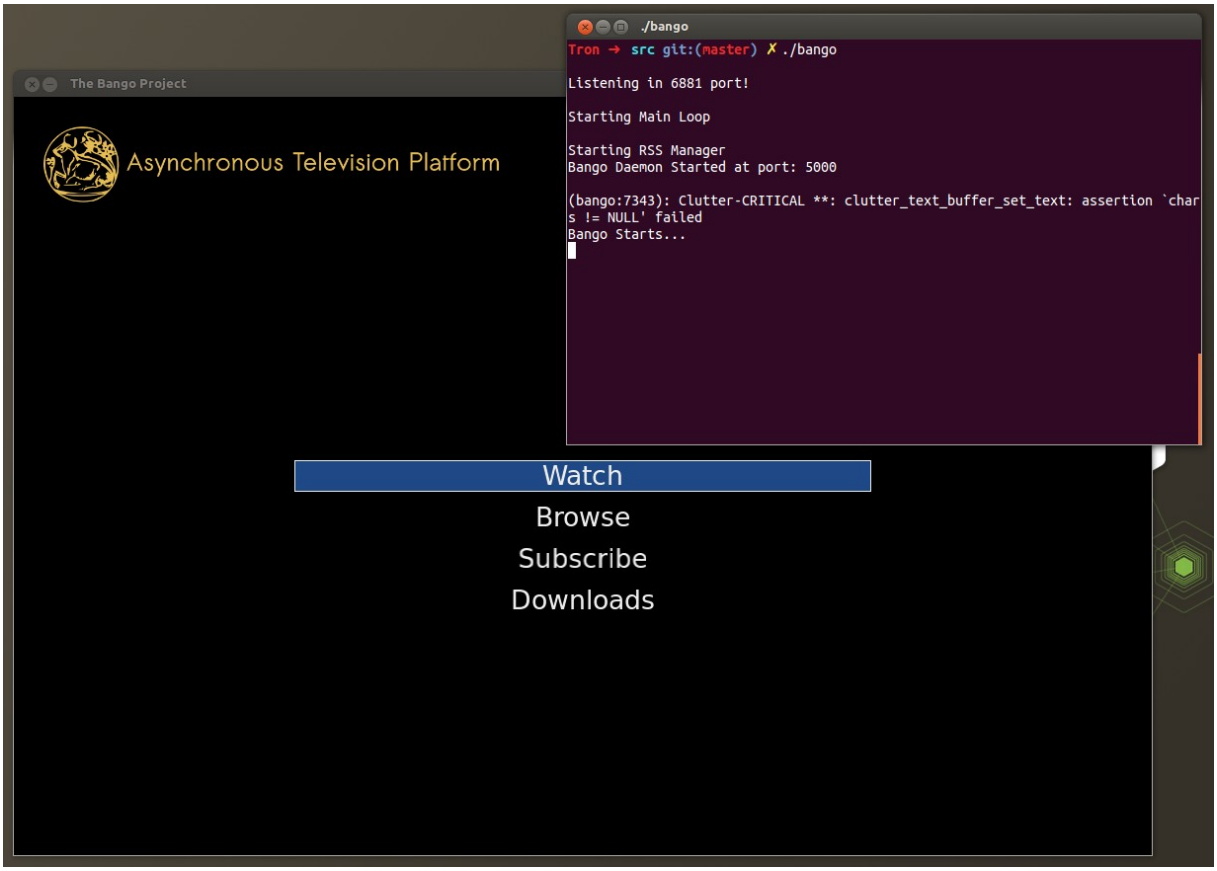
```
#Config File for Bango Daemon

[Configuration]
Default_Save_Location=/Bango
Bangod_Port=5000
Bittorent_Port_Start=6881
Bittorent_Port_End=6889
Download_Limit=-1
Upload_Limit=-1
```



Εικόνα 10: Προετοιμασία για μεταγλώττιση του πηγαίου κώδικα

4.4.1.4 Έναρξη εφαρμογής

Για την έναρξη της εφαρμογής μπορούμε να τρέξουμε το εκτελέσιμο αρχείο από την κονσόλα. Έτσι θα μπορούμε να δούμε μηνύματα από τα υποσυστήματα καθώς αυτά αρχικοποιούνται και λειτουργούν. Κατά την διάρκεια της ανάπτυξης, η κονσόλα καθώς και ο debugger είναι ένα από τα σημαντικότερα εργαλεία για να προχωρήσουμε στην αποσφαλμάτωση της εφαρμογής.



Εικόνα 11: Εκτελεσή της εφαρμογής απο την κονσόλα

Όπως φαίνεται από το παραπάνω στιγμιότυπο η εφαρμογή τρέχει σε ένα ξεχωριστό παράθυρο για λόγους development. Η κανονική λειτουργιά της εφαρμογής είναι να τρέχει σε πλήρη οθόνη.

4.4.2 Προετοιμασία λογισμικού Κεντρικού Διακομιστή

4.4.2.1 Εγκατάσταση – Μεταγλώττιση

Ο κεντρικός διακομιστής είναι ένα σημαντικό κομμάτι της πλατφόρμας και περιλαμβάνει πολλές εφαρμογές οι οποίες πρέπει να εγκατασταθούν. Παρακάτω δίνονται οι εντολές για την εγκατάσταση ορισμένων εφαρμογών. Οι παρακάτω εντολές εκτελούνται στον server και χρειάζονται δικαιώματα root..

Εγκατάσταση απαραίτητων πακέτων

```
apt-get install mysql-server
```

Οι παρακάτω εντολές εγκαθιστούν το node.js (χρειάζονται δικαιώματα root)

```
wget http://nodejs.org/dist/v0.6.18/node-v0.6.18.tar.gz
tar xvf node-v0.6.18.tar.gz
cd node-v0.6.18./configure --prefix=/usr
make
make install
node -v
```

Εγκατάσταση του XBT Tracker

```
apt-get install cmake g++ libboost-date-time-dev libboost-dev
libboost-filesystem-dev libboost-program-options-dev libboost-
regex-dev libboost-serialization-dev libmysqlclient15-dev make
subversion zlib1g-dev
svn co http://xbt.googlecode.com/svn/trunk/xbt/misc xbt/misc
svn co http://xbt.googlecode.com/svn/trunk/xbt/Tracker
xbt/Tracker
cd xbt/Tracker
./make.sh
cp xbt_tracker.conf.default xbt_tracker.conf
```

4.4.3 Ενσωμάτωση πλατφόρμας στο Boxee Box

Η πλατφόρμα του Boxee μας επιτρέπει να δημιουργήσουμε εφαρμογές οι οποίες επεκτείνουν το περιεχόμενο και της δυνατότητες του Boxee Box. Το προγραμματιστικό περιβάλλον που προσφέρεται από την πλατφόρμα του Boxee περιλαμβάνει API για την γλώσσα Python.

Ένας εύκολος τρόπος για την ενοποίηση του δικού μας περιεχομένου στην πλατφόρμα του Boxee είναι μέσω ενός RSS/Library Specification. Το RSS δημιουργείται από το κεντρικό σύστημα διαχείρισης του περιεχομένου που βρίσκεται στον κεντρικό διακομιστή. Το RSS περιγράφει το περιεχόμενο της δικής μας ασύγχρονης πλατφόρμας, μεταφορτώνεται από το Boxee και το περιεχόμενο μας ενσωματώνεται και παρουσιάζεται από την διεπαφή. Όταν ο χρήστης επιλέξει το τηλεοπτικό περιεχόμενο που θέλει να παρακολουθήσει, τότε το υποσύστημα Streaming over HTTP που έχουμε δημιουργήσει με το node.js μεταδίδει σε ροή το βίντεο προς το Boxee.

Το απαραίτητο αρχείο descriptor.xml για τη δήλωση της εφαρμογής στο Boxee

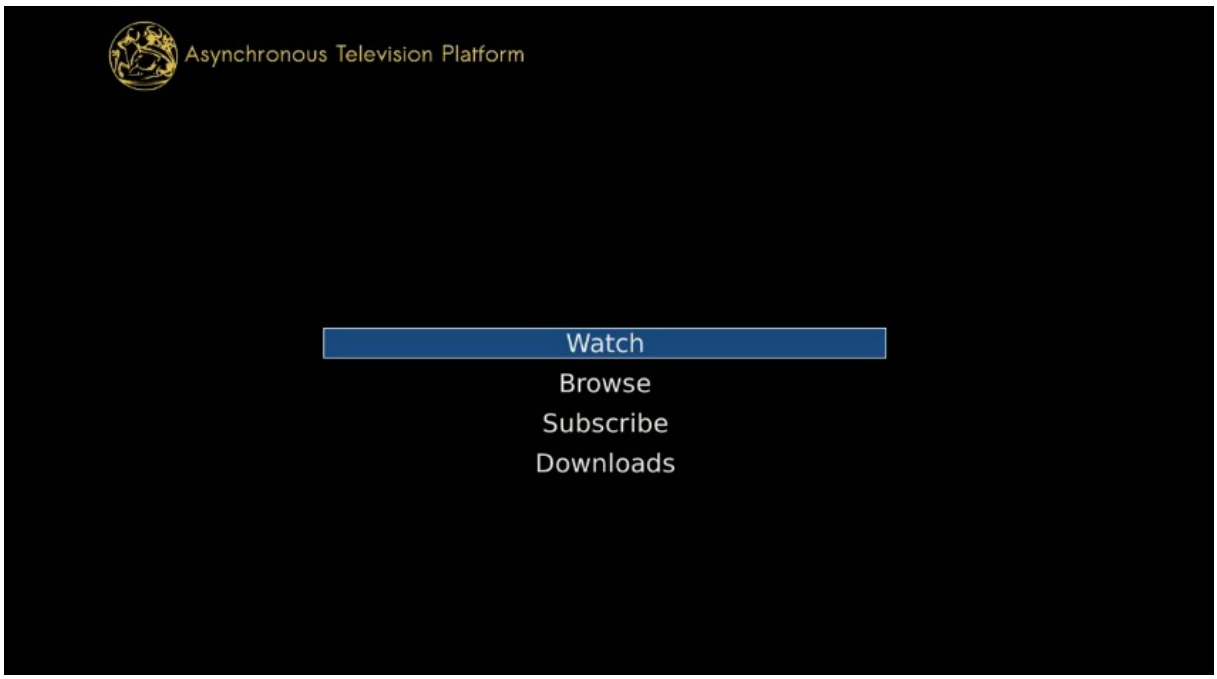
```
<app>
  <id>bangotv</id>
  <name>BangoTV</name>
  <version>1.0</version>
  <description>Asynchronous Television Platform</description>
  <thumb>http://bango.avenet.gr/thumb.png</thumb>
  <media>video</media>
```

```
   <copyright>Angelos Veglektsis</copyright>
   <email>angelix@vegle.gr</email>
   <type>rss</type>
   <url>rss://bango.avenet.gr/rss/boxee.php</url>
   <platform>all</platform>
   <minversion>0.9.11</minversion>
   <startWindow>14000</startWindow>
   <repository>http://dir.boxee.tv/apps/</repository>
   <test-app>true</test-app>
 </app>
```

4.5 Η Διεπαφή του Set-top-Box

Η διεπαφή της εφαρμογής που τρέχει στο Set-top-Box δημιουργήθηκε για να αποτελέσει το κύριο μέσο με το οποίο θα μπορεί ο χρήστης να έχει πρόσβαση στην πλατφόρμα ασύγχρονης τηλεόρασης.

Όπως έχουμε αναφέρει, θέλαμε η διεπαφή να είναι όσο το δυνατόν πιο απλή έτσι ώστε να γίνεται ξεκάθαρο στον χρήστη ο τρόπος λειτουργίας της πλατφόρμας. Κατανοούμε ότι για την δημιουργία ενός επιτυχημένου και όμορφου UI είναι σημαντικό να γίνει συγκεκριμένη έρευνα και μελέτη. Λόγο του υπόβαθρου μας ως προγραμματιστές και όχι ως σχεδιαστές διεπαφής, επιλέξαμε να δημιουργήσουμε μια διεπαφή που θα ήταν αξιοπρεπής εμφανισιακά και παράλληλα θα ήταν εύκολη στην πλοήγηση.



Εικόνα 12: Κεντρικό μενού της διεπαφής

Το κεντρικό μενού είναι το σημείο της εφαρμογής από το οποίο ο χρήστης μπορεί να περιηγηθεί στις διάφορες λειτουργίες της πλατφόρμας. Από τις τέσσερις επιλογές του μενού που

προσφέρονται , ο χρήστης μπορεί να πλοηγηθεί μέσω των πλήκτρων κατεύθυνσης (πάνω , κάτω) χρησιμοποιώντας την συσκευή Logitech niNovo Mini ή οποιαδήποτε άλλης συνδεδεμένης συσκευής εισόδου. Η επιλογή κατηγορίας γίνεται με το πλήκτρο *'enter'*.

Στην ενότητα Subscribe παρουσιάζονται όλες οι τηλεοπτικές εκπομπές που προσφέρονται από την πλατφόρμα. Οι πληροφορίες που παρουσιάζονται στον χρήστη μπορούν να του δώσουν μια πρώτη ιδέα σχετικά με το τηλεοπτικό πρόγραμμα. Για κάθε τηλεοπτικό πρόγραμμα, μια φωτογραφία καθώς και μια περιγραφή του τηλεοπτικού προγράμματος εμφανίζονται δίπλα από την λίστα με τις διαθέσιμες εκπομπές.

Ο χρήστης μπορεί να επιλέξει οποιαδήποτε εκπομπή επιθυμεί να παρακολουθήσει. Για της ήδη επιλεγμένες εκπομπές, εμφανίζεται ένα πράσινο κουτάκι δίπλα από τον τίτλο της εκπομπής. Οι επιλεγμένες εκπομπές που δηλώνονται στην εφαρμογή, είναι αυτές οι εκπομπές που το σύστημα γνωρίζει ότι ενδιαφέρουν τον χρήστη. Η εφαρμογή είναι σε θέση να γνωρίζει πότε υπάρχουν νέα επεισόδια και να τα μεταφορτώνει αυτόματα, έτσι ώστε να είναι άμεσα διαθέσιμα στον χρήστη για προβολή.

Όταν κάποιος χρήστης αποεπιλέξει κάποια τηλεοπτική εκπομπή που παρακολουθούσε παλιότερα, τα ήδη μεταφορτωμένα τηλεοπτικά επεισόδια δεν διαγράφονται αλλά παραμένουν στην διάθεση του χρήστη για να τα παρακολουθήσει, η αλλαγή στην λειτουργία της εφαρμογής είναι ότι πλέον δεν μεταφορτώνει αυτόματα τα καινούργια επεισόδια που γίνονται διαθέσιμα στην πλατφόρμα.

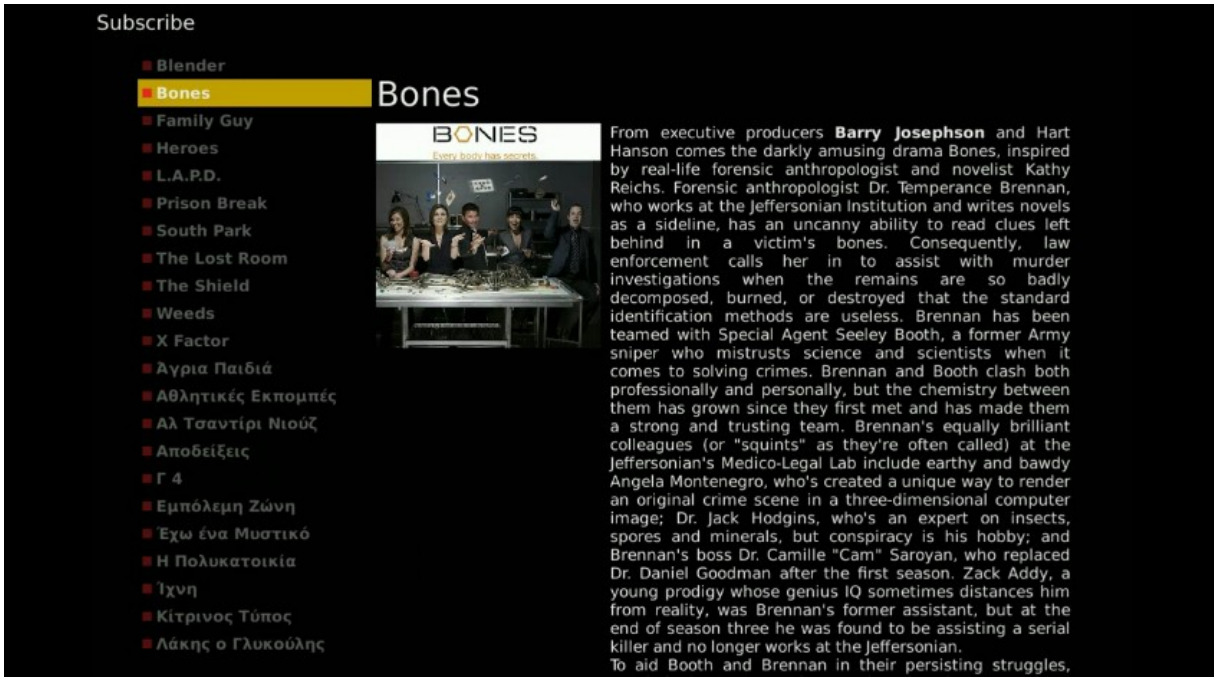
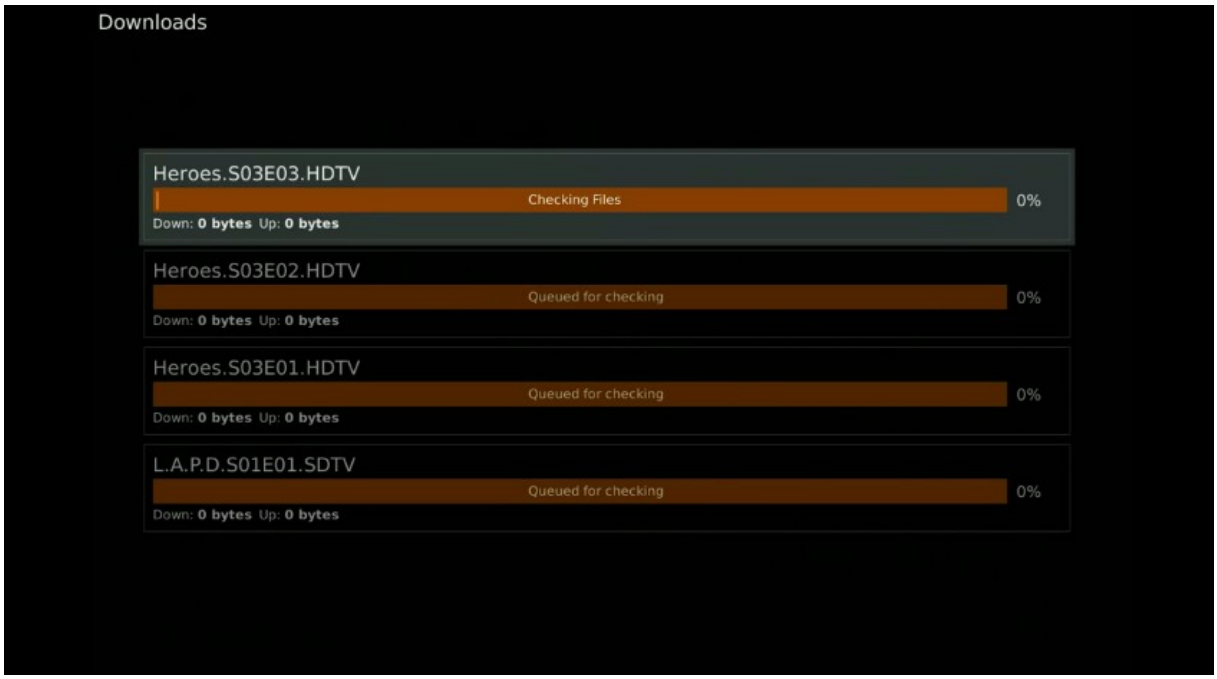


Illustration 1: Η οθόνη επιλογής τηλεοπτικών προγραμμάτων



Εικόνα 13: Η οθόνη μεταφόρτωσης των τηλεοπτικών προγραμμάτων

Η οθόνη μεταφόρτωσης είναι μια προβολή του υποσυστήματος μεταφόρτωσης. Οποιοδήποτε τηλεοπτικά προγράμματα μεταφορτώνονται εκείνη την στιγμή ή έχουν ολοκληρωθεί, εμφανίζονται στην οθόνη αυτή. Ο χρήστης μπορεί να πληροφορηθεί την κατάσταση της μεταφόρτωσης, το ποσοστό ολοκλήρωσης καθώς και την ταχύτητα μετάδοσης δεδομένων. Επίσης μπορεί να κάνει παύση μια μεταφόρτωση ή να την ξανά εκκινήσει.

