ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ: « ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ ΜΕΤΑΔΟΣΗΣ ΒΙΝΤΕΟ ΣΕ ΔΙΚΤΥΟ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ (Peer- To-Peer) »

ΠΡΟΠΤΥΧΙΑΚΟΣ ΦΟΙΤΗΤΗΣ:

ΚΟΣΜΑΝΟΣ ΔΗΜΗΤΡΙΟΣ

Α.Ε.Μ.: 720

Στην Οικογένεια μου,

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω για την πολύτιμη συμβολή και υποστήριξη του σε όλο το χρονικό διάστημα της εκπόνησης της συγκεκριμένης Διπλωματικής Εργασίας τον κ. Αργυρίου Αντώνιο. Επίσης θα ήθελα να ευχαριστήσω και τον κ. Τασιούλα Λέανδρο για την βοήθεια του στην εύρεση των πηγών για τη συγγραφή αυτής της Εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

1)ΕΙΣΑΓΩΓΗ

Τα συστήματα **Peer-to-Peer** (**P2P**) είναι συστήματα ομότιμων δικτύων, στα οποία οι κόμβοι συμμετέχουν ισότιμα στο δίκτυο, για να επιτευχθεί η αποτελεσματική επικοινωνία μεταξύ τους στο δίκτυο. Τα συστήματα αυτά προσφέρουν στους χρήστες τη δυνατότητα να μοιραστούν αρχεία και δεδομένα, και στις περισσότερες περιπτώσεις μοιράζονται αρχεία πολυμέσων.

Τα συστήματα ομότιμων κόμβων (**peer-to-peer**) είναι κατανεμημένα δίκτυα στα οποία οι κόμβοι (**nodes**) συνδέονται μεταξύ τους με διάφορους τρόπους και τεχνικές. Το χαρακτηριστικό των συστημάτων αυτών είναι ότι όλοι οι κόμβοι είναι ομότιμοι, δηλαδή έχουν τις ίδιες δυνατότητες, κρατούν ίδιο μέγεθος πληροφορίας και έχουν τις ίδιες ευθύνες. Στα αρχικά συστήματα ομότιμων κόμβων (**Napster**) υπήρχε ένας κεντρικός κόμβος ο οποίος κράταγε περισσότερη πληροφορία από τους άλλους και ήταν υπεύθυνος για την αναζήτηση αντικειμένων καθώς και για την εισαγωγή ή διαγραφή ενός κόμβου. Αργότερα αναπτύχθηκαν τα πλήρως κατανεμημένα δίκτυα **P2P** (π.χ. **Gnutella**) τα οποία, όμως , για μεγάλο αριθμό κόμβων παρουσίαζαν προβλήματα. Σήμερα, παρουσιάζονται νέα συστήματα ομότιμων κόμβων που προσπαθούν να λύσουν τέτοια προβλήματα και νέες τεχνικές αναζήτησης σε υπάρχοντα συστήματα.

Ένα **Peer-to-Peer** (**P2P**) δίκτυο υπολογιστών εκμεταλλεύεται τη διαφορετική συνδεσιμότητα μεταξύ εκείνων των κόμβων που συμμετέχουν σε ένα δίκτυο και του συσσωρευτικού εύρους ζώνης των συμμετεχόντων δικτύων παρά τους συμβατικούς συγκεντρωμένους πόρους όπου ένας σχετικά χαμηλός αριθμός κεντρικών υπολογιστών (**servers**) παρέχει την αξία των πυρήνων σε μια υπηρεσία ή μια εφαρμογή. Τα **Peer-to-Peer** δίκτυα χρησιμοποιούνται χαρακτηριστικά για τη σύνδεση των κόμβων μέσω των κατά ένα μεγάλο μέρος ειδικών συνδέσεων. Τέτοια δίκτυα είναι χρήσιμα για πολλούς λόγους. Μοιράζουν τα αρχεία που περιέχουν τους ήχους, τα βίντεο, τα δεδομένα ή οτιδήποτε είναι σε ψηφιακή μορφή κάτι που είναι πολύ κοινό, καθώς και δεδομένα σε πραγματικό χρόνο, όπως η τηλεφωνική ροή, χρησιμοποιώντας και αυτά την **P2P** τεχνολογία.

Ένα απλό **Peer-to-Peer** δίκτυο δεν έχει την έννοια των πελατών (**clients**) ή των κεντρικών υπολογιστών (**servers**), αλλά είναι ίσο μόνο με τους όμοιους κόμβους που λειτουργούν ταυτόχρονα όπως τους "**clients**" και "**servers**" στους άλλους κόμβους στο δίκτυο. Αυτό το μοντέλο δικτύων διαφέρει από το μοντέλο πελάτη-εξυπηρετητή όπου η επικοινωνία είναι συνήθως "σε" και "από" έναν κεντρικό υπολογιστή.

Η επιτυχία ενός μηχανισμού διαμοιρασμού αρχείων σε ένα **Peer-to-Peer** σύστημα σαν το **BitTorrent** έχει παρακινήσει ιδέες για τη δημιουργία νέων μεθόδων για τμηματικό streaming ζωντανού περιεχομένου, οι οποίοι ονομάζονται **mesh-based Peer-to-Peer (P2P) streaming**. Σε μια τέτοια μέθοδο, οι peers οργανώνονται σε έναν τυχαία συνδεδεμένο mesh, στον οποίο οι peers διαμοιράζονται το περιεχόμενο που έχουν με στόχο να κάνουν streaming αυτού του ζωντανού περιεχομένου. Ο πιο πρόσφατος και διαδεδομένος τέτοιος αλγόριθμος είναι ο αλγόριθμος Prime, ο οποίος είναι ένας streaming μηχανισμός για ζωντανό περιεχόμενο (live content). Αποδεικνύεται μέσω αυτού του αλγορίθμου ένα παγκόσμιο πρότυπο για το διαμοιρασμό ενός τμήματος ζωντανού περιεχομένου, μέσω δύο διαδοχικών φάσεων, τη φάση diffusion και τη φάση swarming. Οι δύο αυτές διαδοχικές φάσεις έχουν σαν αποτέλεσμα οι peers να εκμεταλλεύονται πλήρως τους διαθέσιμους πόρους, με συνέπεια να προσφέρουν διαβάθμιση στο δίκτυο (scalability) και να μειωθούν οι συνέπειες του content bottleneck.