Εικόνα 14: Η οθόνη των διαθέσιμων τηλεοπτικών εκπομπών α'

Η οθόνη των διαθέσιμων τηλεοπτικών προγραμμάτων, που είναι προσβάσιμη από το την επιλογή Browse του κεντρικού μενού, παρουσιάζει όλες της εκπομπές που έχουν μεταμορφωθεί στην συσκευή Set-top-Box. Μαζί με τους τίτλους των τηλεοπτικών εκπομπών σχεδόν το μισό μέρος της οθόνης παρουσιάζεται μια εικόνα που αντιστοιχεί στο επιλεγμένο τηλεοπτικό πρόγραμμα. Έτσι ο χρήστης μπορεί να έχει καλύτερη κατανόηση για το τι έχει επιλέξει, είτε βάση της εικόνας είτε του επιλεγμένου τίτλου. Η ταξινόμηση των εκπομπών γίνεται αλφαβητικά.

Για την πιο όμορφη εναλλαγή και πλοήγηση ανάμεσα στις εκπομπές, έχει δημιουργηθεί ένα animation έτσι ώστε οι εναλλαγές των εικόνων να γίνονται ευχάριστες στο μάτι. Το animation που χρησιμοποιούμε είναι απλό στην ιδέα, αλλά δίνει μια όμορφη νότα και παρουσιάζει της δυνατότητες που έχουμε μέσω του OpenGL.

Η επιλογή της τηλεοπτικής εκπομπής γίνεται με το πλήκτρο *'enter'*. Η επόμενη οθόνη παρουσιάζει τα διαθέσιμα τηλεοπτικά επεισόδια για την επιλεγμένη εκπομπή.

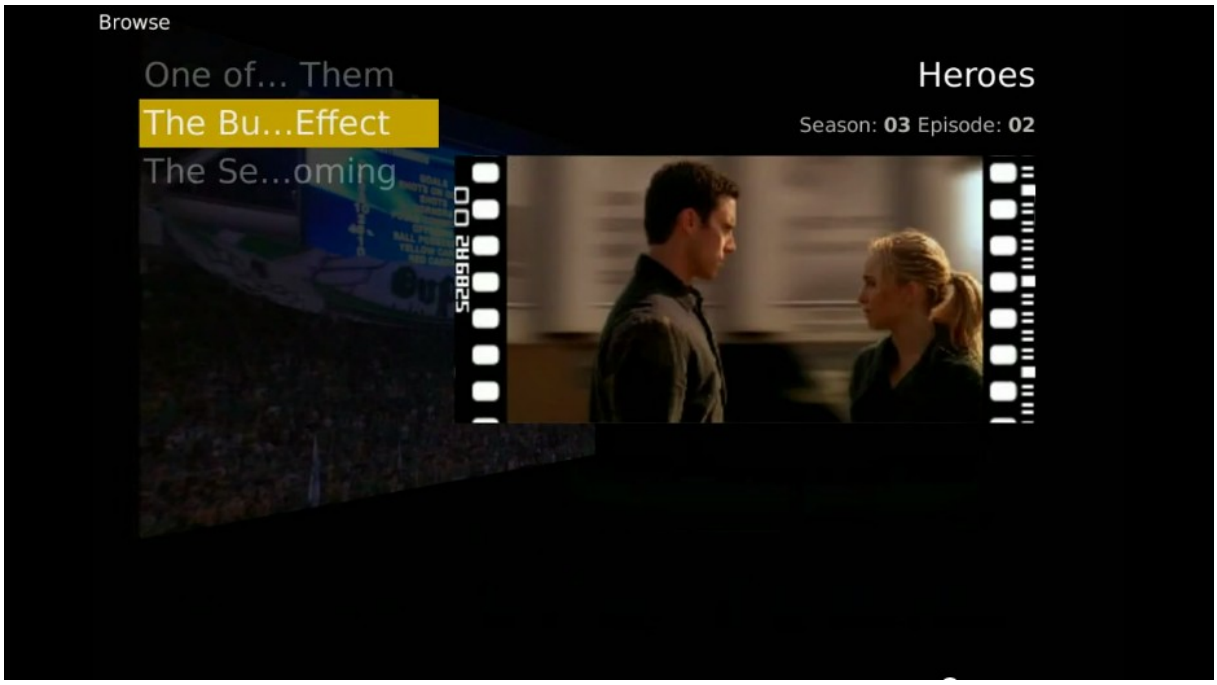
Εικόνα 15: Η οθόνη των διαθέσιμων τηλεοπτικών εκπομπών β'

Η οθόνη των διαθέσιμων τηλεοπτικών επεισοδίων, παρουσιάζει τα επεισόδια για μια συγκεκριμένη εκπομπή που είναι άμεσα διαθέσιμα για αναπαραγωγή. Η διεπαφή κρατάει αρκετά σημεία από την προηγούμενη οθόνη των διαθέσιμων εκπομπών, με κάποιες μικρές προσθήκες. Τα επεισόδια ταξινομούνται ανάλογα με την σειρά προβολής τους. Υπάρχουν κατάλληλες ενδείξεις για την αριθμό της σεζόν και του επεισοδίου.

Επειδή πολλές φόρες κάποιος χρήστης μπορεί να μην θυμάται ποια επεισόδια έχει παρακολουθήσει, υπάρχει ένδειξη δίπλα από τον τίτλο του επεισοδίου, που καταδεικνύει εάν έχει γίνει η αναπαραγωγή του επεισοδίου. Τα ίδια animations ισχύουν όπως και στην προηγούμενη οθόνη. Η εικόνα που καταλαμβάνει χώρο λίγο μεγαλύτερο από το μισό της οθόνης, πλέον είναι ένα στιγμιότυπο από το βίντεο του επεισοδίου.

Η έναρξη της αναπαραγωγής του επεισοδίου, γίνεται με το πλήκτρο *'enter'* του

πληκτρολογίου. Εάν σε περίπτωση που γίνεται αναπαραγωγή κάποιου άλλου επεισοδίου, τότε γίνεται αντικατάσταση του επεισοδίου που αναπαράγεται.



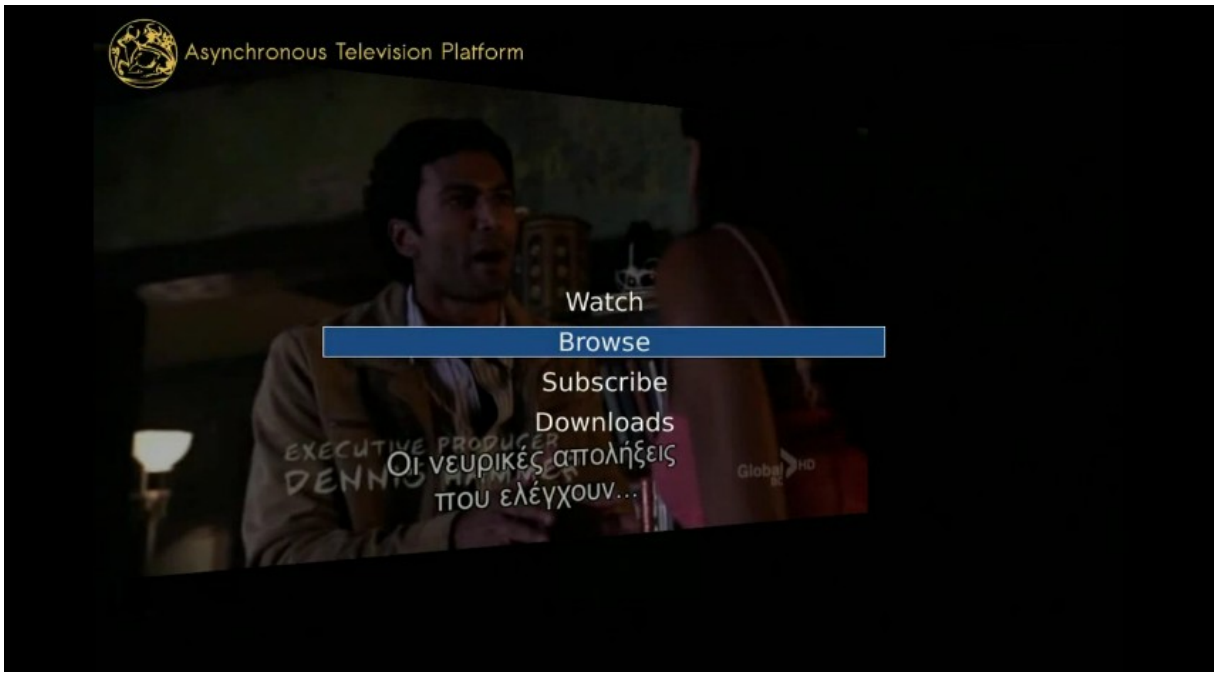
Εικόνα 16: Η οθόνη των διαθέσιμων τηλεοπτικών επεισοδίων

Στην οθόνη αναπαραγωγής, οποιαδήποτε στιγμή ο χρήστης μπορεί να έχει πρόσβαση μέσω On Screen Display (OSD) σε χρήσιμες πληροφορίες σχετικά με το τηλεοπτικό επεισόδιο που αναπαράγεται.

Εικόνα 17: Η οθόνη αναπαραγωγής, στην πάνω μεριά της οθόνης διακρίνεται το OSD

Οι πληροφορίες που βρίσκονται στο OSD πληροφορούν τον χρήστη σχετικά με τον τίτλο του επεισοδίου, την κατάσταση της αναπαραγωγής (play/pause), το συνολικό χρόνο του επεισοδίου και το χρόνο που έχει παρέλθει από την αρχή του επεισοδίου (elapsed time).

Ο χρήστης οποιαδήποτε στιγμή μπορεί να έχει πρόσβαση στο κεντρικό μενού καθώς και στις άλλες οθόνες χωρίς να σταματήσει η αναπαραγωγή του βίντεο. Η διεπαφή είναι σχεδιασμένη έτσι ώστε τα διάφορα επίπεδα μέσα στον τρισδιάστατο χώρο να αλλάζουν θέση και μεσώ διαφάνειας να δημιουργείται ένας συνδυασμός των δυο οθονών χωρίς να υπάρχει πρόβλημα στην πλοήγηση.



Εικόνα 18: Τα επίπεδα της διεπαφής με διαφάνεια

4.5.1 Πλοήγηση

Η πλοήγηση στην διεπαφή μπορεί να γίνει με την χρήση οποιουδήποτε πληκτρολογίου. Η χρήση του ασύρματου πληκτρολογίου Logitech diNovo Mini ενδείκνυται καθώς είναι ένα μικρό και πλήρες πληκτρολόγιο QWERTY.

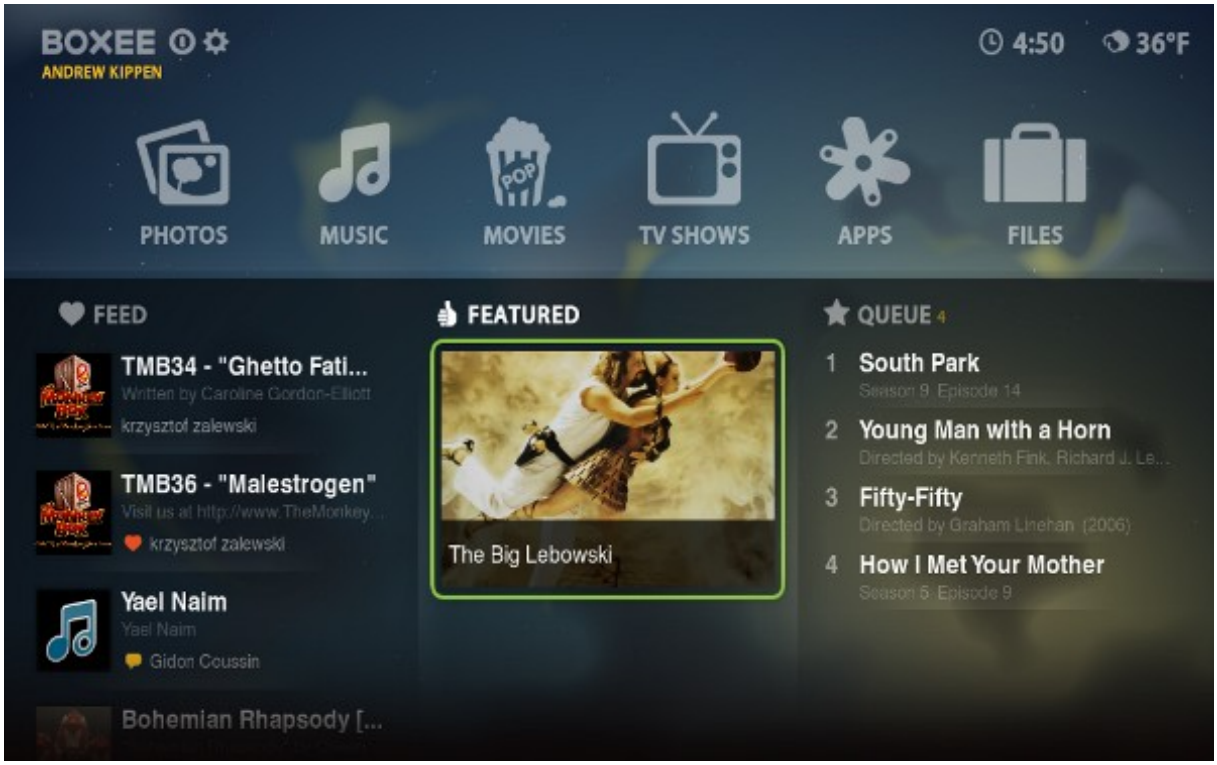
Η λειτουργίες της πλοήγησης συνήθως είναι ίδιες σε οποιοδήποτε σημείο της εφαρμογής και περιγράφονται στην παρακάτω λίστα,

- **Πλήκτρα διεύθυνσης *Up*, *Down*** – χρησιμοποιούνται για την πλοήγηση στα μενού
- **Πλήκτρα διεύθυνσης *Left*, *Right*** – χρησιμοποιούνται για την πλοήγηση στα μενού σε οριζόντια λογική και στην οθόνη αναπαραγωγής χρησιμοποιούνται ως χρονική πλοήγηση μέσα στο βίντεο
- ***Enter*** – εκτελεί την action ενέργεια ανάλογα με την οθόνη. (play/pause , επιλογή/αποεπιλογή)
- ***Space*** – αλλάζει την κατάσταση play/pause του βίντεο
- ***Escape*, *Backspace*** – χρησιμοποιούνται στην πλοήγηση ως επιστροφή στα μενού
- ***Tab*** – πρόσβαση στο μενού χωρίς να σταματήσει η αναπαραγωγή
- **Πλήκτρο *O*** – εμφάνιση του OSD
- **Πλήκτρο *1-9*** – πλοήγηση στο 10 – 90 % του βίντεο
- **Πλήκτρο *R*** – διαγραφή επιλογής από την οθόνη Download
- **Πλήκτρο *S*** – έλεγχος για νέα επεισόδια

4.6 Η Διεπαφή του Boxee Box

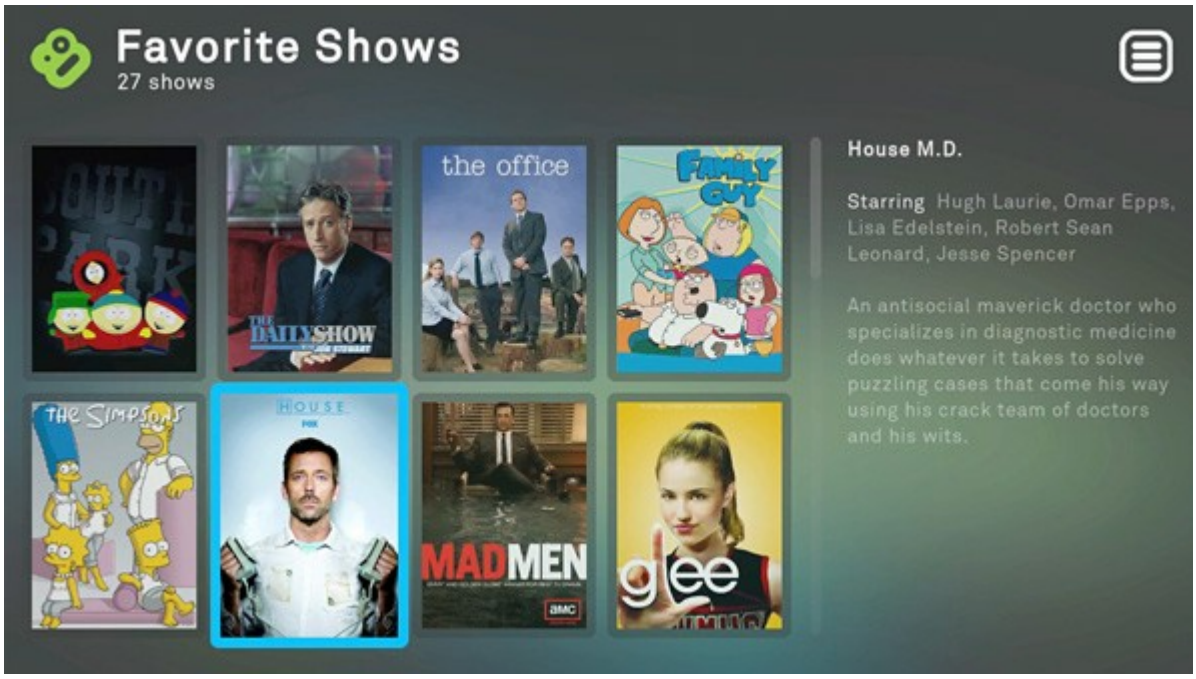
Η διεπαφή του Boxee Box περιλαμβάνει πολλά στοιχεία τα οποία την κάνουν ιδιαίτερα ελκυστική και λειτουργική. Ένα κύριο χαρακτηριστικό της διεπαφής είναι η παρουσίαση περιεχομένου που έχει γίνει μοιραστεί χρήστες κοινωνικών δικτύων.

Στα παρακάτω στιγμιότυπα οθόνης παρουσιάζεται η διεπαφή της συσκευής Boxee Box



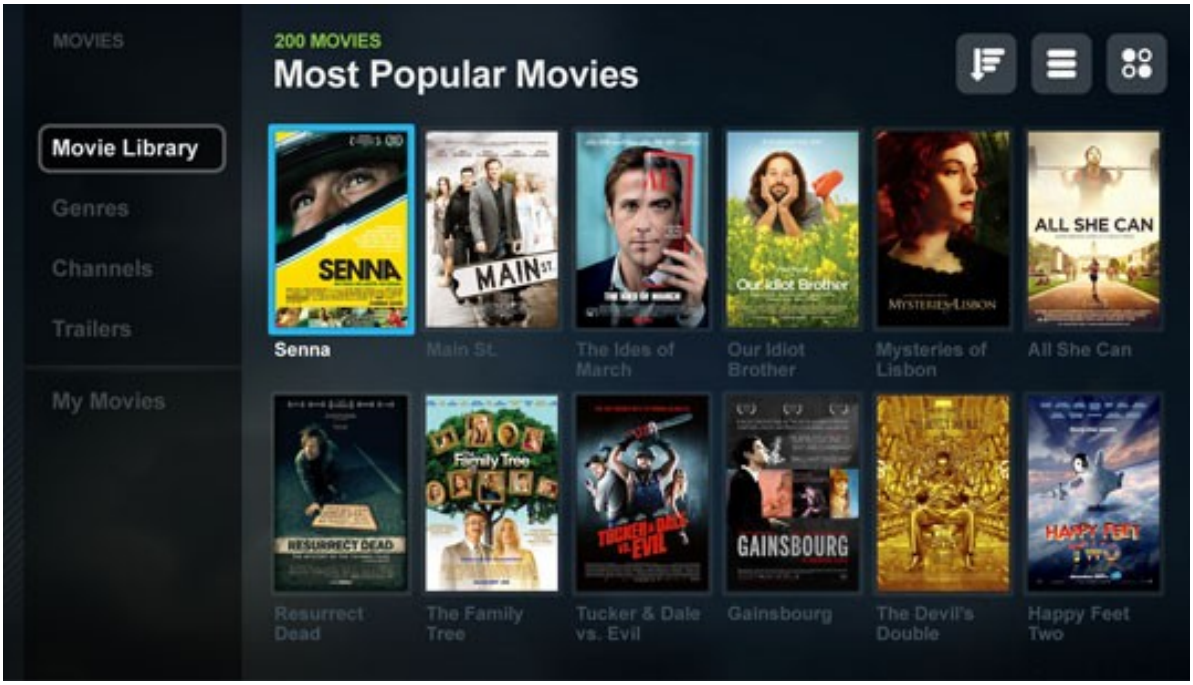
Εικόνα 19: Η κεντρική οθόνη του Boxee Box

Η κεντρική οθόνη του Boxee Box δίνει πρόσβαση σε όλες τις δυνατότητες και λειτουργίες της εφαρμογής. Η οθόνη είναι εννοιολογικά χωρισμένη σε δυο οριζόντια μέρη. Το πάνω μέρος δίνει πρόσβαση στις υπηρεσίες που προσφέρονται από την διεπαφή όπως η προβολή φωτογραφιών, η αναπαραγωγή μουσικής και άλλες εφαρμογές που έχουν γραφτεί για την πλατφόρμα του Boxee Box. Το κάτω μέρος περιλαμβάνει το Feed, Featured και Queue. Στο Feed υπάρχουν βίντεο ή μουσική που έχουν αναπαραχθεί από άλλους χρήστες κοινωνικών δικτύων με τους οποίους είμαστε συνδεδεμένοι. Το Featured δίνει πρόσβαση σε πολυμέσα τα οποία είναι ιδιαίτερα δημοφιλή στην πλατφόρμα του Boxee Box. Το Queue περιλαμβάνει όλα αυτά τα οποία εμείς έχουμε επιλέξει να παρακολουθήσουμε αργότερα.



Εικόνα 20: Αγαπημένα προγράμματα στο Boxee Box

Η διεπαφή του Boxee Box μέσα από ένα εύχρηστο περιβάλλον δίνει την δυνατότητα στον χρήστη να πλοηγηθεί ανάμεσα στα διαθέσιμα τηλεοπτικά προγράμματα ή σε κάποιες κατηγορίες αυτών όπως τα χαρακτηρισμένα από τον χρήστη ως "Αγαπημένα'". Ο χρήστης εκτός από τον τίτλο της τηλεοπτικής σειράς, μπορεί να διαβάσει περισσότερες πληροφορίες όπως οι πρωταγωνιστές καθώς και μια περιγραφή του προγράμματος.



Εικόνα 21: Βιβλιοθήκη ταινιών στο Boxee Box

4.7 Χρήση της πλατφόρμας από Web Browser

Η αναπαραγωγή του τηλεοπτικού περιεχομένου από εφαρμογή web browser μπορεί να γίνει με χρήση του plug-in Adobe Flash ή με το video tag της HTML5. Για να είναι εφικτή η αναπαραγωγή βίντεο από εφαρμογή web browser αναπτύχθηκε ένα ξεχωριστό υποσύστημα αποκλειστικά για αυτό τον σκοπό. Το υποσύστημα Streaming over HTTP περιγράφετε παρακάτω.

Παράδειγμα χρήσης του video tag της HTML5

```
<video width="320" height="240" controls="controls">
  <source src="video_url.mp4" type="video/mp4" />
  <source src="video_url.ogg" type="video/ogg" />
  Your browser does not support the video tag.
</video>
```

4.8 Η Τεχνολογία της πλατφόρμας

4.8.1 Υποσύστημα Streaming over HTTP

Για την υποστήριξη ενός συστήματος το οποίο θα έχει τη δυνατότητα να κάνει streaming σε μεγάλο αριθμό χρηστών ταυτόχρονα χωρίς προβλήματα, βασιστήκαμε στο node.js. Το node.js μας δίνει τη δυνατότητα να υποστηρίξουμε ταυτόχρονα πολλούς χρήστες λόγο της αρχιτεκτονικής του. Η συνήθης πρακτική είναι για κάθε σύνδεση να δημιουργείται ένα καινούργιο thread ή process. Αν και αυτό έχει το πλεονέκτημα να εξυπηρετείτε ο χρήστης ανεξάρτητα από τους άλλους και είναι πιο εύκολος ο προγραμματισμός ενός τέτοιου συστήματος, σύντομα θα δημιουργηθεί πρόβλημα απόδοσης του συστήματος καθώς όσο αυξάνονται οι χρήστες, αυξάνονται και τα threads ή processes. Για κάθε thread ή process καταλαμβάνετε ένας σημαντικός αριθμός πόρων του συστήματος κατά κύριο λόγο μνήμη.

Η διαφορετική προσέγγιση που χρησιμοποιείται από το node.js είναι μέσω του Asynchronous IO και του event-loop. Στο node.js υπάρχει μόνο ένα process που διαχειρίζεται όλη την επικοινωνία με τους χρήστες. Κάθε καινούργια σύνδεση που δημιουργείται μπαίνει σε μια ουρά. Το node.js αφιερώνεται σε μια και μόνο σύνδεση μέχρι που θα συμβεί κάποιο IO. Το IO μπορεί να είναι η ανάγνωση από τον δίσκο ή το δίκτυο, κάποιο query σε μια βάση δεδομένων ή οτιδήποτε θα καθυστερούσε την εκτέλεση του προγράμματος. Σε αυτήν την περίπτωση η το node.js σταματάει να εξυπηρετεί την συγκεκριμένη σύνδεση, μέχρι να ολοκληρωθεί το IO και συνεχίζει να εξυπηρετεί άλλες συνδέσεις.

Εδώ είναι σημαντικό να σημειώσουμε ότι σε ένα υπολογιστικό σύστημα συνήθως ο μεγαλύτερος χρόνος στην εκτέλεση ενός προγράμματος αφορά το IO και όχι την εκτέλεση των εντολών. Το node.js εκμεταλλεύεται αυτό το γεγονός έτσι ώστε να μπορεί να εξυπηρετεί ταυτόχρονα πολλές συνδέσεις χωρίς να χρησιμοποιεί πολλούς πόρους από το σύστημα.

Για να γίνει πιο κατανοητός ο τρόπος λειτουργίας του node.js θα παραθέσουμε ένα παράδειγμα. Ένα πρόγραμμα που να γράφει συνεχώς στην κονσόλα.

Συνεχόμενο loop που γράφει στην κονσόλα ένα κείμενο αλλά μπλοκάρει το node.js

```
while(1){
```

```
    console.log("write something");
}
```

Το ίδιο αποτέλεσμα χωρίς να μπλοκάρει το node.js

```
function f(){
    console.log("write something");
    process.nextTick(f);
}
f();
```

Το πρώτο παράδειγμα εκτελεί συνεχώς μια loop και γράφει στην κονσόλα χωρίς να μπορεί το node.js να σταματήσει την εκτέλεση του προγράμματος για να εξυπηρετήσει άλλες διεργασίες του event-loop. Πρέπει να αποφεύγονται καταστάσεις οι οποίες είναι πολύ CPU intensive.

Στο δεύτερο παράδειγμα έχουμε το ίδιο τελικό αποτέλεσμα, δηλαδή γράφουμε συνεχώς στην κονσόλα, αλλά σε αυτήν την περίπτωση δίνουμε την ευκαιρία στο node.js να εκτελέσει και άλλες διεργασίες του event-loop. Η εντολή *process.nextTick(f)* δηλώνει ότι η συνάρτηση *f* θα μπει στην ουρά για να εκτελεστεί από το event loop.

Έχοντας δώσει μια απλή εξήγηση για το πως λειτουργεί το node.js είναι κατανοητό ότι μπορούμε να χρησιμοποιήσουμε το node.js για να δημιουργήσουμε ένα πολύ ευέλικτο και αποδοτικό streaming σύστημα. Παρακάτω θα παραθέσουμε και θα εξηγήσουμε ένα απλουστευμένο streaming σύστημα γραμμένο σε JavaScript για την πλατφόρμα του node.js

Χρησιμοποιώντας το module HTTP του node.js θα ακούσουμε για τα requests των χρηστών και θα απαντήσουμε σε αυτά. Η συνάρτηση *stream εκτελείται* με ορίσματα το αντικείμενο Request που περιέχει πληροφορίες για το HTTP αίτημα του χρήστη και το αντικείμενο Response που θα το χρησιμοποιήσουμε για να στείλουμε δεδομένα στον χρήστη.

```
function stream(req, res) {
    var stream;
    var stat;
    var info = {};

    var reqUrl = url.parse(req.url, true);

    info.path = typeof reqUrl.pathname === "string" ?
reqUrl.pathname.substring(1) : undefined;


    info.path = info.path.substring(settings.rootPath.length);
    info.file = info.path.match(/(.*[\/|\\])?(.+?)$/)[2];
```

Γνωρίζοντας το αρχείο το οποίο αιτείται ο χρήστης για streaming, είναι σημαντικό να γνωρίζουμε κάποια γνωρίσματα για το αρχείο που θα μεταδώσουμε, όπως ο τύπος του βίντεο, το μέγεθος του αρχείου κτλ. Η συνάρτηση statSync επιστέφει κάποια στοιχεία για το αρχείο. Εδώ να σημειώσουμε ότι η συγκεκριμένη συνάρτηση είναι synchrounous, και την χρησιμοποιούμε για λόγους ευκολίας.

```
    try {
          stat = fs.statSync(info.path);
```

```
            if (!stat.isFile()) {
                  return false;
            }
      } catch (e) {
            return false;
      }

      info.start = 0;
      info.end = stat.size - 1;
      info.size = stat.size;
      info.modified = stat.mtime;

      info.length = info.end - info.start + 1;
```

Για να ξεκινήσουμε την διαδικασία του streaming είναι σημαντικό να στείλουμε της κατάλληλες HTTP επικεφαλίδες δηλώνοντας διάφορες πληροφορίες όσον αφορά την σύνδεση καθώς και τα δεδομένα τα οποία θα στείλουμε στην εφαρμογή του χρήστη.

```
      var code = 200;
      var header;

      header = {
            "Cache-Control": "public",
            Connection: "keep-alive",
            "Content-Type": info.mime,
            "Content-Disposition": "inline; filename=" + info.file + ";"
      };

      header.Pragma = "public";
      header["Last-Modified"] = info.modified.toUTCString();
      header["Content-Transfer-Encoding"] = "binary";
      header["Content-Length"] = info.length;

      header.Server = "Video Streaming Server";

      res.writeHead(code, header);
```

Χρησιμοποιώντας τα Streams του node.js μπορούμε με πολύ εύκολο τρόπο να διαβάζουμε από ένα αρχείο και να το στέλνουμε στην εφαρμογή του χρήστη.

```
      stream = fs.createReadStream(info.path, { flags: "r", start: info.start,
end: info.end });
      stream.pipe(res);
```

Τα Streams μπορούμε να τα φανταστούμε ως μια ροή δεδομένων. Μπορούμε να διαβάζουμε, να γράφουμε ή να κάνουμε και τα δυο σε ένα stream. Συνήθως τα streams έχουν ένα buffer της τάξεως 64 * 1024 αλλά μπορούμε να το αλλάξουμε σύμφωνα με της ανάγκες μας. Στο παραπάνω παράδειγμα, δημιουργούμε ένα stream για ανάγνωση και ανακατευθύνουμε (pipe) τα δεδομένα που θα διαβάζουμε στο Stream του αντικειμένου Response. Με απλά λόγια, κάθε φορά που γεμίζει ο buffer (64 * 1024) του Stream, τα δεδομένα ανακατευθύνονται και στελνονται στην εφαρμογη του χρηστη.

Στο σενάριο όπου υπάρχουν πολλοί ταυτόχρονοι χρήστες, δημιουργούνται Streams για κάθε αρχείο και κάθε φορά που γεμίζει κάποιο buffer του Stream τότε στέλνονται στις εφαρμογές των χρηστών. Το node.js δεν μπλοκάρει πουθενά, αλλά συνεχώς εξυπηρετεί της συνδέσεις των χρηστών που έχουν διαθέσιμα δεδομένα για αποστολή χωρίς να περιμένει να ολοκληρωθεί κάποιο IO.

Αν και το παραπάνω παράδειγμα είναι απλουστευμένο σε σχέση με το υποσύστημα το οποίο

τρέχει στην ασύγχρονη πλατφόρμα είναι ένα καλό δείγμα για να διακρίνουμε τα πλεονεκτήματα της συγκεκριμένης αρχιτεκτονικής.

Ένα παραδοσιακό σύστημα όπου για κάθε σύνδεση θα δημιουργούσε ένα καινούργιο thread ή process θα είχε ως hard limit την διαθέσιμη μνήμη του συστήματος, καθώς για κάθε process/thread καταλαμβάνεται αρκετός χώρος στην μνήμη. Στην δική μας περίπτωση το μόνο hard limit είναι τα file descriptors του λειτουργικού συστήματος που αυξάνονται για κάθε σύνδεση, κάτι το οποίο μπορούμε να αλλάξουμε στον πυρήνα του Linux (ulimit -n).

Όπως έχουμε αναφέρει, το node.js τρέχει μόνο σε ένα process. Αυτό μας δίνει τη δυνατότητα να εκμεταλλευτούμε στο έπακρο ένα σύστημα με πολλούς πυρήνες. Μπορούμε να δημιουργήσουμε ένα pool από ανεξάρτητα processes τα οποία να τρέχουν σε όλους τους πυρήνες αυξάνοντας έτσι την απόδοση του συστήματος. Επίσης θα μπορούσαμε να περιορίσουμε την εκτέλεση του node.js σε συγκεκριμένο αριθμό πυρήνων έτσι ώστε οι υπόλοιποι να μπορούν να χρησιμοποιούνται αποκλειστικά από άλλα συστήματα τα οποία τρέχουν στο ίδιο μηχάνημα όπως μια βάση δεδομένων.