Σκοπός της συγκεκριμένης εργασίας είναι ο σχεδιασμός τέτοιου είδους εφαρμογών, οι οποίες βελτιώνουν τη ποιότητα του streaming ανά τμήματα ενός βίντεο N chunks που διαμοιράζεται ανάμεσα στους 100 κόμβους που υπάρχουν στο δίκτυο. Επίσης στόχος αυτής της εργασίας είναι να βρεθεί μία **μέθοδος μέτρησης της ποιότητας του Streaming**. Αυτό μπορεί να επιτευχθεί υπολογίζοντας ένα ποσοστό για κάθε peer του αριθμού των πακέτων που έχει στη σωστή σειρά και μπορούν να αναπαραχθούν ζωντανά επί του συνόλου των πακέτων του βίντεο. Υπολογίζοντας έναν μέσο όρο των παραπάνω ποσοστών για κάθε peer, μπορούμε να βρούμε ένα συνολικό μέσο όρο για όλους τους peers που θα δείχνει την ποιότητα του Streaming για κάθε εφαρμογή.

Στόχος αυτής της διπλωματικής είναι να μελετήσει αναλυτικά τα χαρακτηριστικά και τα πειραματικά αποτελέσματα του αλγορίθμου Prime, να προσομοιωθούν κάποιες εναλλακτικές μορφές του βασικού αλγορίθμου και να βρεθεί ένα ποσοστό που χαρακτηρίζει την ποιότητα του Streaming για όλους τους peers και για κάθε εφαρμογή. Αρχικά υλοποιήθηκε η Εφαρμογή 2, μία εφαρμογή τύπου server-client, στην οποία κάποιες αρχικές πηγές αποστέλλουν τα κομμάτια/chunks του βίντεο, ενώ ο κάθε κόμβος παρεμβαίνει ζητώντας από τις αρχικές πηγές τα chunks του βίντεο που του λείπουν για την επιτυχή εξέλιξη του streaming. Από την άλλη στην Εφαρμογή 3, χρησιμοποιείται ένα Peer-to-Peer σύστημα για να υλοποιηθεί μία εκδοχή του αλγορίθμου Prime που εστιάζει κυρίως στο streaming. Ουσιαστικά στην εκδοχή του αλγορίθμου Prime που υλοποιήθηκε, η βασική αλγοριθμική διαφορά είναι ότι εφόσον έχει προηγηθεί η φάση diffusion και έχουν διαμοιραστεί κάποια πακέτα του βίντεο σε όλους τους peers, χρησιμοποιείται η φάση του swarming αποκλειστικά για να μπουν τα chunks του βίντεο τμηματικά σε σωστή σειρά, έτσι ώστε να αρχίσει και να συνεχίσει αποτελεσματικά το streaming.

Συγκεκριμένα, χρησιμοποιήθηκε η δυνατότητα που μας δίνει το swarming να ζητηθεί ένα συγκεκριμένο πακέτο του βίντεο από κάποιο peer που το έχει για να βελτιωθεί η ποιότητα του streaming. Η παραπάνω λειτουργία συνδυάστηκε με μια καινοτομία στην υλοποίηση του αλγορίθμου Prime που μέχρι τώρα δεν έχει αναλυθεί λεπτομερώς. Η διαφορά μας έγκειται στο γεγονός ότι επιτρέπουμε στους peers να επικοινωνούν μεταξύ τους, έτσι ώστε όταν κάθε ένας από αυτούς τελειώνει τη φάση diffusion να ενημερώνει όλους τους υπόλοιπους peers, προκειμένου η φάση swarming να αρχίζει μόνο όταν τελειώνει η φάση diffusion σε όλους τους peers. Δηλαδή, σχεδιάστηκε ένας τρόπος έτσι ώστε να διαχωρίζονται οι φάσεις diffusion- swarming. Επίσης, έγινε μία προσπάθεια να αποδειχθούν συνέπειες που έχει η παραπάνω διαφοροποίηση της υλοποίησής μας στο streaming, συγκρίνοντας μία παραλλαγή του αλγορίθμου Prime στην οποία δεν διαχωρίζονται οι φάσεις diffusion- swarming και μία παραλλαγή του αλγορίθμου στον οποίος αυτές οι φάσεις διαχωρίζονται.

Εκτός από την ποιότητα στην εξέλιξη του streaming (μετρώντας πόσα πακέτα του βίντεο είναι στη σωστή σειρά στο σύνολο των πακέτων του βίντεο), μετρήθηκαν και κάποιες άλλες σημαντικές παράμετροι που έχουν σχέση με την αποτελεσματική μετάδοση του βίντεο χωρίς καθυστερήσεις σε κάθε προσομοίωση και δείχνουν παράλληλα κατά πόσο η κάθε εφαρμογή είναι κατάλληλη γι' αυτό το σκοπό. Μερικές από αυτές τις παραμέτρους είναι το chunk delivery time (ο χρόνος για να φτάσουν όλα τα πακέτα του βίντεο σε κάθε peer), τα ίδια πακέτα του βίντεο που φτάνουν παραπάνω από μία φορά σε έναν peer, καθώς και το startup delay (την καθυστέρηση μέχρι να αρχίσει το streaming σε κάθε peer). Όλες οι παραπάνω παράμετροι συμβάλλουν σημαντικά στην ποιότητα του streaming.

Ένα άλλο σημαντικό στοιχείο που διερευνήθηκε είναι κατά πόσο η καλύτερη διαβάθμιση (scalability) του δικτύου επηρεάζει τις παραπάνω παραμέτρους που καθορίζουν την ποιότητα του streaming. Η καλύτερη scalability στις παραπάνω εφαρμογές επιτεύχθηκε προσθέτοντας περισσότερες αρχικές πηγές, από τις οποίες ξεκινάει ο διαμοιρασμός των πακέτων του βίντεο και μοιράζοντας πιο ομοιόμορφα τα πακέτα του βίντεο που έχουν αρχικά οι πηγές.

Επιπλέον ένα άλλο σημαντικό στοιχείο που διερευνήθηκε είναι το κατά πόσο το διαθέσιμο bandwidth που έχουν οι κόμβοι επηρεάζει τις παραπάνω παραμέτρους που καθορίζουν την ποιότητα του streaming. Πιο συγκεκριμένα προσομοιώθηκε μια πραγματική (ADSL) σύνδεση, στην οποία το upload bandwidth είναι συνήθως κατά ¼ μικρότερο από το download bandwidth. Γι' αυτό έγιναν διαφορετικές προσομοιώσεις των παραπάνω εφαρμογών με 2 διαφορετικές κατηγορίες (upload bandwidth- download bandwidth). Αυτές οι 2 κατηγορίες ήταν: 1) (upload bandwidth: 1 Mbps- download bandwidth: 4 Mbps), 2) (upload bandwidth: 2 Mbps- download bandwidth: 8 Mbps). Με τον τρόπο αυτό, δηλαδή, αποδείχθηκε κατά πόσο η μείωση του διαθέσιμου bandwidth δυσκολεύει την εξέλιξη του streaming σε κάθε εφαρμογή.