4.8.2 Υποσύστημα ενημερώσεως περιεχόμενου

Ένα από τα βασικά χαρακτηριστικά της ασύγχρονης πλατφόρμας, είναι να μπορεί να μεταφορτώνει αυτόματα τηλεοπτικό περιεχόμενο που ενδιαφέρει τον χρήστη αμέσως μόλις γίνει αυτό διαθέσιμο στην πλατφόρμα. Για να ενημερώνεται η εφαρμογή για το διαθέσιμο περιεχόμενο της πλατφόρμας, χρησιμοποιούμε την γνωστή και προτυποποιημένη μέθοδο ανταλλαγής ψηφιακού πληροφοριακού περιεχομένου, στηριγμένη στην πρότυπη, καθιερωμένη και ευρέως υποστηριζόμενη γλώσσα σήμανσης XML.

Ο κεντρικός διακομιστής κάνει διαθέσιμο ανά πάσα στιγμή το feed με το διαθέσιμο περιεχόμενο καθώς και επιπρόσθετες πληροφορίες όπως περιγραφή του περιεχομένου, ημερομηνία προβολής καθώς και κάποιο στιγμιότυπο του βιντεο.

Το λογισμικό ασύγχρονης τηλεόρασης το οποίο είναι εγκαταστημένο στην συσκευή Set-top-Box ανά τακτά χρονικά διαστήματα μεταφορτώνει το αρχείο xml από τον κεντρικό διακομιστή. Σε περίπτωση που βρει νέο τηλεοπτικό περιεχόμενο, αυτόματα ξεκινάει την διαδικασία μεταφόρτωσης.

Για να επιτευχθεί η όλη διαδικασία χωρίς πρόβλημα, χρειάζεται να δώσουμε προσοχή στον τρόπο με τον οποίο λειτουργεί αυτό το υποσύστημα της εφαρμογής. Έχοντας υπόψιν ότι δεν μπορούμε να εκτελέσουμε εντολές που θα μπορούσαν να μπλοκάρουν την εκτέλεση της εφαρμογής, όπως η επικοινωνία μέσω δικτύου, είναι αναγκαίο να δημιουργήσουμε ένα καινούργιο thread.

Παρακάτω παραθέτουμε κομμάτια κώδικα από τα οποία επιτελείται η λειτουργιά του υποσυστήματος ως κομμάτι της εφαρμογής. Να σημειωθεί ότι ο κώδικας είναι απλουστευμένος, έχουν αφαιρεθεί εσκεμμένα κάποιοι ελέγχει καθώς και κάποιες εντολές που κάνουν αποδέσμευση της μνήμης.

Κάθε 15 λεπτά εκτελείται η συνάρτηση *sync_shows* που είναι υπεύθυνη για την έναρξη του υποσυστήματος ενημέρωσης του περιεχομένου.

```
g_timeout_add (15*60*1000,(GSourceFunc)sync_shows,app);
```

Η συνάρτηση sync_shows θα καλεστεί από το κεντρικό thread το οποίο είναι υπεύθυνο για την διεπαφή. Είναι λοιπόν σημαντικό να μην μπλοκάρουμε την εκτέλεση της εφαρμογής σε αυτό το σημείο, διότι κάτι τέτοιο θα πάγωνε την διεπαφή.

Για αυτό τον λόγο χρησιμοποιώντας την συνάρτηση της βιβλιοθήκης Glib *g_thread_create δημιουργούμε* ένα καινούργιο thread το οποίο από την στιγμή που θα είναι ανεξάρτητο από το κυρίως process είμαστε σίγουροι ότι δεν θα δημιουργήσει κάποιο κόλλημα στην διεπαφή.

Είναι σημαντικό να μην δημιουργήσουμε δυο threads του ίδιου υποσυστήματος, καθώς κάτι τέτοιο θα είχε δυσάρεστα αποτελέσματα. Γι' αυτό το λόγο, δημιουργούμε μια στατική μεταβλητή οπού θα κρατήσουμε το id του thread. Μονό όταν δεν υπάρχει κάποιο άλλο thread του υποσυστήματος θα ξεκινάμε ένα καινούργιο thread.

```
static GThread *thread = NULL;
gboolean sync_shows(Bango * app){
     if(!thread)
            thread = g_thread_create((GThreadFunc)update_sync, app, TRUE, NULL);
     return TRUE;
}
```

Η συνάρτηση *update_sync* θα εκτελεστεί σε ένα καινούργιο thread. Ότι και εάν εκτελεστεί σε αυτήν την συνάρτηση, είμαστε σίγουροι ότι δεν θα επηρεάσει την απόδοση της μητρικής διεργασίας και κατά συνέπεια της διεπαφής. Λόγο όμως ότι θα threads μπορούν να έχουν πρόσβαση σε κοινή μνήμη με την μητρική διεργασία, είναι σημαντικό να μην αλλάξουμε μεταβλητές ή να καλέσουμε συναρτήσεις που έχουν σχέση με την διεπαφή. Στο συγκεκριμένο υποσύστημα δεν έχουμε τέτοιο πρόβλημα καθώς δεν υπάρχει κάποια σχέση με την διεπαφή.

Η λειτουργιά της συνάρτησης αυτής είναι να μεταφορτώνει για κάθε τηλεοπτικό πρόγραμμα που έχει επιλέξει ο χρήστης το αρχείο xml το οποίο περιγράφει τα διαθέσιμα τηλεοπτικά επεισόδια για το συγκεκριμένο πρόγραμμα.

Παρατηρούμε ότι όταν τελειώσει η εκτέλεση της συνάρτησης, που σημαίνει και το τέλος ζωής του thread, η στατική μεταβλητή γίνεται Null έτσι ώστε να δηλωθεί με αυτόν τον τρόπο ότι δεν τρέχει κάποιο thread του υποσυστήματος.

```
void update_sync(gpointer data){
     Bango *app = (Bango *)data;
     GKeyFile *keyfile;
     mrss_t *rss_data;
     gchar **groups;
     episode_info info;

     // **** //
     while(// All registered episodes //){
            if(rss_data = download_rss_data (url)){
                  int                              season                          =
g_key_file_get_integer(keyfile,groups[i],"Season",NULL);
                  int                              episode                         =
g_key_file_get_integer(keyfile,groups[i],"Episode",NULL)l
                  info = episode_downloader(rss_data,season,episode,app);

                  mrss_free(rss_data);
            }
     }
     // **** //

     thread = NULL;
     g_thread_exit(0);
}
```

Η συνάρτηση *download_rss_data* μεταφορτώνει το περιεχόμενο κάποιου url, στην συγκεκριμένη περίπτωση αρχεία xml, και τα επιστρέφει ως δομές κατάλληλες για να διαβάσουμε την δομή του αρχείου RSS. Χρησιμοποιείται η mRSS η οποία είναι μια βιβλιοθήκη για την ανάγνωση RSS καθώς και η βιβλιοθήκη libCurl για την μεταφόρτωση δεδομένων από το διαδίκτυο. Αξίζει να σημειωθεί ότι η βιβλιοθήκη mRSS κάνει χρήση της libCurl.

```
mrss_t * download_rss_data(char * url_un){
     gchar * url = g_uri_escape_string(url_un,":/?=&%",TRUE);

     mrss_error_t error;
     CURLcode curl_code;
     mrss_t *data;
     error = mrss_parse_url_with_options_and_error(url,&data,NULL,&curl_code);
     if (error)
            return NULL;
     return data;
}
```

Η συνάρτηση *episode_downloader* ελέγχει για την ύπαρξη νέου τηλεοπτικού προγράμματος και ξεκινάει την διαδικασία μεταφορτώσεις. Σε περίπτωση που βρει τηλεοπτικό περιεχόμενο το οποίο ο αριθμός επεισοδίου είναι μεγαλύτερος από αυτόν που έχει αποθηκευμένο το σύστημα, το θεωρεί ως καινούργιο περιεχόμενο και ξεκινάει την διαδικασία για την μεταφόρτωση του περιεχομένου. Να σημειωθεί ότι η μεταφόρτωση του περιεχομένου αφορά άλλο υποσύστημα.

```
episode_info episode_downloader(mrss_t *rss_data,int season, int episode, Bango
*app){
     mrss_item_t *item;
     episode_info info;
     episode_info max;

     max.season = season;
     max.episode = episode;

     item = rss_data->item;

     while(item){
            info = get_episode_info(item->description);

            if((season<=info.season) && (episode<info.episode)){
                   CURL *curl;
                   CURLcode res;
                   FILE *file;

                   // *** //

                   curl = curl_easy_init();
                   if(curl) {
                          curl_easy_setopt(curl, CURLOPT_URL, url);
                          curl_easy_setopt(curl, CURLOPT_WRITEDATA , file);
                          res = curl_easy_perform(curl);
                   }
                   fclose(file);

                   if(res==0){
```

Σε περίπτωση που βρεθεί νέο περιεχόμενο, καλείται η συνάρτηση *add_torrent_file* η οποία ενημερώνει το υποσύστημα το οποίο είναι υπεύθυνο να ξεκινήσει την διαδικασία μεταφόρτωσης.

```
                          add_torrent_file(filename->str, app);
                   }
            }
            item = item->next;
     }
     return max;
```

```
}
```

4.8.3 Υποσύστημα μεταφόρτωσης

Το υποσύστημα μεταφόρτωσης είναι υπεύθυνο για την μεταφόρτωση του τηλεοπτικού περιεχομένου με αποτελεσματικό και αξιόπιστο τρόπο. Για την αποδοτικότερη χρήση του διαθέσιμου εύρους ζώνης γίνεται η χρήση του πρωτοκόλλου BitTorrent για την μεταφόρτωση του περιεχομένου.

Το BitTorrent είναι ένα πρωτόκολλο μεταφοράς μεγάλου όγκου δεδομένων το οποίο βασίζεται στην συμμετοχή των χρηστών για τον διαμοιρασμό του περιεχομένου. Τα αρχεία *.torrent είναι* αρχεία τα οποία περιέχουν metadata σχετικά με το περιεχόμενο, διευθύνσεις για τους trackers κτλ.

Κάθε τηλεοπτικό περιεχόμενο στην πλατφόρμα ασύγχρονης τηλεόρασης έχει και ένα αντίστοιχο αρχείο bittorrent. Όταν ο χρήστης επιλέξει να παρακολουθήσει κάποιο τηλεοπτικό επεισόδιο, τότε το αρχείο *.torrent* το οποίο περιέχει της κατάλληλες πληροφορίες μεταφορτώνεται από το διαδίκτυο και χρησιμοποιείται από την βιβλιοθήκη libtorrent για να ξεκινήσει η μεταφόρτωση του τηλεοπτικού περιεχομένου.

Η βιβλιοθήκη libtorrent όπως έχει περιγραφεί και σε προηγούμενο κεφάλαιο είναι μια ανοικτού κώδικα υλοποίηση του πρωτοκόλλου bittorrent. Από τεχνικής πλευράς, είναι μια πολύ ευέλικτη βιβλιοθήκη που χρησιμοποιεί πολύ καλά οργανωμένες δομές της C++ και της βιβλιοθήκης Boost που είναι επέκταση της C++.

Αν και η επιλογή της βιβλιοθήκης libtorrent θεωρείται η καλύτερη δυνατή, λόγο του αρθρωτού σχεδιασμού του συστήματος, θα μπορούσαμε είτε να αλλάξουμε τελείως τον τρόπο που χρησιμοποιεί η πλατφόρμα για την μεταφόρτωση του τηλεοπτικού περιεχομένου, είτε να προσθέσουμε νέες τεχνικές. Παραθέτουμε κομμάτια κώδικα τα οποία χρησιμοποιούνται στο υποσύστημα μεταφορτώσεις της ασύγχρονης πλατφόρμας.

Σε όλα τα κομμάτια κώδικα που παραθέτονται παρακάτω χρησιμοποιείται το *namespace std* και *libtorrent*. Η συνάρτηση *add_torrent_file* καλείται για να προσθέσει ένα αρχείο *.torrent* στο session της libtorrent.

```
using namespace std;
using namespace libtorrent;

libtorrent::torrent_handle add_torrent_file(char *file, Bango * app) {
     torrent_handle handle;

     std::ifstream in(file, std::ios_base::binary);
     in.unsetf(std::ios_base::skipws);

     try{
            GString *resume;
            GString *hashname;
            GString *torrent_new_name;
            gboolean paused;
            gchar * save_path;
            gchar *hash;

            entry       e       =       bdecode(std::istream_iterator<char>(in),
istream_iterator<char>());

            hash = g_filename_display_basename(file);

            resume = get_configuration_folder(RESUME_FOLDER);
```

```
            resume = g_string_append(resume,G_DIR_SEPARATOR_S);
            resume = g_string_append(resume,hash);
            resume = g_string_append(resume,".resume");

            entry resume_data;
```

Πριν την προσθήκη του .torrent αρχείου στο session της libtorrent, γίνεται αναζήτηση τοπικά για την ύπαρξη ενός αρχείου που περιγράφει την κατάσταση της μεταφόρτωσης, το αρχείο αυτό αν και δεν είναι απαραίτητο, βελτιώνει την απόδοση του συστήματος όπως θα εξηγηθεί παρακάτω. Η μέθοδος add_torrent του αντικειμένου session παίρνει ως ορίσματα ένα αντικείμενο torrent_data το οποίο περιγράφει το αρχείο .torrent, ένα string που αναφέρεται στο path όπου θα αποθηκευτούν τα δεδομένα, το αρχείο περιγραφής της κατάστασης εάν υπάρχει και επιστρέφει ένα αντικείμενο torrent_handle που είναι μια αναφορά στο torrent που εισάγαμε στο session.

Τα αντικείμενα torrent_handle αποθηκεύονται σε μια λίστα η οποία χρησιμοποιείται για να γνωρίζουμε ανά πάσα στιγμή τα torrents τα οποία είναι ενεργά στο σύστημα αλλά και για να τα παρουσιάσουμε στον χρήστη ως μέρος της διεπαφής.

```
            try{
                  boost::filesystem::ifstream           resume_file(resume->str,
ios_base::binary);
                  resume_file.unsetf(ios_base::skipws);
                  resume_data   =   bdecode(istream_iterator<char>(resume_file),
istream_iterator<char>());
            }catch (invalid_encoding&) {}

            paused                  =                 g_key_file_get_boolean(app-
>list_keyfile,hash,"Paused",NULL);
            save_path                =                g_key_file_get_string(app-
>list_keyfile,hash,"Save_Path",NULL);
            if(save_path == NULL)
                  save_path = app->save_location_incompleted;

            boost::intrusive_ptr<torrent_info> info(new torrent_info(e));

            handle = app->session->add_torrent(info, save_path,resume_data);

            app->torrent_list.push_back(handle);

            if(paused)
                  handle.pause();

            torrent_new_name = get_configuration_folder(TORRENT_FOLDER);
            torrent_new_name                                                    =
g_string_append(torrent_new_name,G_DIR_SEPARATOR_S);
            hashname = hash_to_string(handle.info_hash());
            torrent_new_name = g_string_append(torrent_new_name,hashname->str);

            if(strcmp(file,torrent_new_name->str))
                  boost::filesystem::copy_file(file,torrent_new_name->str);

      }
      catch(std::exception& e){
      }

      in.close();
```

```
	if(handle.is_valid()){

		DownloadPack * pack = g_slice_new(DownloadPack);
		pack->handle = g_slice_new(torrent_handle);
		*(pack->handle) = handle;
		pack->app = app;

		clutter_threads_add_idle ((GSourceFunc)add_new_download,pack);
	}

	return handle;
}
```

Η συνάρτηση *remove_torrent χρησιμοποιείται* για να αφαιρέσει ενα torrent από το session της libtorrent.

```
void remove_torrent(Bango *app,libtorrent::torrent_handle handle){
	if(handle.is_valid()){
		app->session->remove_torrent(handle);
	}
}
```

Όταν προστίθεται ένα καινούργιο torrent στο session της libtorrent, θα πρέπει να διαβαστούν όλα τα δεδομένα τα οποία υπάρχουν στον δίσκο και να υπολογιστεί κατά ποιο ποσοστό έχει μεταφορτωθεί το περιεχόμενο και ποια κομμάτια των αρχείων είναι ολοκληρωμένα και ποια όχι. Η όλη διαδικασία είναι αρκετά χρονοβόρα, και εάν θεωρήσουμε ότι εισάγουμε 10 torrent αρχεία τα οποία γνωρίζουμε ότι είναι πλήρως μεταφορτωμένα, τότε άσκοπα σπαταλούμε και χρόνο αλλά και πόρους του συστήματος. Ο υπολογισμός γίνεται με κρυπτογραφικές συναρτήσεις και συγκρίσεις των hash που έχουν υπολογιστεί από το σύστημα με τα hash τα οποία υπάρχουν στο *.torrent* αρχείο.

Για να αποφύγουμε όλη αυτήν την διαδικασία μπορούμε να αποθηκεύσουμε τα δεδομένα τα οποία περιγράφουν την κατάσταση της μεταφορτώσεις σε ένα αρχείο. Έπειτα όταν προσθέσουμε το torrent στο session της libtorrent αντί να υπολογίσει από την αρχή την κατάσταση της μεταφόρτωσης, χρησιμοποιεί της πληροφορίες που είχαμε αποθηκεύσει σε προηγούμενο στάδιο. Συνήθως αποθηκεύουμε τα δεδομένα αυτά ανά διαστήματα, όταν το torrent γίνει pause ή όταν η εφαρμογή πρόκειται να τερματιστεί.

Η συνάρτηση *torrent_save_resume_data* παίρνει σαν όρισμα ένα *torrent_handle* το οποίο αντιπροσωπεύει ένα torrent στο session της libtorrent και αποθηκεύει τα δεδομένα που περιγράφουν την κατάσταση της μεταφόρτωσης σε κάποιο αρχείο στον δίσκο.

```
void torrent_save_resume_data(libtorrent::torrent_handle handle){
	libtorrent::entry e;
	GString * filename;
	GString * hashname;
	bool paused;

	// *** //

	ofstream out(filename->str,ios_base::binary);
	out.unsetf(ios_base::skipws);

	paused = handle.is_paused();
	handle.pause();

	e = handle.write_resume_data();
```

```
	bencode(ostream_iterator < char >(out), e);

	out.close();

	if(!paused)
		handle.resume();
}
```

Η συνάρτηση *resume_torrents* χρησιμοποιείται για να εισάγει όλα τα torrent αρχεία τα οποία υπάρχουν τοπικά στο session της libtorrent και εκτελείται για να επαναφέρει το σύστημα έπειτα από την εκκίνηση του στην προηγούμενη κατάσταση του.

```
void resume_torrents(Bango * app){
	GString * temp;

	// *** //

	while(// For all saved torrent files //){
		// *** //
		if(!g_file_test(temp->str,G_FILE_TEST_IS_DIR)){
			add_torrent_file(temp->str,app);
		}
	}
}
```

Η δημιουργία ενός session της libtorrent περιλαμβάνει κάποιες ρυθμίσεις οι οποίες θα επηρεάσουν όλες της μεταφορτώσεις καθώς και την επικοινωνία με τους trackers. Κάθε torrent client χρησιμοποιεί κάποιο fingerprint για να αναγνωρίσει ο tracker με ποιο πρόγραμμα επικοινωνεί. Η ασύγχρονη πλατφόρμα χρησιμοποιεί των κωδικό BN ως fingerprint της.

Επίσης σημαντικός είναι ο παράγοντας του bandwidth που γίνεται διαθέσιμο για της μεταφορτώσεις. Επειδή το πρωτόκολλο μεταφοράς bittorrent κάνει χρήση πολλών πηγών για την μεταφόρτωση των δεδομένων, χρησιμοποιεί πολλές ανοικτές συνδέσεις. Το ανεξέλεγκτο άνοιγμα συνδέσεων θα μπορούσε να δημιουργήσει συμφόρηση στο τοπικό δίκτυο. Για το λόγο αυτό γίνεται χρήση περιορισμού στην ταχύτητα ανεβάσματος και κατεβάσματος των δεδομένων.

Η βιβλιοθήκη libtorrent υποστηρίζει extensions τα οποία αν και δεν είναι μέρος του επίσημου πρωτοκόλλου έχουν υλοποιηθεί αρχικά από γνωστούς torrent clients και κατ' επέκταση προτυποποιήθηκαν και χρησιμοποιήθηκαν και από άλλους torrent clients. Ένα απο τα extension που χρησιμοποιούμε είναι το Peer Exchange το οποίο επεκτείνει της δυνατότητες επικοινωνίας μεταξύ των peers σε ένα δίκτυο bittorrent.

Επιτρέπει μια ομάδα χρηστών, που συνεργάζονται για να μοιράζονται ένα συγκεκριμένο αρχείο να το κάνουν πιο γρήγορα και αποτελεσματικά. Στον αρχικό σχεδιασμό του πρωτοκόλλου BitTorrent, οι χρήστες σε μια ομάδα διαμοιρασμού αρχείων (γνωστή ως "swarm") βασιζονται σε έναν κεντρικό server που ονομάζεται tracker για να βρουν ο ένας τον άλλο και να διατηρήσουν την ομάδα(swarm) λειτουργική. Το Peer Exchange μειώνει σημαντικά την εξάρτηση τους κεντρικούς διακομιστές(trackers), επιτρέποντας σε κάθε χρήστη να ενημερώσετε άμεσα τους άλλους χρήστες της ομάδας(swarm) ως προς το ποιοι είναι οι χρήστες της ομάδας. Με τη μείωση της εξάρτησης από έναν κεντρικό διακομιστή (tracker), αυξάνεται η ταχύτητα, η αποτελεσματικότητα και η ευρωστία του πρωτοκόλλου BitTorrent.

```
libtorrent::session * session;
session = new libtorrent::session(libtorrent::fingerprint("BN", 0, 1, 0, 0));
```

```
session->listen_on(make_pair(3000, 3010));
session->set_download_rate_limit(500 * 1000);
session->set_upload_rate_limit(30 * 1000);

session->add_extension(&libtorrent::create_ut_pex_plugin);

if(session->is_listening()){
       g_print("\nListening in %d port!\n",session->listen_port());
}else{
       g_print("\nNAT PROBLEM!!");
}
```

4.8.4 Threads και Αμοιβαίος Αποκλεισμός

Η διεπαφή μιας εφαρμογής είναι το μέσο όπου έρχεται σε επαφή ο χρήστης με τον υπολογιστή. Είναι σημαντικό η λειτουργιά της διεπαφής να είναι πολύ προσεκτικά σχεδιασμένη έτσι ώστε να μην επηρεάζει την εμπειρία χρήσης της εφαρμογής.

Ένα από τα προβλήματα τα οποία θέλαμε να αποφύγουμε ήταν το *"πάγωμα"* της διεπαφής. Ολόκληρο το υποσύστημα το οποίο διαχειρίζεται την διεπαφή τρέχει σε μια μόνο διεργασία. Χρησιμοποιούμε την βιβλιοθήκη Clutter για την δημιουργία της διεπαφής σε OpenGL, που μεταξύ άλλων βασίζεται στην βιβλιοθήκη Glib. Το Clutter λειτουργεί με την έννοια που έχουμε αναφέρει και στα προηγούμενα κεφάλαια του event-loop. Αυτό σημαίνει ότι υπάρχει μια εσωτερική ουρά στην οποία καταχωρούνται συμβάντα ή εντολές που θα εκτελεστούν στο μέλλον. Για να λειτουργήσει αυτό ικανοποιητικά, προϋποθέτει ότι η εκτέλεση των εντολών πρέπει να γίνει πολύ γρήγορα, έτσι ώστε η διεπαφή να ανταποκρίνεται στην αλληλεπίδραση του χρήστη.

Σε περίπτωση που πρέπει να εκτελεστεί κώδικας ο οποίος δεν γνωρίζουμε πόσο χρόνο θα κάνει για να ολοκληρωθεί η εκτέλεση του, όπως για παράδειγμα το κατέβασμα ενός αρχείου από το διαδίκτυο ή η ανάγνωση από μια βάση δεδομένων, θα πρέπει να χρησιμοποιήσουμε threads. Σε διαφορετική περίπτωση, η διεπαφή θα κολλάει, δηλαδή δεν θα έχει καμιά αλληλεπίδραση με τον χρήστη, μέχρι να ολοκληρωθεί η εκτέλεση του χρονοβόρου κώδικα και το event-loop να πάρει τον έλεγχο και να εκτελέσει τα υπόλοιπα συμβάντα.

Η χρήση threads όμως εισάγει νέα προβλήματα τα οποία θα πρέπει να τα αντιμετωπίσουμε εξ' αρχής. Επειδή τα threads μοιράζονται το ίδιο memory space, θα μπορούσαμε να αλλάξουμε μεταβλητές του process ή να καλέσουμε συναρτήσεις όπου επηρεάζουν την διεπαφή μέσα από το thread. Κάτι τέτοιο πρέπει να γίνει με προσοχή και συγκεκριμένη τεχνική, διαφορετικα η η εφαρμογη θα σταματούσε να λειτουργεί σε ανύποπτες στιγμές. Γενικότερα ο προγραμματισμός με πολλά threads αυξάνει την δυσκολία και θα πρέπει να εφαρμοστούν τεχνικές οι οποίες θα προστατεύσουν την εφαρμογή από προβλήματα.

Ο αμοιβαίος αποκλεισμός(mutual exclusion) στην επιστήμη των υπολογιστών, αναφέρεται στο πρόβλημα εξασφάλισης ότι δεν υπάρχουν δυο διεργασίες ή νήματα(threads) οι οποίες να βρίσκονται ταυτόχρονα στο κρίσιμο σημείο τους(critical section). Κρίσιμο σημείο αναφέρεται μια χρονική περίοδο όπου μια διαδικασία αποκτά πρόσβαση σε έναν κοινόχρηστο πόρο, όπως για παράδειγμα η κοινόχρηστη μνήμη. Για να προστατεύσουμε την εφαρμογή μας από τέτοιου είδους προβλήματα, χρησιμοποιήσαμε το Gmutex της βιβλιοθήκης Glib όπου υλοποιείται μια τεχνική αμοιβαίου αποκλεισμού για τα δεδομένα που διαχειρίζεται η εφαρμογή μας.

Υπάρχουν όμως περιπτώσεις όπου θα χρειαστεί κάποιο thread να επηρεάσει την διεπαφή. Πάλι για λόγους αμοιβαίου αποκλεισμού, δεν μπορούμε απευθείας να αλλάξουμε την διεπαφή. Θα πρέπει πρώτα να ζητήσουμε ένα Lock του Clutter, δηλαδή να διασφαλίσουμε ότι κανένα άλλο thread

δεν θα αλλάξει τις εσωτερικές δομές που αφορούν την διεπαφή. Παρακάτω περιγράφονται οι τρόποι που χρησιμοποιήθηκαν για τα threads και τον αμοιβαίο αποκλεισμό.

Κατά την έναρξη της εφαρμογής, αρχικοποιούμε τα υποσυστήματα καθώς και κάποιες δομές που χρειαζόμαστε κατά την λειτουργία της εφαρμογής. Είναι σημαντικό κατά την αρχικοποίηση του Clutter να δηλωθεί ότι θα χρησιμοποιηθούν threads έτσι ώστε η βιβλιοθήκη να προστατεύει εσωτερικά τις δομές της για την αποφυγή προβλημάτων. Γίνεται δυναμική δέσμευση μνήμης για την δομή που θα αντιπροσωπεύει την εφαρμογή καθώς και αρχικοποίηση ενός συστήματος αμοιβαίου αποκλεισμού.

```
Bango * initialize(int argc, char **argv){
     Bango *app;

     // *** //

     app = g_slice_new(Bango);
```

Για να μπορεί το Clutter να χρησιμοποιηθεί από πολλά threads, είναι σημαντικό να γίνει αρχικοποίηση του συστήματος διαχείρισης νημάτων. Ο μηχανισμός αυτός δίνει τη δυνατότητα στον προγραμματιστή να να χρησιμοποιήσει το Clutter από πολλά threads χρησιμοποιώντας τις συναρτήσεις *clutter_threads_enter()* και *clutter_threads_leave()* για να περικλείσει τα κρίσιμα σημεία της εφαρμογής.

```
     if (!g_thread_supported ()) g_thread_init (NULL);
     ClutterInitError init = clutter_init (&argc, &argv);
     clutter_threads_init();
     gst_init (&argc, &argv);
```

Γίνεται χρήση του Gmutex που είναι ένας μηχανισμός της Glib σχετικά με τον αμοιβαίο αποκλεισμό. Μπορεί να χρησιμοποιηθεί για την προστασία των δεδομένων από κοινή πρόσβαση, όπως θα μπορούσε να συμβεί μεταξύ των threads.

```
     app->mutex = g_mutex_new();

     // *** //
     return app;
}
```

Η συνάρτηση *download_thumbnails* εκτελείται από την μητρική διεργασία έτσι ώστε να γίνει μεταφόρτωση τοπικά thumbnails τα οποία χρειάζονται από την διεπαφή για την παρουσίαση των διαθέσιμων τηλεοπτικών προγραμμάτων. Η διαδικασία της μεταφορτώσεις όμως, είναι μια διαδικασία που δεν γνωρίζουμε την χρονική διάρκεια που χρειάζεται για να ολοκληρωθεί και είναι σίγουρο ότι εκτελώντας αυτήν την διαδικασία από την μητρική διεργασία, η διεπαφή θα πάγωνε.

Είναι αναγκαίο λοιπόν, να δημιουργηθεί ένα καινούργιο thread το οποίο θα μεταφορτώσει της εικόνες από το διαδίκτυο και όταν ολοκληρωθεί η εργασία να ενημέρωση την εφαρμογή για της διαθέσιμες εικόνες.

```
gpointer download_thumbnails(Bango *app){
     static int thread = 0;
     sqlite3_stmt *stmt;
     gchar * show;
     int rv;

     sqlite3_prepare_v2(app->db,"SELECT  id,name  FROM  shows  WHERE  thumb='';",-
```

```
1,&stmt,NULL);
     sqlite3_step(stmt);

     show = (gchar*)sqlite3_column_text(stmt,1);

     ThumbnailPack * pack = g_slice_new(ThumbnailPack);

     pack->id = sqlite3_column_int(stmt,0);
     pack->show = g_strdup((const gchar*)sqlite3_column_text(stmt,1));
     pack->app = app;
```

Η δημιουργία ενός thread είναι κάτι το οποίο υλοποιείται στο επίπεδο του λειτουργικού συστήματος. Χρησιμοποιώντας την βιβλιοθήκη Glib μπορούμε με μεγάλη ευκολία να δημιουργήσουμε ένα καινούργιο thread δηλώνοντας παραμέτρους όπως priority, stack_size κτλ. Η συνάρτηση *thumbnail_downloader* θα εκτελεστεί σε ένα καινούργιο thread με χαμηλή προτεραιότητα.

```
     g_thread_create_full((GThreadFunc)
thumbnail_downloader,pack,0,FALSE,TRUE,G_THREAD_PRIORITY_LOW,NULL);

     sqlite3_finalize(stmt);
}
```

Η συνάρτηση *thumbnail_downloader* θα εκτελεστεί σε ένα καινούργιο thread το οποίο θα έχει κοινή μνήμη με την μητρική διεργασία. Πλέον από την στιγμή που η λειτουργιά του thread έχει ξεφύγει από τον έλεγχο του event-loop δεν μας απασχολεί το πόσο χρονικό διάστημα θα χρειαστεί μέχρι να ολοκληρώσει την εργασία του καθώς δεν θα μπλοκάρει την διεπαφή.

```
gpointer thumbnail_downloader(ThumbnailPack *pack){
     CURL *curl;
     CURLcode res;
     FILE *file;
     mrss_item_t *item;
     mrss_t * rss;
     GdkPixbuf *pixbuf;

     GString *url = g_string_new("http://tracker/bango/rss/shows.php?show=");
     url = g_string_append(url,pack->show);
     rss = download_rss_data(url->str);

     item = rss->item;
     if(item){
           GString                 *                 filename                 =
path_convert(get_configuration_folder(THUMB_FOLDER));
           filename = g_string_append(filename,item->title);

           gchar   *   uri   =   g_uri_escape_string(item->enclosure_url,":/=?&
%",TRUE);

           file = fopen(filename->str,"w");

           curl = curl_easy_init();
           if(curl) {
                 curl_easy_setopt(curl, CURLOPT_URL, uri);
                 curl_easy_setopt(curl, CURLOPT_WRITEDATA , file);
                 res = curl_easy_perform(curl);
           }
```

```
            pixbuf = gdk_pixbuf_new_from_file (filename->str,NULL);
            pack->pixbuf = pixbuf;
```

Η διαδικασία της μεταφορτώσεις σε αυτό το σημείο έχει ολοκληρωθεί. Για να γίνουν αλλαγές στην διεπαφή, δηλώνουμε το κρίσιμο σημείο της εφαρμογής. Η συνάρτηση clutter_threads θα επιστρέψει μόνο όταν έχει πάρει το Clutter Lock έτσι ώστε να γίνουν οι αλλαγές στην διεπαφή με ασφάλεια.

```
            clutter_threads_enter();
            set_thumb_to_show_pack(pack);
            models_update(pack->app);
            clutter_threads_leave();
      }
      g_thread_exit(0);
}
```

Η συνάρτηση *update_episode_list* εκτελείται σε ένα ξεχωριστό thread. Αν και δεν αλλάζει άμεσα την διεπαφή, έχει πρόσβαση σε δομές οι οποίες είναι σημαντικό να προστατευτούν έτσι ώστε να μην προσπελαστούν ταυτόχρονα από διαφορετικά threads. Το παρακάτω κομμάτι κώδικα δείχνει πως χρησιμοποιούμε αμοιβαίο αποκλεισμό σε ολόκληρη την εφαρμογή.

```
gpointer update_episode_list(void * user_data){
      SubscribePack * pack = (SubscribePack*) user_data;
      Bango * app = pack->app;
      // *** //
      GString  *  url  =  g_string_new  ("http://tracker/bango/rss/episodes.php?
show=");
      url = g_string_append (url,pack->title);

      rss = download_rss_data(url->str);
      item = rss->item;
      while(item){
            if(app->state == EPISODE_LISTER){
```

Χρησιμοποιώντας την συνάρτηση g_mutex_lock παίρνουμε ένα αποκλειστικό lock έτσι ώστε να συνεχίσουμε την εκτέλεση του προγράμματος με την διαβεβαίωση ότι κανένα άλλο thread της εφαρμογής δεν θα εκτελέσει κάποιο κρίσιμο σημείο.

```
                  g_mutex_lock(app->mutex);
                  SubscribePack *pack = g_slice_new(SubscribePack);
                  episode_info info = get_episode_info(item->description);

                  GString * title = g_string_new(NULL);
                  g_string_printf(title,"(S%02d.E%02d)
%s",info.season,info.episode,item->title);

                  pack->title        =        g_string_chunk_insert        (app-
>model_episode_list_strings,title->str);
                  pack->text        =        g_string_chunk_insert        (app-
>model_episode_list_strings,item->enclosure_url); // subtitle url
                  pack->url         =        g_string_chunk_insert        (app-
>model_episode_list_strings,item->link);

                  app->episode_list = g_list_append(app->episode_list, pack);

                  g_mutex_unlock(app->mutex);
```

Όταν τελειώσει η εκτέλεση του κρίσιμου σημείου, απελευθερώνουμε το lock. Ο παραπάνω κώδικας ήταν σημαντικό να δηλωθεί ως κρίσιμο σημείο διότι μια κεντρική δομή της εφαρμογής άλλαξε περιεχόμενα.

```
            }
            item = item->next;
        }
        g_thread_exit(0);
    }
```

Σε ολόκληρη την εφαρμογή μπορεί να τρέξουν τα παρακάτω threads

- update_sync – ανανέωση των διαθέσιμων τηλεοπτικών προγραμμάτων μέσω ροής RSS
- time_monitor – χρησιμοποιείται στην δημιουργία thumbnails των βίντεο
- network_server – ανεξάρτητος tcp network server που χρησιμοποιείται κυρίως για debugging
- thumbnail_downloader – μεταφορτώνει thumbnails από το διαδίκτυο
- update_episode_list – προσθέτει κάποια μεταφόρτωση στην λίστα των διαθέσιμων επεισοδίων
- update_subscribe – διαχειρίζεται της εκπομπές στις οποίες έχει γραφτεί ο χρήστης
- download_torrent_file – μεταφόρτωση ενός .torrent αρχείου από το διαδίκτυο.