Ένα από τα σημαντικότερα σημεία στους αλγορίθμους διανομής βίντεο είναι ο τρόπος με τον οποίο αποφασίζεται ποιοι peers θα λάβουν ποια chunks του βίντεο (scheduling). Οι αλγόριθμοι για τη διανομή και το streaming, διαχωρίζονται σε 2 κατηγορίες ανάλογα με το πώς κάνουν το scheduling των chunks του βίντεο. Στην πρώτη κατηγορία ανήκουν οι **push** αλγόριθμοι, στους οποίους οι peers οργανώνονται σε δέντρα διανομής, τα οποία είναι στατικά και μέσα από αυτά μεταφέρονται συνεχόμενα chunks του βίντεο στους peers. Η δεύτερη κατηγορία αλγορίθμων είναι οι **pull** αλγόριθμοι, στους οποίους οι peers οργανώνονται σε ένα γενικό overlay και υπάρχει αρχικά μία trading φάση, στην οποία οι peers ανακοινώνουν σε κάποιο αριθμό γειτόνων τους κάποια από τα πακέτα που έχουν και αντίστοιχα οι γείτονες τους ζητάν όποια από αυτά τα πακέτα χρειάζονται για την εξέλιξη του streaming.

Τέλος έγινε μία προσπάθεια να συγκριθεί σε ένα βαθμό ο αλγόριθμος Prime, ο οποίος είναι ένας τύπου push αλγόριθμος με έναν τύπου pull αλγόριθμο που προσπαθήσαμε να υλοποιήσουμε σε ένα P2P- TV σύστημα. Έτσι είδαμε κάποια χαρακτηριστικά, στα οποία διαφέρει ένας push αλγόριθμος από έναν pull αλγόριθμο. Θα παρατηρήσουμε ότι και σε αυτή την υλοποίηση του pull αλγορίθμου χρησιμοποιήθηκε η φάση trade, για να βοηθηθεί η εξέλιξη του streaming στους peers. Επίσης διερευνήθηκε το κατά πόσο επηρεάζει τα αποτελέσματα ο αριθμός των γειτόνων στους οποίους οι peers ανακοινώνουν τα πακέτα τους, κάνοντας διαφορετικές προσομοιώσεις της συγκεκριμένης εφαρμογής για διαφορετικό αριθμό γειτόνων.

Γενικότερα, μπορούμε να διαπιστώσουμε ότι υλοποιήθηκαν 3 παραλλαγές εφαρμογών διαφορετικών συστημάτων, κοινό σημείο των οποίων είναι ότι εστιάζουν στο streaming. Η Εφαρμογή 2 είναι μια τύπου server- client εφαρμογή, στην οποία οι κόμβοι ζητάν από τις πηγές τα πακέτα που τους χρειάζονται για το streaming, η Εφαρμογή 3 είναι μια παραλλαγή του αλγορίθμου Prime, στην οποία η φάση swarming χρησιμοποιείται για την εξέλιξη του streaming και η τελευταία εφαρμογή του pull αλγορίθμου χρησιμοποιεί επίσης τη φάση trade για τη βελτίωση του streaming. Επομένως στόχος μας ήταν η υλοποίηση κάποιων

διαφορετικών συστημάτων προσαρμοσμένων στην επιτυχή εξέλιξη του streaming και επιπλέον η πραγματοποίηση συγκρίσεων ανάμεσα σε αυτά.

2)ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ PRIME, ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΚΑΙ ΠΑΡΑΜΕΤΡΩΝ ΤΟΥ

2.1) ΓΕΝΙΚΕΣ ΠΡΟΔΙΑΓΡΑΦΕΣ ΑΛΓΟΡΙΘΜΟΥ PRIME

Ο αλγόριθμος Prime είναι ένας διαβαθμισμένου τύπου mesh αλγόριθμος για το streaming ζωντανού περιεχομένου (live content). Υλοποιείται σε ένα mesh-based Peer-to-Peer σύστημα. Ο συγκεκριμένος αλγόριθμος αποτελείται αρχικά από μία φάση push/προώθησης κομματιών, η οποία ονομάζεται φάση διάχυσης/diffusion και μία φάση διαμοιρασμού/pull κομματιών (swarming). Στην συνέχεια θα γίνει αναλυτικότερη αναφορά και στις δύο φάσεις. Στόχος του αλγορίθμου είναι με τη φάση diffusion να αντιμετωπίσει το bandwidth bottleneck, το οποίο παρατηρείται όταν η συχνότητα με την οποία μεταδίδουν οι γείτονες σε έναν peer δεν είναι ικανή για να εκμεταλλευτεί πλήρως ο συγκεκριμένος peer το download bandwidth του. Από την άλλη, η φάση swarming βοηθάει στην αντιμετώπιση του content bottleneck, το οποίο συμβαίνει όταν ένας peer δεν μπορεί να εκμεταλλευτεί πλήρως το download bandwidth του, αφού οι γείτονές του δεν έχουν αρκετό περιεχόμενο για να του μεταδώσουν.

Θα ήταν χρήσιμο τώρα να δούμε πώς θα πρέπει να κατασκευαστεί το overlay στην περίπτωση του αλγορίθμου Prime. Το overlay είναι τύπου mesh και συνδέεται άμεσα με το bandwidth μεταξύ των συνδέσεων και του βαθμού του κάθε peer (degree), δηλαδή του αριθμού των γειτόνων με τους οποίους συνδέεται. Για να αποφύγουμε τη συμφόρηση στις συνδέσεις είναι σημαντικό να ικανοποιείται η συνθήκη του bandwidth- degree, σύμφωνα με την οποία 2 peers όπως ο i και ο j πρέπει να ικανοποιούν τη συνθήκη: bwpf = outbw i/outdeg i = inbw j / indeg j (δηλαδή το bandwidth εισόδου δια τον βαθμό εισόδου θα πρέπει να ισούται με την αντίστοιχη διαίρεση στην έξοδο.) Γενικότερα, όταν ισχύει η παραπάνω συνθήκη μπορούμε να ισχυριστούμε ότι μεταδίδονται δεδομένα στη μέγιστη συχνότητα και χωρίς την παρουσία bandwidth bottleneck. Θα δούμε όμως παρακάτω ότι ο περιορισμός που επιφέρει η παραπάνω αρχή μπορεί να προκαλέσει ορισμένα προβλήματα. Γι' αυτό το λόγο στη δική μας υλοποίηση που θα περιγράψουμε παρακάτω δεν ακολουθήθηκε πιστά αυτή η συνθήκη.

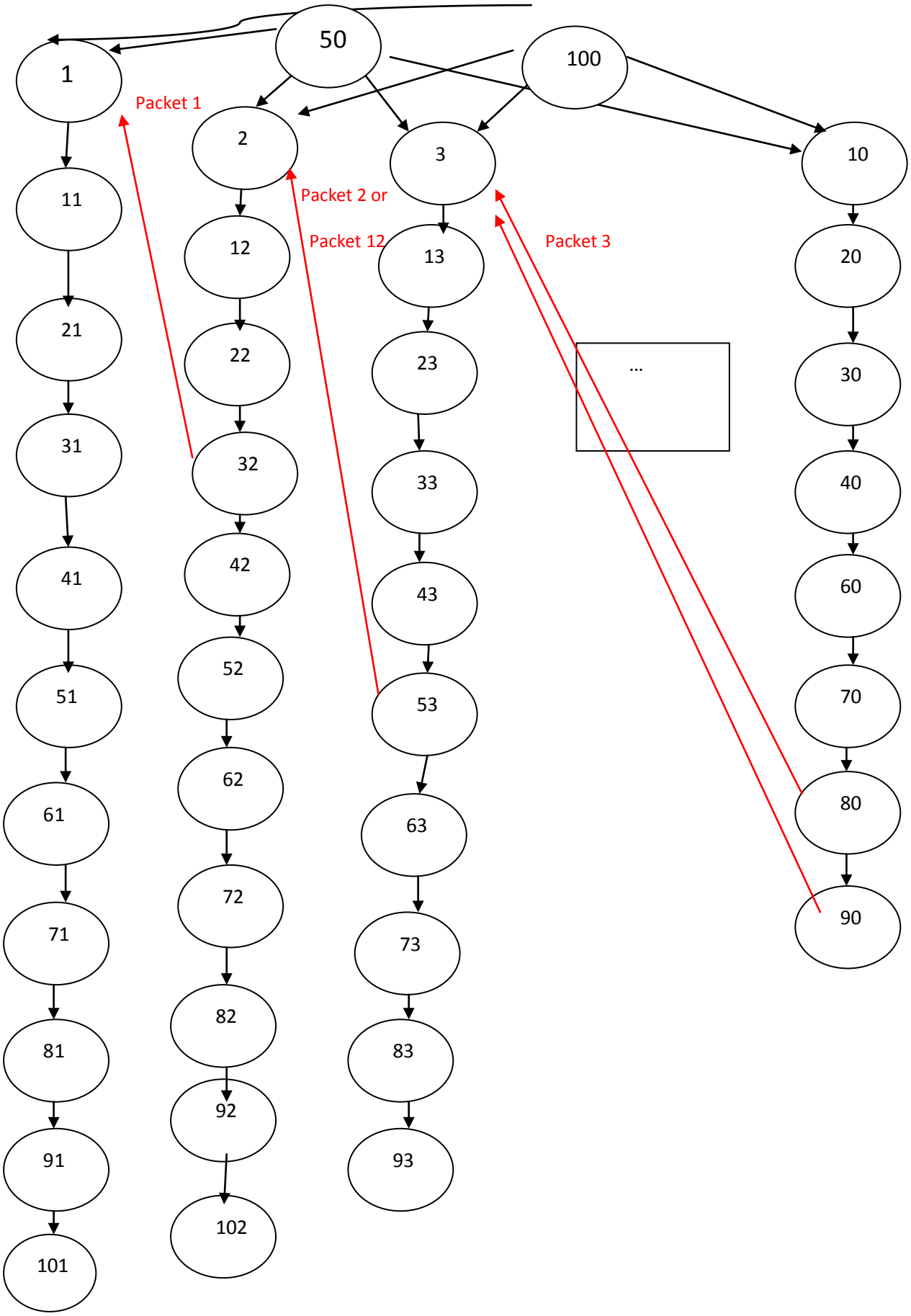
Υποθέτουμε ότι το περιεχόμενο κωδικοποιείται με τον μηχανισμό του Multiple Description Coding (MDC). Με αυτό τον τρόπο ο κάθε peer λαμβάνει ένα περιεχόμενο ποιότητας ανάλογης του download bandwidth του, λαμβάνοντας το κατάλληλο αριθμό description. Στόχος της δικής μας υλοποίησης δεν είναι τόσο η ενασχόληση με την ποιότητα του βίντεο που λαμβάνουν οι peers μέσω του αλγορίθμου Prime, αλλά με την όσο το δυνατόν καλύτερη εξέλιξη του streaming.

2.2) ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ PRIME

ΔΟΜΗ OVERLAY:

Το overlay στη δική μας υλοποίηση έχει τη μορφή που φαίνεται στην παρακάτω εικόνα (σχήμα Α).

Το σύνολο των peers που έχουν την ίδια απόσταση από τις πηγές (peers 50,100), π.χ. n hops από τις πηγές μπορούν να κατηγοριοποιηθούν ως peers του επιπέδου n. Όλοι οι peers είναι ομοιογενείς αφού ίδιο bandwidth εισόδου και ίδιο bandwidth εξόδου (=10Mbps). Βλέπουμε ότι οι peers του $1^{ου}$ επιπέδου έχουν βαθμό εισόδου 2, λόγω των 2 πηγών και βαθμό εξόδου 1. Οι 2 πηγές έχουν βαθμό εξόδου όσο είναι οι peers του $1^{ου}$ επιπέδου (10 peers).


50
100
1
2
3
10
11
12
13
20
21
22
23
30
31
32
33
40
41
42
43
60
51
52
53
70
61
62
63
80
71
72
73
90
81
82
83
91
92
93
101
102
...
Packet 1
Packet 2 or
Packet 12
Packet 3

Οι peers που βρίσκονται σε επίπεδο μεγαλύτερο του 1 είναι ομοιογενείς όσον αφορά τον βαθμό εισόδου και εξόδου. Σε όλους αυτούς τους peers ο βαθμός εισόδου ισούται με τον βαθμό εξόδου (=1), αν δεν λάβουμε υπόψη τα κόκκινα βέλη που υπάρχουν στο παραπάτω σχήμα, την λειτουργία των οποίων θα εξηγήσουμε πιο κάτω. Τέλος οι peers του τελευταίου επιπέδου (n=depth) έχουν έναν μοναδικό πατέρα στο επίπεδο (n-1), ενώ από την άλλη μπορεί να έχουν κάποια παιδιά στο επίπεδο 1 (κόκκινα βέλη).