Ένας γενικός κανόνας για την δημιουργία καινούργιου thread για την εκτέλεση μιας εργασίας είναι ο χρόνος εκτέλεσης, δηλαδή εάν θα μπλοκάρει την διεπαφή και το event-loop. Γίνεται κατανοητό ότι οτιδήποτε έχει σχέση με IO ή προκαλεί μεγάλο φόρτο στην CPU, είναι καλό να εκτελείται ως thread.

4.8.5 Διαχείριση Μνήμης

Ένα από τα κρισιμότερα σημεία της εφαρμογής είναι η διαχείριση μνήμης. Η βασική απαίτηση όσον άφορα την διαχείριση μνήμης είναι να καταχωρούνται δυναμικά τμήματα της μνήμης για την εφαρμογή κατόπιν αιτήσεώς της και η ελευθέρωση της για επαναχρησιμοποίηση, όταν δεν χρειάζονται πλέον.

Υπάρχουν πολλές τεχνικές για την διαχείριση μνήμης. Δυο από αυτές είναι μέσω Garbage Collector και μέσω Manual memory management. Η τεχνική του Garbage Collector χρησιμοποιείται από γλώσσες όπως η Java και η C# όπου εκτελούνται μέσα σε Virtual Machine. Υπάρχουν και υλοποιήσεις για γλώσσες όπως η C και C++ όπου υλοποιούνται μέσω της τεχνικής Reference Counting. Η τεχνική του Manual memory management αναφέρεται στην διαχείριση της μνήμης από τον ίδιο τον προγραμματιστή με εντολές όπου δεσμεύουν και αποδεσμεύουν κομμάτια της μνήμης. Η εφαρμογή χρησιμοποιεί και τους δυο τρόπους για την διαχείριση μνήμης.

Η διεπαφή είναι γραμμένη χρησιμοποιώντας το Clutter το οποίο βασίζεται στην βιβλιοθήκη Gobject. Η βιβλιοθήκη GObject υλοποιεί αντικειμενοστρέφεια στην γλώσσα C. Τα αντικείμενα δημιουργούνται δυναμικά και διαχειρίζονται μέσω της τεχνικής reference count.

Το reference counting με απλά λόγια είναι μια τεχνική για την αποθήκευση του αριθμού των αναφορών που έχει ένα αντικείμενο. Όταν το ο αριθμός αναφορών για το αντικείμενο γίνει μηδέν, τότε θεωρούμε ότι μπορεί να σβηστεί το αντικείμενο από την μνήμη.

Ένα απλό παράδειγμα είναι η δημιουργία ενός αντικειμένου το οποίο δίνεται σε δυο διαφορετικά υποσυστήματα. Ο αριθμός αναφοράς του αντικειμένου είναι 2, καθώς χρησιμοποιείται

από τα δυο υποσυστήματα. Όταν πλέον δεν χρειάζεται το αντικείμενο από το πρώτο υποσύστημα, μειώνεται ο αριθμός αναφοράς κατά μια μονάδα. Όταν συμβεί το ίδιο και από το δεύτερο υποσύστημα, τότε ο αριθμός αναφοράς γίνεται μηδέν και θεωρούμε ότι πλέον το αντικείμενο δεν χρησιμοποιείται και μπορούμε να το αποδεσμεύσουμε από την μνήμη.

Στην τεχνική του manual memory management ο προγραμματιστής δηλώνει την δημιουργία του αντικειμένου καθώς και το πότε μπορεί να καταστραφεί. Όποια τεχνική και εάν ακολουθηθεί, είναι πολύ σημαντικό να γίνουν ενέργειες έτσι ώστε να αποφευχθεί το λεγόμενο memory leaking.

Memory leak είναι όταν μια εφαρμογή δεσμεύει μνήμη χωρίς να μπορεί να την απελευθερώσει στο λειτουργικό σύστημα όταν πλέον δεν χρειάζεται. Αυτό μπορεί να συμβεί και στις δυο τεχνικές. Στην πρώτη θα μπορούσε με κάποιον τρόπο να αυξηθεί ο αριθμός αναφοράς και να μην γίνει μηδέν ποτέ, ακόμα και εάν το αντικείμενο δεν χρειάζεται πλέον. Στην δεύτερη περίπτωση, ο προγραμματιστής μπορεί να δεσμεύει μνήμη χωρίς να την απελευθερώνει όταν πλέον αυτή δεν χρειάζεται.

Όπως και εάν έχει προκληθεί το memory leak, τα συμπτώματα είναι ίδια. Η εφαρμογή συνεχώς καταναλώνει αδικαιολόγητα όλο και μεγαλύτερα ποσά μνήμης στην διάρκεια της ζωής της. Κάτι τέτοιο επιβραδύνει την απόδοση του συστήματος και θεωρείται ως κακής ποιότητας λογισμικό. Να σημειωθεί ότι τα memory leaks πολλές φορές είναι δύσκολο να αναγνωριστούν και να διορθωθούν, είναι μια συνεχή μάχη που πρέπει να δώσει ο προγραμματιστής για την αποφυγή τους.

Σε όλη την εφαρμογή έχουμε χρησιμοποιήσει της δυο τεχνικές που αναφέραμε για την διαχείριση μνήμης. Παρακάτω περιγράφονται κομμάτια κώδικα τα οποία δεσμεύουν μνήμη, απελευθερώνουν και μειώνουν τον αριθμό αναφοράς.

Σε όλη την διάρκεια ζωής της εφαρμογής γίνεται χρήση μιας δομής (*Bango*) στην οποία αποθηκεύονται όλα τα δεδομένα τα οποία χρειάζεται να έχουμε πρόσβαση και είναι διαθέσιμη σε όλα τα υποσυστήματα της εφαρμογής. Η δομή αυτή δεσμεύεται στην μνήμη δυναμικά μόνο μια φορά κατά την έναρξη της εφαρμογής και είναι από της ελάχιστες φορές για της οποίες δεν μας απασχολεί η απελευθέρωση της μνήμης.

```
Bango * initialize(int argc, char **argv){
     Bango *app;

      // *** //

     app = g_slice_new(Bango);
     return app;
}
```

Ένα απλό παράδειγμα χρήσης manual memory management αναφέρεται παρακάτω. Ο προγραμματιστής δεσμεύει δυναμικά ένα slice μνήμης για την δομή *ThumbPack* και όταν πλέον δεν την χρειάζεται απελευθερώνει την μνήμη με την εντολή *g_slice_free*. Το πρώτο όρισμα δηλώνει τον τύπο της μεταβλητής, το οποίο χρειάζεται για να υπολογίσει το μέγεθος της δεσμευμένης μνήμης και το δεύτερο όρισμα είναι ο pointer στην μνήμη για την μεταβλητή που θα απελευθερωθεί.

```
ThumbPack *pack = g_slice_new(ThumbPack);

// *** //

g_slice_free(ThumbPack , pack);
```

Στο παρακάτω παράδειγμα παρουσιάζεται η τεχνική της διαχείρισης μνήμης με reference counting. Η συνάρτηση *create_shows_model* δημιουργεί ένα μοντέλο (MVC pattern) το οποίο μεταξύ άλλον περιέχει ένα αντικείμενο pixbuf. Το αντικείμενο pixbuf παριστά μια εικόνα RGB στην μνήμη.

Όταν δημιουργήσουμε το αντικείμενο pixbuf, ο αριθμός αναφοράς είναι ένα, και είμαστε εμείς υπεύθυνη για το αντικείμενο καθώς εμείς το δημιουργήσαμε.

Όταν εισάγουμε το αντικείμενο pixbuf στο μοντέλο με την συνάρτηση *clutter_model_append* ο αριθμός αναφοράς αυξάνεται κατά ένα, καθώς πλέον το pixbuf χρησιμοποιείται και από το μοντέλο. Από την στιγμή που θέλουμε το μοντέλο να διαχειριστεί πλήρως το αντικείμενο pixbuf είναι σημαντικό να μειωθεί ο αριθμός αναφοράς από την πλευρά του προγραμματιστή. Η συνάρτηση *g_object_unref* μειώνει τον αριθμό αναφοράς. Όταν ο αριθμός αναφοράς φτάσει στο μηδέν, τότε το αντικείμενο απελευθερώνεται από τη μνήμη.

Στο συγκεκριμένο παράδειγμα γίνεται χρήση των συναρτήσεων *g_new0* και *g_free* όπου και αυτές αναφέρονται στην τεχνική manual memory management και είναι παρόμοιες με τις συναρτήσεις *g_slice_**.

```
void create_shows_model(Bango *app){
     sqlite3_stmt *stmt;

     sqlite3_prepare_v2(app->db,"SELECT id,name,thumb FROM shows WHERE id IN
(SELECT show_id FROM episodes WHERE episodes.show_id=shows.id) ORDER BY name",-
1,&stmt,NULL);

     while(sqlite3_step(stmt)==SQLITE_ROW){
            gchar *title;
            const guint8 * blob;
            int len;
            GdkPixdata * pixdata;
            GdkPixbuf * pixbuf = NULL;

            // *** //

            pixdata = g_new0 (GdkPixdata, 1);
            gdk_pixdata_deserialize(pixdata,len,(const guint8*)blob,NULL);
            pixbuf = gdk_pixbuf_from_pixdata(pixdata,TRUE,NULL);
            g_free(pixdata);

            clutter_model_append                                    (app->model_shows,
COL_S_TITLE,title,COL_S_ID,id,COL_S_THUMB,pixbuf, -1);

            g_object_unref (pixbuf);
     }
     sqlite3_finalize(stmt);
}
```

Η χρήση διαφορετικών βιβλιοθηκών σημαίνει ότι θα πρέπει να προσαρμοστούμε στον τρόπο όπου η βιβλιοθήκη διαχειρίζεται την μνήμη. Ο παρακάτω κώδικας περιλαμβάνει τρόπους απελευθέρωσης μνήμης χρησιμοποιώντας της τεχνικές της κάθε βιβλιοθήκης. Η συνάρτηση *curl_easy_cleanup* καλείται για να απελευθερώσει τα resources που έχουν δεσμευτεί μετά από μια μεταφόρτωση μέσω της βιβλιοθήκης libCurl. Η συνάρτηση *mrss_free* απελευθερώνει όλα τα resources της αναπαράστασης του RSS από την μνήμη. Εάν δεν γίνει απελευθέρωση των resources των δομών που επιστρέφουν οι βιβλιοθήκες, αυτό θα οδηγήσει σε memory leak.

```
gpointer thumbnail_downloader(ThumbnailPack *pack){
     CURL *curl;
     CURLcode res;
     mrss_item_t *item;
     mrss_t * rss;
     GdkPixbuf *pixbuf;
```

```
     GString *url = g_string_new("http://tracker/bango/rss/shows.php?show=");
     url = g_string_append(url,pack->show);

     rss = download_rss_data(url->str);

     item = rss->item;

     if(item)
           GString                 *                 filename                 =
path_convert(get_configuration_folder(THUMB_FOLDER));
           filename = g_string_append(filename,item->title);

           gchar   *   uri   =   g_uri_escape_string(item->enclosure_url,":/=?&
%",TRUE);

           file = fopen(filename->str,"w");

           curl = curl_easy_init();
           if(curl) {
                 curl_easy_setopt(curl, CURLOPT_URL, uri);
                 curl_easy_setopt(curl, CURLOPT_WRITEDATA , file);
                 res = curl_easy_perform(curl);
                 curl_easy_cleanup(curl);
           }
           g_free(uri);
           fclose(file);
           g_string_free (filename,TRUE);

     }
     mrss_free(rss);
}
```

Για μεγαλύτερη ευκολία στην δημιουργία και επεξεργασία συμβολοσειρών γίνεται η χρήση του αντικειμένου Gstring της βιβλιοθήκης Glib. Το αντικείμενο Gstring κρύβει την πολυπλοκότητα για δημιουργία και τροποποίηση της συμβολοσειράς. Παρακάτω περιγράφετε ένα παράδειγμα χρήσης Gstring. Αξίζει να σημειωθεί ότι η συνάρτηση *g_string_free* όπου απελευθερώνει το αντικείμενο Gstring παίρνει ένα δεύτερο όρισμα το οποίο ορίζει εάν θα απελευθερωθεί η συμβολοσειρά που έχει δημιουργηθεί. Σε αυτήν την περίπτωση ο προγραμματιστής θα πρέπει να καλέσει την συνάρτηση *g_free* για να απελευθερώσει την συμβολοσειρά.

```
gchar *  example(gchar * save_location){
     gchar * str;
     s = g_string_new(save_location);
     s = path_convert(s);
     s = g_string_append(s,"Completed");
     str = s->str;
     g_string_free (s,FALSE);
     return str;
}
```

Στην παρακάτω συνάρτηση γίνεται δέσμευση μνήμης με την συνάρτηση *g_malloc0* για την δημιουργία μιας συμβολοσειράς.

```
gchar * seconds_to_time(guint position, guint duration){
     int hours_p = position / 60 / 60;
     int min_p = position / 60 - hours_p*60;
     int sec_p = position - hours_p*60*60 - min_p*60;
```

```
     int hours_d = duration / 60 / 60;
     int min_d = duration / 60 - hours_d*60;
     int sec_d = duration - hours_d*60*60 - min_d*60;

     gchar *buffer = (gchar*)g_malloc0 (50);

     g_sprintf(buffer,"%01d:%02d:%02d                /                %01d:%02d:
%02d",hours_p,min_p,sec_p,hours_d,min_d,sec_d);

     return buffer;
}
```

Όσον αφορά την τεχνική manual memory management αναφέρθηκαν αρκετές συναρτήσεις για την δέσμευση μνήμης. Από την αρκετά γνωστή g_malloc0 η οποία είναι μια ενθυλάκωση της malloc του λειτουργικού συστήματος μέχρι την g_slice_new. Η διαφορά τους βρίσκεται στην τεχνικές που χρησιμοποιούνται για την δέσμευση μνήμης.

Ενώ η *g_malloc0* χρησιμοποιεί την υλοποίηση της malloc του λειτουργικού συστήματος, η *g_slice_new* υλοποιεί ένα πολύπλοκο σύστημα βασισμένο στον Bonwick's slab allocator, χρησιμοποιώντας την συνάρτηση *posix_memalign* για τη βελτιστοποίηση της κατανομής πολλών ίσον memory-chunks.

5 Αποτελέσματα

Σε όλη την πορεία της ανάπτυξης της ψηφιακής πλατφόρμας ασύγχρονης τηλεόρασης ήρθα αντιμέτωπος με πολλές προκλήσεις και προβλήματα. Τα προβλήματα χρειάζονται λύσεις, και οι λύσεις πολύ συχνά είναι δύσκολες να βρεθούνε. Οι προκλήσεις χρειάζονται σωστές επιλογές, και οι επιλογές μπορεί στο τέλος να είναι λανθασμένες. Στο τέλος της πτυχιακής εργασίας, έλυσα πολλά από τα προβλήματα τα οποία αντιμετώπισα και πιστεύω ότι πήρα της σωστές αποφάσεις στις προκλήσεις που ανέκυψαν.

Τα οφέλη που αποκόμισα από την ενασχόληση μου με την δημιουργία της πλατφόρμας ασύγχρονης τηλεόρασης, μου έδωσαν τα κατάλληλα εφόδια έτσι ώστε να είμαι πολύ καλά καταρτισμένος, περνώντας από το θεωρητικό επίπεδο στην πρακτική εφαρμογή των γνώσεων που έλαβα από της σπουδές μου στο ΤΕΙ Κρήτης. Μπόρεσα να ασχοληθώ με τεχνολογίες που ποτέ πριν δεν είχα ασχοληθεί και μου δόθηκε η ευκαιρία να εφαρμόσω τις αρχές της πληροφορικής που γνώριζα αλλά πότε δεν είχα εφαρμόσει στην πράξη.

Η πτυχιακή εργασία θα μπορούσε να βοηθήσει και άλλους φοιτητές να ασχοληθούν με τις τεχνολογίες που έχουν χρησιμοποιηθεί στην ασύγχρονη πλατφόρμα. Θεωρώ ότι η χρήση του node.js είναι τελείως καινούργια για τον ακαδημαϊκό χώρο του ΤΕΙ Κρήτης και είμαι ειδικότερα περήφανος για το ότι μπόρεσα να ασχοληθώ με αυτήν την τεχνολογία, που πιστεύω ότι θα γίνει ένα βασικό κομμάτι των διαδικτυακών εφαρμογών του μέλλοντος.

5.1 Συμπεράσματα

Η επιτυχής υλοποίηση της πλατφόρμας ασύγχρονης τηλεόρασης μέσω διαδικτύου δηλώνει ξεκάθαρα ότι η χρήση της πλατφόρμας είναι εφικτή, αξιόπιστη και βιώσιμη. Θα μπορούσε να συνεχιστεί η ανάπτυξη έτσι ώστε να δημιουργηθεί ένα επιχειρηματικό μοντέλο που θα βασίζεται σε αυτήν. Η ανάπτυξη της τεχνολογίας σημαίνει ότι θα πρέπει διαρκώς να βρίσκουμε καινούργιες υπηρεσίες που να βελτιώνουν τη ζωή μας. Μια τέτοια υπηρεσία είναι η πλατφόρμα ασύγχρονης τηλεόρασης που δημιουργήσαμε.

Υλοποιήσαμε ένα σύστημα το οποίο πληρεί τις προδιαγραφές που είχαμε θέσει εξ' αρχής για την πλατφόρμα. Αξιοποιεί πλήρως της νέες τεχνολογίες και υλοποιεί όλα τα παρακάτω χαρακτηριστικά:

- Υψηλή ποιότητα εικόνας και ήχου
- Ελκυστική και Εύχρηστη διεπαφή
- Υψηλές ταχύτητες μεταφοράς δεδομένων
- Εξοικονόμηση εύρους ζώνης
- Μικρο κόστος απόκτησης
- Βιωσιμότητα
- Ευέλικτη αρχιτεκτονική

5.2 Απόδοση συστήματος

Για να μετρήσουμε την απόδοση του συστήματος χρησιμοποιήσαμε την online υπηρεσία blitz.io. Η υπηρεσία που προσφέρει το blitz.io είναι η μέτρηση της απόδοσης ενός web συστήματος. Έχει τη δυνατότητα να ανοίξει έως και 50.000 ταυτόχρονες συνδέσεις από πολλά γεωγραφικά σημεία προς μια διεύθυνση URL.