ΤΡΟΠΟΣ ΜΕΤΑΔΟΣΗΣ ΤΟΥ ΒΙΝΤΕΟ ΣΤΟ OVERLAY ΤΟΥ ΣΧΗΜΑΤΟΣ Α:

Ψάχνουμε να βρούμε έναν τρόπο μετάδοσης του βίντεο ώστε να μειώσουμε τον συνολικό αριθμό των χρονικών διαστημάτων (intervals) που θα χρειαστούμε, βοηθώντας παράλληλα και την εξέλιξη του streaming. Ταυτόχρονα, ο τρόπος που θα επιλεχθεί πρέπει να μειώνει και το content bottleneck μεταξύ των peers. Έτσι επιλέγεται ένας τρόπος μετάδοσης που αποτελείται από 2 φάσεις. Αρχικά, κατά την $1^{η}$ φάση (diffusion) ένα διαφορετικό σύνολο πακέτων μεταδίδεται στους peers που αποτελούν κάθε υποδέντρο. Στη συνέχεια, στη φάση του swarming, οι peers των διαφορετικών υποδέντρων συνεισφέρουν το download bandwidth τους ανταλλάσοντας πακέτα με peers διαφορετικών υποδέντρων του $1^{ου}$ επιπέδου. Ας δούμε τώρα κάποια πράγματα πιο αναλυτικά για τις 2 φάσεις:

1) ***DIFFUSION PHASE:*** Κάθε 1 second οι 2 πηγές μεταδίδουν ένα διαφορετικό σύνολο πακέτων σε κάθε υποδέντρο. Για παράδειγμα το 1ο υποδέντρο αποτελείται από τους peers 1,11,21,31,41,…,101. Αφού τα πακέτα φτάσουν στους peers του $1^{ου}$ επιπέδου, αυτοί προωθούν τα αντίστοιχα πακέτα στους peers του $2^{ου}$ επιπέδου μέσα στο επόμενο χρονικό διάστημα (π.χ. Δ seconds) και συνεχίζεται η ίδια διαδικασία μέχρι τα πακέτα να φτάσουν στο χαμηλότερο επίπεδο του υποδέντρου. Μετά το τέλος της διαδικασίας, όλοι οι peers κάθε υποδέντρου έχουν τα ίδια πακέτα, τα οποία είναι διαφορετικά από αυτά που έχουν οι peers των άλλων υποδέντρων. Το συνολικό χρονικό διάστημα που χρειάζεται η φάση του diffusion εξαρτάται σημαντικά από το βάθος του κάθε υποδέντρου, άρα ο χρόνος για την προώθηση των πακέτων σε 1 υποδέντρο είναι περίπου (depth * Δ seconds). Όλες οι συνδέσεις στο παραπάνω σχήμα που αναπαριστώνται με μαύρα βέλη ονομάζονται diffusion connections και οι αντίστοιχοι πατέρες, diffusion parents. Ο αριθμός των diffusion connections ισούται με το άθροισμα των εξωτερικών βαθμών των 2 πηγών (outdeg(50)+outdeg(100)) με τον αριθμό των συνολικών peers, αν αφαιρέσουμε τον peer του χαμηλότερου επιπέδου για κάθε υποδέντρο (n). Συνολικά, λοιπόν, έχουμε diffusion connections=(outdeg(50)+outdeg(100)) +(peers- peers_level(n))=
   =(10 + 10) + (100- 10)= 110 diffusion συνδέσεις. Επίσης παρατηρούμε ότι ο αριθμός των διαφορετικών υποδέντρων είναι 10 και ταυτίζεται με τον βαθμό εξόδου των δύο πηγών outdeg(50)=outdeg(100)=subtrees=10, άρα 10 είναι και τα διαφορετικά γκρουπ στα οποία έχουμε μοιράσει το σύνολο των πακέτων, αφού όπως ξαναείπαμε κάθε ένα υποδέντρο αποκτά μόνο ένα γκρουπ πακέτων.
   Στη δική μας υλοποίηση όλες οι diffusion connections έχουν ένα default bandwidth (=10Mbps). Παρόλο που το bandwidth είναι αρκετό για να εξελίσσεται γρήγορα το diffusion, λόγω του τρόπου που κατασκευάζει το overlay ο προσομοιωτής (θα αναφερθούμε σε αυτό παρακάτω) πολλές φορές δεν φτάνουν τα ίδια πακέτα σε όλους τους diffusion parents, με συνέπεια ορισμένες diffusion connections να αντιμετωπίζουν content bottleneck στη διάρκεια της φάσης του diffusion.
   Αρκετά σημαντικό ρόλο στην εξέλιξη του diffusion μπορούμε να πούμε ότι παίζουν η συμπεριφορά των πηγών. Όπως ξαναείπαμε, κάθε 1 second οι πηγές στέλνουν πακέτα στους peers του $1^{ου}$ επιπέδου. Μετράμε τη συχνότητα με την οποία οι πηγές στέλνουν πακέτα στους peers του $1^{ου}$ επιπέδου με την παράμετρο (diffusion rate). Το diffusion rate παίρνει τη μέγιστή του τιμή , η οποία είναι ίση με το συνολικό bandwidth των πηγών, στην περίπτωση που οι πηγές στέλνουν διαφορετικά πακέτα σε κάθε

peers του 1ου επιπέδου. Στη δική μας υλοποίηση, εφόσον σε κάθε υποδέντρο στέλνεται ένα διαφορετικό σύνολο πακέτων από τις πηγές, η παραπάνω προϋπόθεση τηρείται, με συνέπεια το diffusion rate να ισούται με το συνολικό bandwidth των πηγών. Έτσι ισχύει ότι diffusion rate=10Mbps.
Παρατηρώντας την δομή των δέντρων που σχηματίζονται στη φάση diffusion, μπορούμε να θεωρήσουμε αυτή τη φάση ως έναν tree-based αλγόριθμο και να τη συγκρίνουμε με τους υπόλοιπους tree-based αλγορίθμους. Η δομή των δέντρων στη φάση του diffusion, όπως και στους tree-based αλγορίθμους, είναι στατική. Από την άλλη, στους mesh-based αλγορίθμους η δομή είναι δυναμική. Αυτό σημαίνει ότι κατά τη διάρκεια του diffusion, αν υπάρχει μία σύνδεση με μικρότερο bandwidth θα έχει σαν συνέπεια να καθυστερήσει η ολοκλήρωση της φάσης diffusion. Αυτό οφείλεται στο ότι δεν υπάρχει επιλογή να μεταφερθούν πακέτα από άλλη σύνδεση, εκτός από αυτή που αποτελεί το συγκεκριμένο υποδέντρο που έχει αναλάβει τη μετάδοση ενός συγκεκριμένου αριθμού πακέτων. Αντίθετα στους mesh-based αλγορίθμους, αν υπάρχει μία σύνδεση με χαμηλό bandwidth μπορούμε να την αντικαταστήσουμε επιλέγοντας ένα διαφορετικό ισοδύναμο μονοπάτι με καλύτερο bandwidth.

2) ***SWARMING PHASE:*** Όπως είδαμε παραπάνω, στο τέλος της φάση diffusion όλοι οι peers έχουν ένα συγκεκριμένο αριθμό πακέτων (30, στη δική μας υλοποίηση). Το ποια πακέτα διαθέτουν εξαρτάται από το υποδέντρο στο οποίο βρίσκονται. Κατά τη διάρκεια της φάσης swarming, οι peers ζητούν πακέτα που τους χρειάζονται για να αρχίσουν το streaming από peers που βρίσκονται σε πιο μικρό επίπεδο (π.χ. επίπεδο 1 στη δική μας υλοποίηση) . Έτσι οι συνδέσεις που αναπαριστώνται με κόκκινα βέλη συνδέουν πατέρες (που βρίσκονται στο επίπεδο j) με παιδιά που βρίσκονται σε επίπεδο i, για το οποίο ισχύει (i>j). Στη δική μας υλοποίηση επιλέγουμε ως σταθερό j=1, δηλαδή οι peers των χαμηλότερων επιπέδων ζητούν πακέτα μόνο από τους peers του 1ου επιπέδου, στη φάση του swarming. Όλες αυτές οι συνδέσεις ονομάζονται swarming connections και οι αντίστοιχοι πατέρες swarming parents.
Στο παραπάνω σχήμα, αν υπήρχε μία swarming connection από τον peer 93 στον peer 3 (swarming parent) αμφότεροι οι συγκεκριμένοι peers μετά τη φάση diffusion θα είχαν τα ίδια πακέτα, με συνέπεια να υπάρχει κίνδυνος να εμφανιστεί content bottleneck. Γι' αυτό το λόγο, στη δική μας υλοποίηση αποφεύγεται η χρησιμοποίηση swarming connections σε ίδια υποδέντρα. Επίσης, υποθέτουμε ότι οι peers του overlay έχουν γνώση για το ποια πακέτα έχει το κάθε υποδέντρο. Έτσι, όταν ένας peer χρειάζεται ένα συγκεκριμένο πακέτο για το streaming, το ζητάει από τον peer του 1ου επιπέδου, ο οποίος ανήκει στο υποδέντρο το οποίο έχει το ζητούμενο πακέτο. Ένα αντίστοιχο παράδειγμα στο παραπάνω σχήμα είναι η swarming connection μεταξύ των peer 53, 2.
Γενικότερα μπορούμε να πούμε ότι η πιθανότητα του content bottleneck εξαρτάται από τους εξής παράγοντες: τον βαθμό εισόδου των peer (1, στη δική μας υλοποίηση), τον αριθμό των diffusion υποδέντρων (10, στην παρούσα υλοποίηση) και τον πληθυσμό των peer στο κατώτερο επίπεδο του κάθε diffusion υποδέντρου (1=in_degree, στη δική μας υλοποίηση).
Μπορούμε να διαχωρίσουμε το συνολικό χρόνο του swarming που χρειάζεται ένας peer για να λάβει όλα τα πακέτα του βίντεο σε ξεχωριστά swarming intervals (K), όπου σε κάθε swarming interval υπάρχει μόνο μία swarming connection. Είναι προφανές ότι το Kmin εξαρτάται άμεσα από τα συνολικά πακέτα που πρέπει να λάβει ένας peer στη φάση του swarming. Στη δική μας υλοποίηση, κατά τη διάρκεια της φάσης swarming, κάθε peer πρέπει να λάβει όλα τα υπόλοιπα πακέτα του βίντεο (N=300 πακέτα) εκτός από αυτά που έλαβε στη φάση του diffusion (300-30=270= swarming_packets=swarming intervals (K)), αριθμός που ταυτίζεται με τα απαιτούμενα swarming intervals. Παρακάτω όμως θα αναφέρουμε και ορισμένους άλλους παράγοντες του συστήματος που επηρεάζουν το Kmin. Μπορούμε να παρατηρήσουμε ότι είναι απαραίτητο ο κάθε peer να έχει έναν buffer, ο οποίος θα μπορέσει να στεγάσει όλα τα βήματα για την ολοκλήρωση του diffusion και όλα τα βήματα για την ολοκλήρωση του swarming. Έτσι πρέπει να ικανοποιείται η παρακάτω συνθήκη:

(depth + Kmin) <= ω, όπου το depth είναι λόγω του diffusion (το βάθος των diffusion trees), το Kmin είναι λόγω του swarming και το ω αναπαριστά το μέγεθος του buffer.
Στην υλοποίηση αυτής της διπλωματικής όλες οι συνδέσεις μεταξύ των κόμβων έχουν το ίδιο bandwidth, και γι' αυτό το λόγο στην φάση swarming όλοι οι peers ζητούν πακέτα ομοιόμορφα από τους peers του 1ου επιπέδου. Σε συστήματα που είναι πιο ανομοιογενές το bandwidth, αν μία swarming connection έχει χαμηλό bandwidth, ο swarming parent θα ζητούσε το πακέτο που ήθελε από κάποιον άλλο peer του ίδιου υποδέντρου (αφού θα είχαν τα ίδια πακέτα λόγω diffusion) με καλύτερο διαθέσιμο bandwidth, αποφεύγοντας έτσι πιθανό content bottleneck. Παρατηρούμε ότι στη φάση diffusion κάτι αντίστοιχο δεν μπορούσε να γίνει γιατί είχαμε μία μοναδική επιλογή για τη ροή των πακέτων.