5.2.1 Χαρακτηριστικά διακομιστή

Για να μπορέσουμε να κάνουμε της μετρήσεις σε όσο το δυνατόν πιο πραγματικό περιβάλλον, νοικιάσαμε έναν server από την εταιρία Hetzner στην Γερμανία είναι συνδεδεμένος σε δίκτυο με διαθέσιμο bandwidth 240 Gbit και κάρτα σύνδεσης με το δίκτυο 100 Mbps. Ο server έχει τα παρακάτω χαρακτηριστικά.

- Intel® Core™ i7-2600 Quadcore
- 16 GB DDR3 RAM
- 2 x 3 TB SATA 6 Gb/s HDD 7200 rpm Software-RAID 1
- NIC 1 GBit connected at 100 Mbit
- Debian 6.0 64bit
- IPv6 subnet (/64)

5.2.2 Σενάρια μετρήσεων

Για τις μετρήσεις δημιουργήσαμε τρία σενάρια χρήσεις. Το πρώτο σενάριο αφορά μετρήσεις χωρίς επιπρόσθετα τα οποία θα μείωναν την απόδοση του συστήματος. Δηλαδή την προσφορά μόνο στατικού περιεχομένου. Ο λόγος που προχωρήσαμε σε μια τέτοια μέτρηση ήταν για να δούμε την απόδοση των συστημάτων χωρίς να συμπεριλαμβάνονται άλλοι παράγοντες.

Το δεύτερο σενάριο αφορά το video streaming και το τρίτο σενάριο αφορά ένα σύστημα με πιο ολοκληρωμένες παροχές.

Πιο συγκεκριμένα, οι μετρήσεις αφορούν την πλατφόρμα node.js και των web server Cherokee. Να σημειωθεί ότι από μετρήσεις που έχουν γίνει ο Cherokee είναι από τους πιο γρήγορους web server μαζί με τους nginx και lighthttpd.

Το πρώτο σενάριο είναι η μετάδοση ενός στατικού αρχείου κειμένου. Το δεύτερο σενάριο μετράει την απόδοση στο streaming του συστήματος που έχουμε γράψει στο node.js και του συστήματος του module media streaming του Cherokee web server. Το τρίτο σενάριο μετράει την απόδοση χρησιμοποιώντας components τα οποία χρησιμοποιούνται συνήθως στις εφαρμογές web. Για την πλατφόρμα του node.js αυτό περιλαμβάνει την σύνδεση σε βάση δεδομένων, την αποστολή cookie στον browser, την δημιουργία HTML από template. Όλα τα παραπάνω γίνονται για κάθε request. Για τον Cherokee web server υπάρχει υποστήριξη PHP, σύνδεση με βάση δεδομένων και δημιουργία HTML από PHP αρχεία.

Οι μετρήσεις έγιναν για ένα λεπτό και υπήρχε σταδιακή αύξηση του αριθμού των χρηστών από 1 έως 5000 ταυτόχρονους χρήστες.

5.2.3 Μετρήσεις

Οι μετρήσεις έγιναν σύμφωνα με τα παραπάνω κριτήρια. Παρουσιάζονται όλα τα στοιχεία τα οποία έγιναν διαθέσιμα από το εργαλείο blitz.io

5.2.3.1 Σενάριο 1 - Αναφορά

Analysis for Node.js

This rush generated 84,144 successful hits in 1.0 min. The average hit rate of 1343/second translates to about 116,059,966 hits/day. The average response time of 562 ms is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 3.32% of the users during this rush experienced timeouts or errors!

Errors

The first error happened at 10.03 seconds into the test when the number of concurrent users was at 834. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

Timeouts

The first timeout happened at 20.07 seconds into the test when the number of concurrent users was at 1671. Looks like you've been rushing with a timeout of 1 second.

Analysis for Cherokee

This rush generated 70,794 successful hits in 1.0 min. The average hit rate of 1129/second translates to about 97,559,758 hits/day. The average response time of 793 ms is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 5.71% of the users during this rush experienced timeouts or errors!

Errors

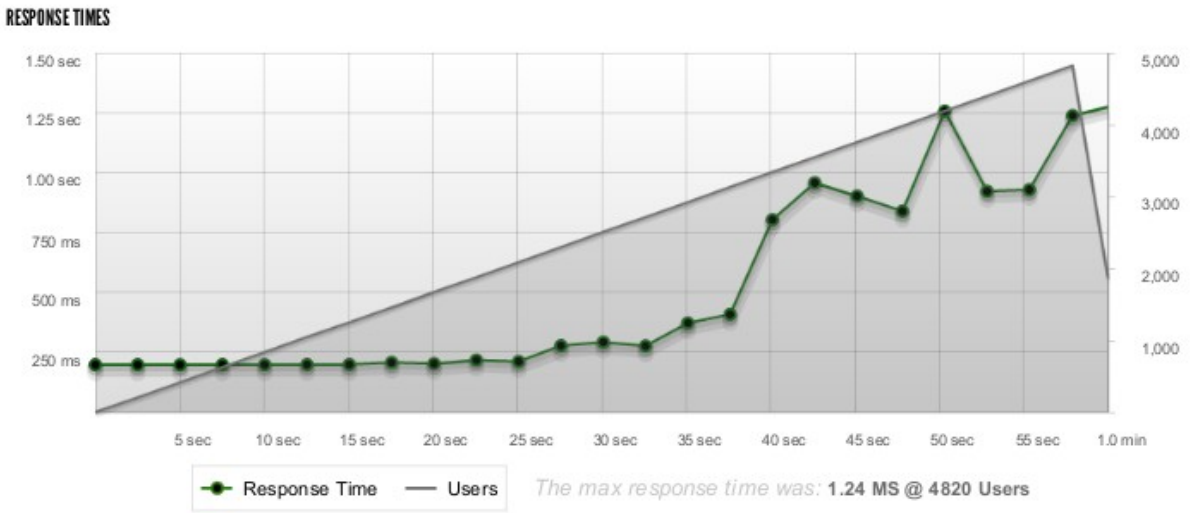
The first error happened at 22.68 seconds into the test when the number of concurrent users was at 1885. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

Timeouts

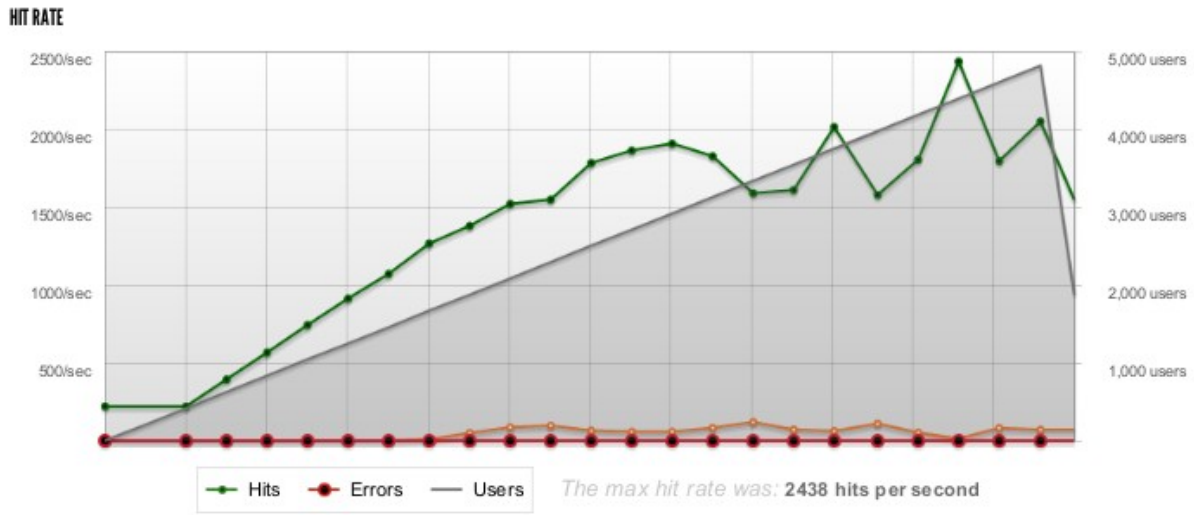
The first timeout happened at 15.13 seconds into the test when the number of concurrent users was at 1260. Looks like you've been rushing with a timeout of 1 second.

5.2.3.2 Σενάριο 1 – Γραφήματα

Node.js



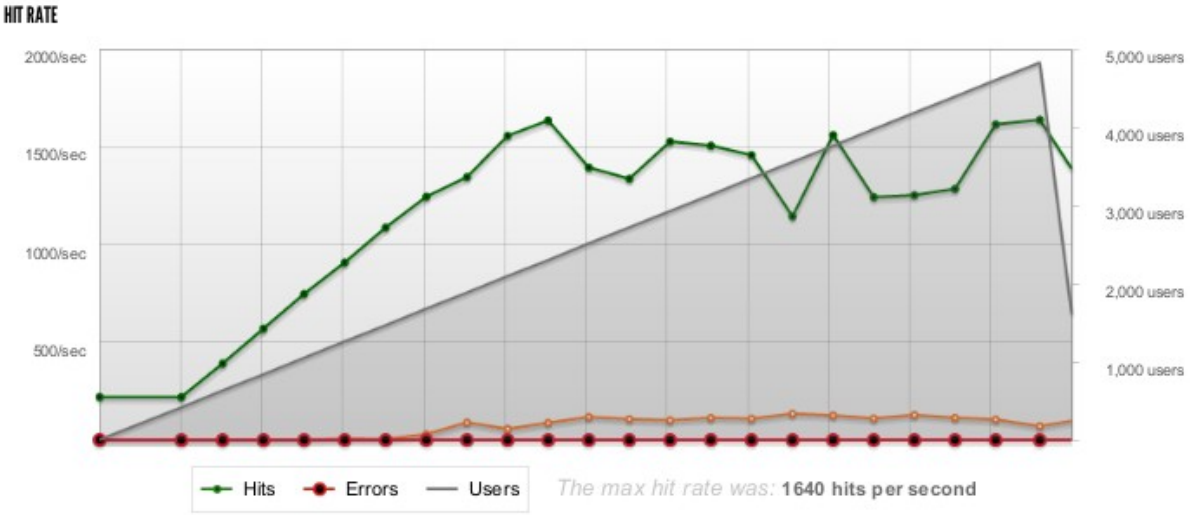
Εικόνα 22: Σενάριο 1 - Node.js - Response Times



Εικόνα 23: Σενάριο 1 - Node.js - Hit Rate

Cherokee

Εικόνα 24: Σενάριο 1 - Cherokee - Response Times



Εικόνα 25: Σενάριο 1 - Cherokee - Hit Rate

5.2.3.3 Σενάριο 2 - Αναφορά

Analysis for Node.js

This rush generated 46,949 successful hits in 1.0 min. The average hit rate of 751/second translates to about 64,921,064 hits/day. The average response time of 1.45 seconds is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 15.62% of the users during this rush experienced timeouts or errors!

Errors

The first error happened at 30.17 seconds into the test when the number of concurrent users was at 2512. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

Timeouts

The first timeout happened at 17.58 seconds into the test when the number of concurrent users was at 1463. Looks like you've been rushing with a timeout of 1 second.

Analysis for Cherokee

This rush generated 20,377 successful hits in 1.0 min. The average hit rate of 331/second translates to about 28,663,439 hits/day. The average response time of 1.95 seconds is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 49.22% of the users during this rush experienced timeouts or errors!

Errors

The first error happened at 10.03 seconds into the test when the number of concurrent users was at 832. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

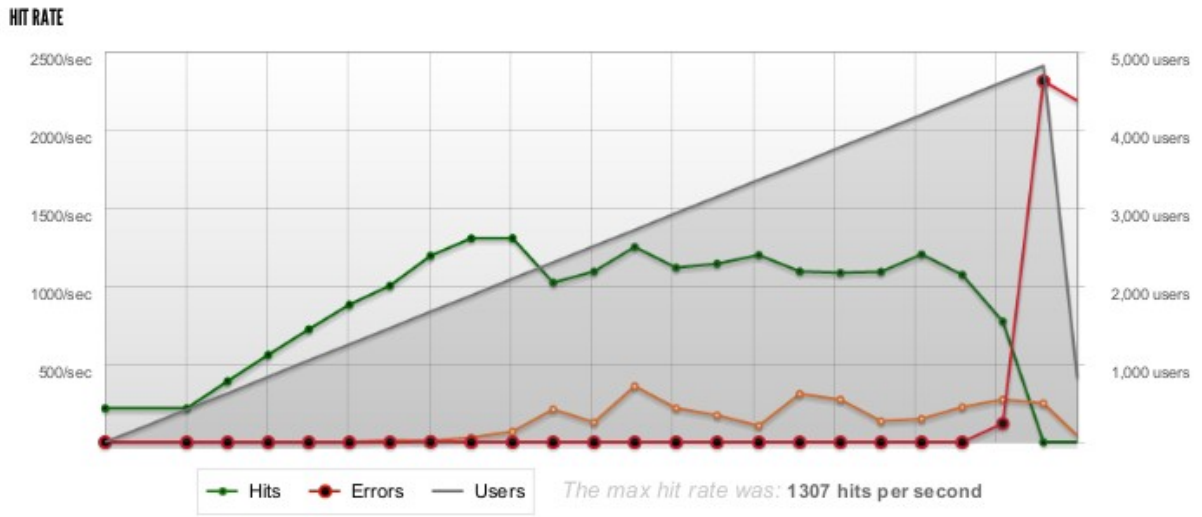
Timeouts

The first timeout happened at 15.05 seconds into the test when the number of concurrent users was at 1252. Looks like you've been rushing with a timeout of 1 second.

5.2.3.4 Σενάριο 2 – Γραφήματα

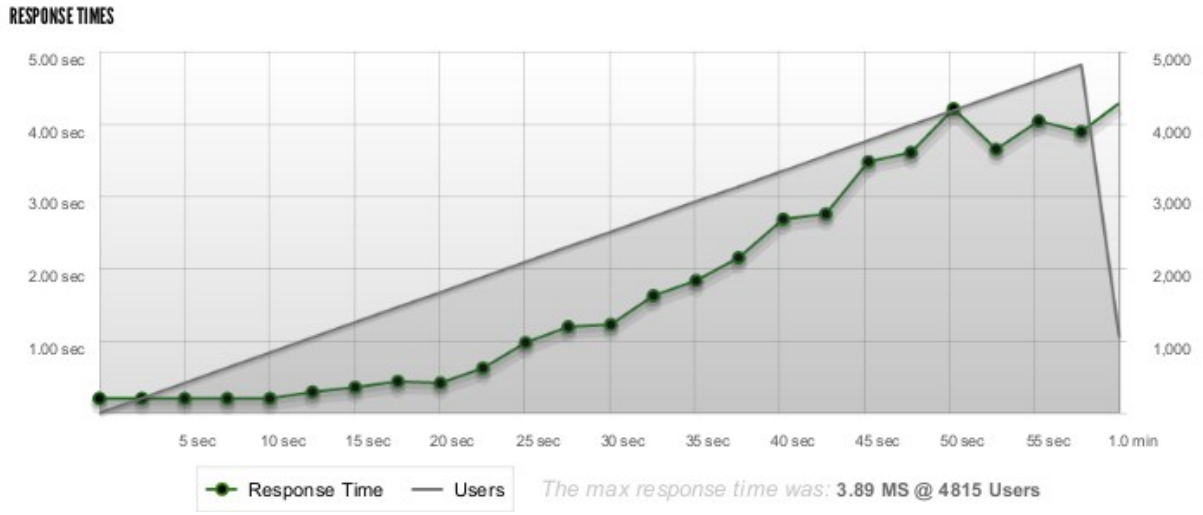
Node.js

Εικόνα 26: Σενάριο 2 - Node.js - Response Times

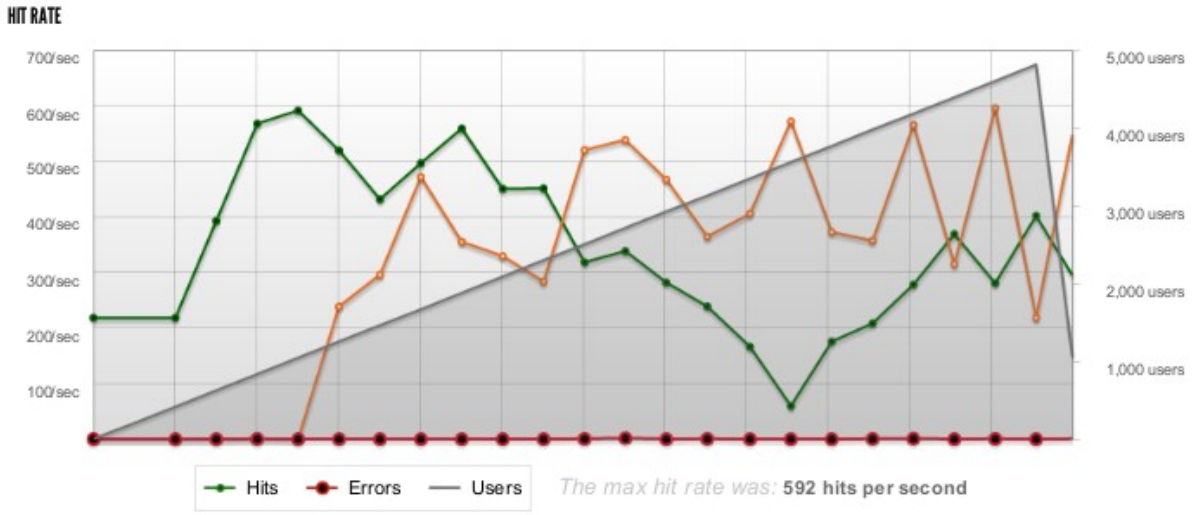


Εικόνα 27: Σενάριο 2 - Node.js - Hit Rate

Cherokee



Εικόνα 28: Σενάριο 2 - Cherokee - Response Times



Εικόνα 29: Σενάριο 2 - Cherokee - Hit Rate

5.2.3.5 Σενάριο 3 - Αναφορά

Analysis for Node.js

This rush generated 52,320 successful hits in 1.0 min. The average hit rate of 872/second translates to about 75,375,140 hits/day. The average response time of 752 ms is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 26.50% of the users during this rush experienced timeouts or errors!