Παρατηρήσεις Για Τον Αλγόριθμο Prime

Το πρώτο θέμα που θα είχε ενδιαφέρον είναι η συνδεσιμότητα του δικτύου κατά τη δημιουργία των δέντρων του diffusion στο overlay. Ένα αμφιλεγόμενο θέμα είναι η επιλογή του βαθμού (degree) των peers. Αν ο βαθμός των peers είναι μικρός, αυξάνεται το depth των diffusion trees, πράγμα που έχει ως συνέπεια την καθυστέρηση της ολοκλήρωσης του diffusion σε κάθε υποδέντρο. Όπως έχουμε δει το diffusion time για κάθε υποδέντρο είναι περίπου (depth * Δ sec), ποσό που είναι ανάλογο με το depth. Το πιο σημαντικό πρόβλημα όμως που προκαλεί το μικρό peer degree είναι ότι όταν εφαρμόζεται στις πηγές ελαχιστοποιεί τον αριθμό των diffusion υποδέντρων, προκειμένου να ικανοποιείται η συνθήκη bandwidth-degree. Έτσι, κατά τη διάρκεια του swarming, μειώνονται οι επιλογές των peers για να ζητήσουν πακέτα από peers διαφορετικών υποδέντρων, με συνέπεια να αναγκάζονται να ζητούν πακέτα και από peers του ίδιου υποδέντρου, προκαλώντας έτσι content bottleneck. Αντίθετα, αν υπήρχαν περισσότερα υποδέντρα, οι peers θα έπαιρναν τα ζητούμενα πακέτα από διάφορους peers διαφορετικών υποδέντρων, κατανέμοντας πιο ομοιόμορφα το φόρτο του δικτύου. Από την άλλη μεριά, όμως, η απεριόριστη αύξηση του peers degree, μπορεί να επιφέρει αρκετή παραμόρφωση στο δίκτυο. Στη δική μας υλοποίηση, λαμβάνοντας υπόψη τις παραπάνω παρατηρήσεις, χρησιμοποιούμε μεγάλο output degree για τις πηγές (=10), έτσι ώστε να έχουμε αρκετά υποδέντρα (10), μειώνοντας με αυτόν τον τρόπο την πιθανότητα του content bottleneck και μικρό output degree σε όλους τους υπόλοιπους peers (=1), προκειμένου να αποφύγουμε την παραμόρφωση του δικτύου.

Μία άλλη ενδιαφέρουσα παράμετρος που πρέπει να εξετάσουμε είναι ο ελάχιστος αριθμός χρονικών διαστημάτων για την ολοκλήρωση του swarming (Kmin). Ισχύει Kmin= ωmin + depth, όπου ωmin η ελάχιστη απαίτηση buffer στους peers. Μπορούμε να παρατηρήσουμε ότι με την αύξηση του peers degree το Kmin μειώνεται, αφού όπως είπαμε παραπάνω αυξάνεται ο αριθμός των diffusion subtrees, έχοντας περισσότερη ποικιλία για επιλογή swarming parents (peers από τους οποίους θα ζητήσουμε και θα λάβουμε πακέτα στο swarming). Ωστόσο μεγάλη αύξηση του peers degree, αυξάνει και το Kmin, λόγω της παραμόρφωσης στο δίκτυο η οποία προκαλεί content bottleneck . Επίσης, μπορούμε να πούμε ότι το Kmin μειώνεται σημαντικά και με την αύξηση του bandwidth των swarming connections. Τέλος, αποδεικνύεται ότι το ελάχιστο απαιτούμενο buffer των peer (ω min) εξαρτάται άμεσα με το depth των υποδέντρων και το Kmin, λόγω της σχέσης ωmin= depth + Kmin.

Ας εξετάσουμε τώρα μία περίπτωση που θα μπορούσε να εμφανιστεί κατά τη διάρκεια του swarming. Έστω ότι ένας peer χρειάζεται ένα πακέτο, το οποίο έχουν κάποιοι swarming parents του ίδιου υποδέντρου (π.χ. swarming parents του 1ου επιπέδου στη δική μας υλοποίηση). Το πρόβλημα είναι ότι το ζητούμενο πακέτο δεν μπορεί να ζητηθεί από τους παραπάνω swarming parents γιατί έχουν διαθέσει όλο το bandwidth τους για τη μεταφορά άλλων πακέτων κατά τη διάρκεια του swarming. Αυτό το πρόβλημα ονομάζεται αδιέξοδο (deadlock) και προκαλεί content bottleneck. Ένας τρόπος με τον οποίο θα μπορούσε να λυθεί το παραπάνω πρόβλημα είναι αυξάνοντας το Kmin, τα intervals που απαιτούνται για το swarming. Έχοντας περισσότερα

swarming intervals, ένας peer που έχει deadlock event, το αντιμετωπίζει ζητώντας ξανά το ζητούμενο πακέτο από τους ίδιους ή από διαφορετικούς swarming parents στο επόμενο interval, ελπίζοντας πως τώρα οι swarming parents θα έχουν πλέον διαθέσιμο bandwidth για να του μεταφέρουν το ζητούμενο πακέτο.

Τέλος αυξάνοντας το συνολικό αριθμό των κόμβων του overlay κατά την υλοποίηση του αλγορίθμου Prime, αυξάνεται το depth των diffusion subtrees, με συνέπεια να καθυστερεί περισσότερο και η ολοκλήρωση του diffusion. Αντίθετα η αύξηση των κόμβων δεν επηρεάζει τη φάση του swarming, αφού παραμένει ίδιος ο αριθμός των diffusion subtrees. Εξετάζοντας τη σχέση ω min= depth + Kmin, βλέπουμε ότι με την αύξηση των κόμβων, αυξάνονται και οι απαιτήσεις για buffer των peer (ω min), λόγω της αύξησης του depth, ενώ το Kmin παραμένει σταθερό.