Errors

The first error happened at 20.09 seconds into the test when the number of concurrent users was at 1672. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

Timeouts

The first timeout happened at 17.59 seconds into the test when the number of concurrent users was at 1459. Looks like you've been rushing with a timeout of 1 second.

Analysis for Cherokee

This rush generated 21,657 successful hits in 1.0 min. The average hit rate of 351/second translates to about 30,389,243 hits/day. The average response time of 1.84 seconds is considerably higher than most other sites that are built to scale out. Response times less than 250 ms are what the cool kids strive for. You've got bigger problems, though: 46.67% of the users during this rush experienced timeouts or errors!

Errors

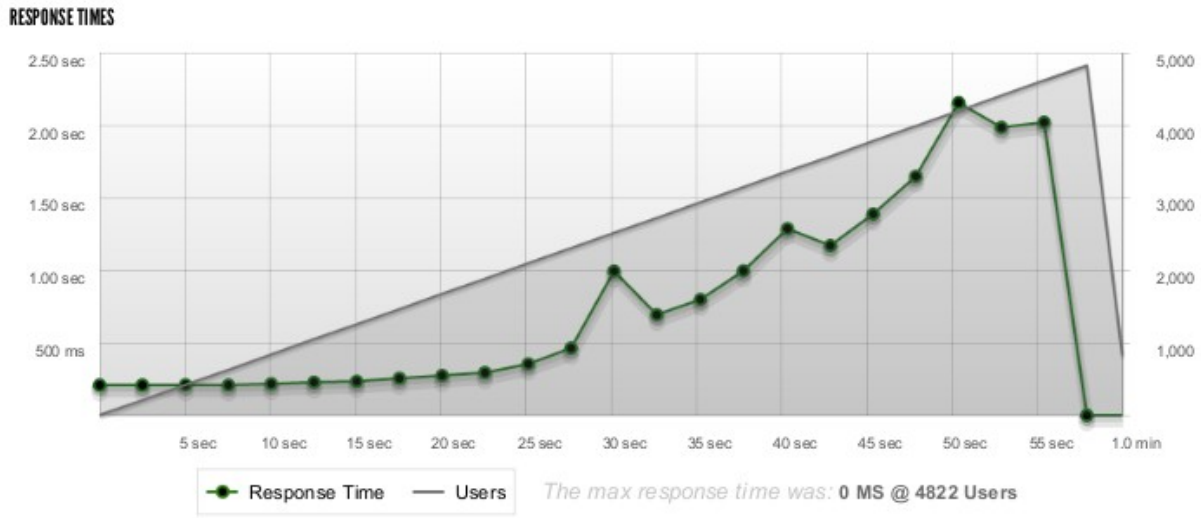
The first error happened at 30.10 seconds into the test when the number of concurrent users was at 2501. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

Timeouts

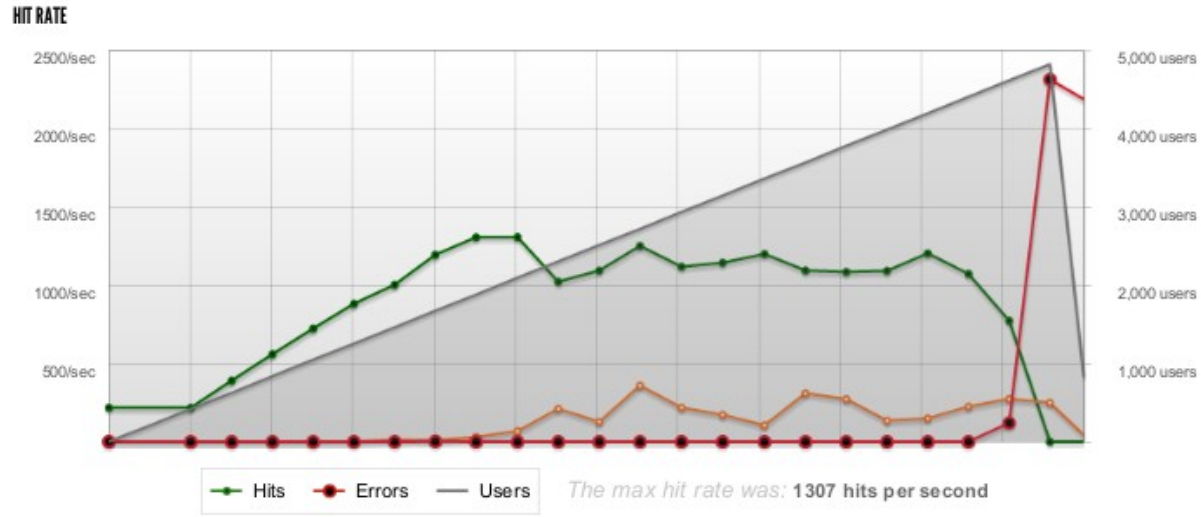
The first timeout happened at 10.03 seconds into the test when the number of concurrent users was at 836. Looks like you've been rushing with a timeout of 1 second.

5.2.3.6 Σενάριο 3 – Γραφήματα

Node.js



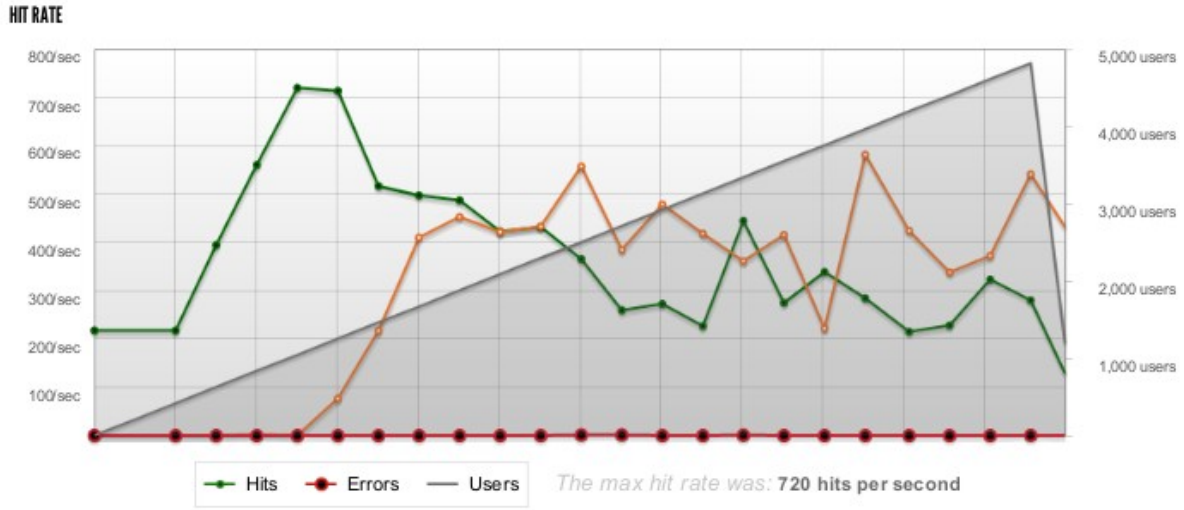
Εικόνα 30: Σενάριο 3 - Node.js - Response Times



Εικόνα 31: Σενάριο 3 - Node.js - Hit Rate

Cherokee

Εικόνα 32: Σενάριο 3 - Cherokee - Response Times



Εικόνα 33: Σενάριο 3 - Cherokee - Hit Rate

5.2.4 Συμπέρασμα

Η εκτέλεση benchmarks σε ένα σύστημα έχει ως στόχο να μας αποκαλύψει της δυνατότητες του συστήματος κάτω από μεγάλο φόρτο. Όλα τα συστήματα έχουν μια πεπερασμένη αντοχή και ένα συγκεκριμένο όριο που έπειτα από αυτό αρχίζει να μειώνεται η απόδοση τους ή να παρουσιάζονται προβλήματα στην λειτουργία τους. Είναι λοιπόν φυσιολογικό για ένα σύστημα να μειώσει την απόδοση κάτω από μεγάλο φόρτο ή ακόμα και να διακόψει την λειτουργία του. Η απόδοση του συστήματος αναφέρεται στο μέγεθος του φόρτου που λειτουργεί αξιόπιστα το σύστημα. Η σύγκριση δυο συστημάτων αναφέρεται στο πιο σύστημα ανταποκρίνεται καλύτερα σε σχέση με τον αυξανόμενο φόρτο.

Ένα σύστημα μπορεί να αποδίδει διαφορετικά ανάλογα με την χρήση. Για παράδειγμα, η παροχή στατικού περιεχομένου θα μπορούσε να είναι αρκετά αποδοτική για ένα σύστημα ενώ η παροχή δυναμικού περιεχομένου να κάνει το σύστημα μη αποδοτικό. Είναι λοιπόν σημαντικό να επιλέξουμε ένα αποδοτικό σύστημα για τις ανάγκες και της λειτουργίες που θα το χρησιμοποιήσουμε.

Γνωρίζοντας εκ τον προτέρων την τεχνολογία που βασίζονται τα συστήματα που θέλουμε να συγκρίνουμε, μπορούμε να υπολογίσουμε για ποια χρήση είναι κατάλληλο κάθε σύστημα. Τα δυο συστήματα βασίζονται στο event-loop για την εξυπηρέτηση των χρηστών. Είναι συστήματα που από τεχνολογικής πλευράς μπορούν να ανταγωνιστούν το ένα το άλλο. Δεν συγκρίνουμε συστήματα τα οποία είναι διαφορετικά, όπως για παράδειγμα ένα σύστημα που βασίζεται σε event-loop, με ένα σύστημα που δημιουργεί καινούργιες διεργασίες για κάθε σύνδεση, καθώς η διαφορά στην απόδοση θα ήταν μεγάλη από θεωρητική και πρακτική πλευρά.

Παρατηρώντας τα αποτελέσματα του σεναρίου 1 καταλαβαίνουμε ότι η απόδοση τον δυο συστημάτων είναι σχεδόν η ίδια. Αν και υπάρχει μια μικρή διαφορά υπέρ του node.js θεωρούμε ότι για την παροχή στατικού περιεχομένου η απόδοση είναι η ίδια.

Στο σενάριο 2 μπορούμε να βγάλουμε περισσότερα συμπεράσματα, καθώς αρχίζει να φαίνεται ότι το node.js υπερτερεί έναντι του web server Cherokee. Επίσης το γεγονός ότι ο web server Cherokee μπορεί να προσφέρει μόνο στατικά αρχεία πολυμέσων, έναντι της δυνατότητας του node.js να προσφέρει ροή από διάφορους τύπους εισόδου, όπως η μετάδοση live προγράμματος, καθιστά το node.js ως καλύτερη επιλογή για streaming πολυμέσων.

Στο τελευταίο σενάριο, το οποίο προσομοιώνει ένα σύστημα δυναμικού περιεχομένου, η απόδοση του node.js είναι πολύ καλύτερη έναντι του Cherokee. Αξίζει να σημειωθεί ότι η μειωμένη απόδοση του Cherokee οφείλεται στα υποσυστήματα τα οποία περιλαμβάνει όπως ο interpreter της PHP, που μειώνει αρκετά την απόδοση του συστήματος. Επίσης, η αξιολόγηση ενός συστήματος κυρίως περιλαμβάνει το μέγεθος του φόρτου όπου το σύστημα έχει μια συγκεκριμένη απόδοση και όχι πόσο αντέχει το σύστημα μέχρι να σταματήσει να λειτουργεί.

Λαμβάνοντας υπόψιν τις παραπάνω μετρήσεις, η καλύτερη χρήση του node.js και του Cherokee είναι ο συνδυασμός τους. Ο Cherokee ως Reverse Proxy και Static file server και το node.js θα αναλαμβάνει οποιοδήποτε δυναμικό περιεχόμενο καθώς και το streaming των πολυμεσικών αρχείων. Έτσι και τα δυο συστήματα θα συνεργάζονται και θα υπάρχει μεγάλη αύξηση της απόδοσης της πλατφόρμας ως σύνολό.

5.3 Μελλοντική Εργασία και Επεκτάσεις

5.3.1 Μελλοντική ανάπτυξη

Η ανάπτυξη της πτυχιακής εργασίας έγινε στα πλαίσια της υλοποίησης της ιδέας της πλατφόρμας ασύγχρονης τηλεόρασης, έτσι ώστε να αξιολογήσουμε τα αποτελέσματα για το εάν είναι μια υλοποιήσιμη ιδέα που θα μπορούσε να είναι η βάση για μια επιτυχημένη επιχειρηματική δραστηριότητα.

Ένα από τα υποσυστήματα τα οποία θα είχε νόημα να αναπτύξουμε λόγο του χαμηλού ανταγωνισμού, είναι το υποσύστημα Streaming μέσω HTTP που έχουμε υλοποιήσει σε node.js. Θεωρούμε ότι η ανάπτυξη της συγκεκριμένης πλατφόρμας στα επόμενα χρόνια σε συνδυασμό με τα χαρακτηριστικά του, θα είναι μια βασική τεχνολογία που θα χρησιμοποιηθεί στο διαδίκτυο.

Όσον αφορά το λογισμικό Set-top-Box που έχουμε αναπτύξει, λόγο του αυξημένου ανταγωνισμού καθώς και λόγο των πολλών και καλύτερων υλοποιήσεων, όπως το Boxee Box, θα είχε νόημα να προχωρήσουμε στην ανάπτυξη προγραμμάτων που θα ενσωμάτωναν την δική μας πλατφόρμα στις υλοποιήσεις αυτές, και όχι να συνεχίσουμε με την ανάπτυξη του δικού μας πρωτοτύπου.

5.3.2 Προοπτικές για Επιχειρηματικότητα

Παρατηρώντας την ολοένα αυξανόμενη ενασχόληση των Ελλήνων με το διαδίκτυο, δημιουργούνται επιχειρηματικές ευκαιρίες που σχετίζονται με την παροχή τηλεοπτικού προγράμματος μέσω διαδικτύου. Αν και στο εξωτερικό υπάρχουν πολλές υπηρεσίες που δίνουν την δυνατότητα στους χρήστες τους να παρακολουθήσουν τηλεοπτικό πρόγραμμα με ασύγχρονο τρόπο, δηλαδή να έχει προβληθεί σε προγενέστερο χρόνο, στην Ελλάδα οι υπηρεσίες αυτές βρίσκονται σε νηπιακό στάδιο και δεν προσφέρουν την εμπειρία χρήσης που θα μπορούσε να προσελκύσει το ενδιαφέρον τον χρηστών.

Η ιδέα μιας ενιαίας ψηφιακής πλατφόρμας ασύγχρονης τηλεόρασης, όπου ο χρήστης θα μπορεί να έχει πρόσβαση από ένα κεντρικό σημείο σε πολυποίκιλα τηλεοπτικά προγράμματα ανεξαρτήτου παραγωγού ή τηλεοπτικού σταθμού, είναι μια ιδέα που θεωρούμε ότι έχει προοπτικές για επιχειρηματικότητα.

5.3.3 OTE Bussiness Live

Στην διάρκεια της ανάπτυξης της πτυχιακής εργασίας, εργαζόμουν στην εταιρία OgilvyOne Worldwide Athens. Για λογαριασμό του ΟΤΕ, του μεγαλύτερου τηλεπικοινωνιακού φορέα της Ελλάδος χρησιμοποίησα μέρος του κώδικα που είχα αναπτύξει για την πλατφόρμα έτσι ώστε να υποστηρίξω μια διαδραστική καμπάνια.

Εικόνα 34: Η ιστοσελίδα otebusinesslive.gr

Η ιδέα ήταν να μπορέσουν οι ελληνικές εταιρίες να διαφημιστούν σε ένα billboard σε κάποιο κεντρικό σημείο του Λονδίνου. Οι χρήστες ανέβαζαν μια διαφήμιση, και αυτή μετά από μερικά λεπτά εμφανιζόταν στο Λονδίνο.

Το billboard ήταν συνδεδεμένο με έναν υπολογιστή που έτρεχε μια περιορισμένη και λίγο διαφοροποιημένη έκδοση του λογισμικού για το set-top-box. Η απόδοση του λογισμικού καθώς και η επιτυχία της συγκεκριμένης διαφημιστικής καμπάνιας επιβεβαίωσαν την αξιοπιστία του λογισμικού να ανταποκριθεί σε καθημερινή χρήση.

Αναφορά για την καμπάνια έγινε από πολλές ιστοσελίδες καθώς και τηλεοπτικούς σταθμούς. Επίσης διακρίθηκε με το βραβείο Best Special Audience Campaign των Telekom Media Awards 2012,

που διοργανώνει ετησίως η Deutsche Telecom. Ένα case study για την ενέργεια βρίσκετε στην διεύθυνση: http://www.ogilvyone.gr/blog/i-ogilvyone-sto-londino

Εικόνα 35: Αναφορά στο OTE Business Live απο τηλεοπτικούς σταθμούς

Βιβλιογραφία

[1] http://blog.chromium.org/2011/11/game-changer-for-interactive.html
[2] https://developers.google.com/v8/design
[3] https://developers.google.com/v8/get_started
[4] http://en.wikipedia.org/wiki/Reference_counting
[5] http://forum.linuxbasix.com/viewtopic.php?f=20&t=93
[6] http://developer.gnome.org/clutter/1.8/index.html
[7] http://developer.gnome.org/glib/2.32/
[8] http://en.wikipedia.org/wiki/Node.js
[9] http://www.mywikinet.com/mpl/paper/html/
[10] http://www.boost.org/doc/libs/1_49_0/
[11] http://www.rasterbar.com/products/libtorrent/manual.html
[12] http://www.cherokee-project.com/doc/
[13] http://en.wikipedia.org/wiki/C%2B%2B
[14] http://www.autistici.org/bakunin/libmrss/doc/
[15] http://www.php.net/manual/en/
[16] http://www.hulu.com/labs
[17] http://curl.haxx.se/libcurl/c/
[18] http://nodejs.org/api/
[19] http://www.robobrarian.info/b_pubs/IJSNSchrader.pdf
[20] http://en.wikipedia.org/wiki/Boxee_Box
[21] http://www.vudu.com/
[22] http://paulstamatiou.com/review-vudu
[23] http://www.roku.com/
[24] http://www.google.com/tv/
[25] http://www.apple.com/appletv/
[26] https://bugbase.adobe.com/index.cfm?event=file.view&id=2943064&seqNum=6&name=httpdynamicstreaming_wp_ue.pdf