Source Scheduling And Swarming Packets- Parents Scheduling

Source Scheduling: Γενικά στον αλγόριθμο Prime, οι πηγές χρησιμοποιούν 2 τακτικές για το scheduling των πακέτων στους peers του 1^{ου} επιπέδου. Πρώτον, χρησιμοποιούν loss detection για τα πακέτα που μεταφέρονται σε κάθε παιδί και κρατά τον έλεγχο των αντιγράφων του κάθε πακέτου που μεταφέρονται. Δεύτερον, οι πηγές αναθέτουν στα πακέτα χρονοσφραγίδες, ανάλογα με τη χρονική στιγμή που λαμβάνουν το κάθε πακέτο. Έτσι κάθε ζητούμενο πακέτο με χρονοσφαγίδα ts πού έχει ήδη μεταφερθεί σε άλλους peers, ανταλλάσσεται (swapped) με το πιο σπάνιο πακέτο με την πιο κοντινή χρονοσφραγίδα μέσα σε ένα recent window [ts – Δ, ts]. Αυτή η δεύτερη τακτική ονομάζεται swapping. Στη δική μας υλοποίηση, προκειμένου να ικανοποιηθεί απόλυτα η συνθήκη κάθε υποδέντρο να έχει διαφορετικά πακέτα, χρησιμοποιήθηκε ένα λίγο πιο διαφορετικό source scheduling. Οι 2 πηγές μας (50, 100) κάθε 1 second, στέλνουν στον κόμβο 1 (αρχή του 1^{ου} υποδέντρου) τα πακέτα (1, 11, 21 ,31, 41, 51, 61, 71,81,91,101,…,291), στον κόμβο 2 τα πακέτα (2,22,32,42,52,62,72,82,92,…,292), στο κόμβο 3 τα πακέτα (3,13,23,33,43,53,63,73,83,…,293), στον κόμβο 4 τα πακέτα (4,14,24,34,44,54,64,74,84,…,294) και συνεχίζουμε με τον ίδιο τρόπο με όλους τους κόμβους του 1^{ου} επιπέδου με τον κόμβο 10, στον οποίο ο πηγές στέλνουν τα πακέτα (10,20,30,40,50,60,70,80,90,100,..,300). Με αυτό το scheduling, είμαστε σίγουροι ότι μετά το τέλος της φάσης diffusion, οι peers που συμμετέχουν σε καθένα από τα 10 υποδέντρα έχουν 30 διαφορετικά πακέτα, πράγμα που ζητάει και ο αλγόριθμος Prime.

Swarming Packets Scheduling: Συνήθως στον αλγόριθμο Prime, οι peers αποφασίζουν για το ποια πακέτα θα ζητήσουν από τους swarming parents είτε με τυχαίο τρόπο, είτε ζητώντας το πιο σπάνιο πακέτο ανάμεσα στους swarming parents (πακέτα που έχουν αρκετή ώρα να μεταφερθούν σε κάποιο peer) εφόσον τα πακέτα λαμβάνουν χρονοσφραγίδες όπως έχουμε πει. **Η διαφορά στη δική μας υλοποίηση έγκειται στο γεγονός ότι οι peers ζητούν το πιο σημαντικό πακέτο που χρειάζονται για να συνεχίσουν αποτελεσματικά το streaming**. Για παράδειγμα, αν ένας peer έχει το πακέτο με identifier 1 και δεν έχει το πακέτο με identifier 2, θα ζητήσει από κάποιο swarming parent το πακέτο με identifier 2, έτσι ώστε να έχει το επόμενο στη σειρά πακέτο και να συνεχίσει το streaming. Με το παραπάνω σκεπτικό, οι peers ζητούν τα πακέτα στα επόμενα intervals του swarming έτσι ώστε να αποκτήσουν τα πακέτα στη σωστή σειρά, μέχρι να αποκτήσουν όλα τα πακέτα του βίντεο.

Swarming Parents Scheduling: Οι swarming parents στον αλγόριθμο Prime, επιλέγονται είτε τυχαία, είτε επιλέγοντας εκείνον τον πατέρα, ο οποίος έχει το μεγαλύτερο διαθέσιμο bandwidth (πράγμα που

σημαίνει ότι δε χρησιμοποιείται αρκετά από άλλους peers ως swarming parent). Στη δική μας υλοποίηση, οι peers επιλέγουν ως swarming parent τον peer του 1^{ου} επιπέδου που έχει το ζητούμενο πακέτο. Όπως είπαμε

παραπάνω, οι peers που βρίσκονται σε καθένα από τα 10 υποδέντρα έχουν 30 διαφορετικά πακέτα. Υποθέτουμε ότι όλοι οι peers του overlay έχουν γνώση για το ποια πακέτα έχουν οι peers κάθε υποδέντρου, έτσι ώστε να ζητούν το πακέτο που χρειάζονται από το σωστό peer του 1ου επιπέδου το ζητούμενο πακέτο, χωρίς να υπάρχουν αστοχίες. Για παράδειγμα, αν κάποιος peer χρειάζεται το πακέτο με identifier 13 θα το ζητήσει από τον peer του 1ου επιπέδου με identifier 3, που έχει αποκτήσει το συγκεκριμένο πακέτο στη φάση diffusion (όπως και όλοι οι peers του ίδιου υποδέντρου).

Παράδειγμα Εκτέλεσης Της Δικής Μας Υλοποίησης Του Prime

Επιλέγουμε τυχαία έναν κόμβο του overlay. Για παράδειγμα, τον κόμβο με identifier 37. Μετά το τέλος της φάσης του diffusion ο συγκεκριμένος κόμβος θα πρέπει να έχει τα εξής 30 πακέτα: (7,27,37,47,57,67,77,87,97,107,117,127,..,197). Αυτό φαίνεται και στο παρακάτω σχήμα.

End Of Diffusion

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	163	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280
281	282	283	283	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300

Στο παραπάνω γράφημα φαίνονται με κίτρινο χρώμα τα πακέτα που έχει αποκτήσει ο κόμβος 37 στη φάση diffusion. Όπως φαίνεται όμως, κανένα από αυτά τα πακέτα δεν είναι στη σωστή σειρά. Έτσι τη δεδομένη χρονική στιγμή είναι αδύνατο να αρχίσει ο συγκεκριμένος κόμβος τη ζωντανή μετάδοση του βίντεο (streaming), αφού δεν έχει τα πακέτα στη σωστή σειρά. Αυτό το πρόβλημα έγινε προσπάθεια να λυθεί στην υλοποίησή μας, χρησιμοποιώντας τη φάση swarming του αλγορίθμου Prime. Ο κόμβος 37, δηλαδή στη φάση swarming θα προσπαθήσει να γεμίσει σειριακά από την αρχή προς το τέλος τα κενά που υπάρχουν στο παραπάνω γράφημα. Έτσι, όταν θα έχουμε κάποια από τα πρώτα πακέτα στη σωστή σειρά, θα μπορούμε να αρχίσουμε το streaming του βίντεο, χωρίς να έχουν φτάσει όλα τα πακέτα του βίντεο.

Ο κόμβος 37, λοιπόν, χρησιμοποιεί τα πρώτα 5 swarming intervals για να πάρει τα πρώτα 5 πακέτα του βίντεο που δεν έχει. Μάλιστα, τα ζητάει και προσπαθεί να τα πάρει με τη σωστή σειρά. Όπως έχουμε ξαναπεί, στην παρούσα υλοποίηση τον ρόλο των swarming parents αναλαμβάνουν αποκλειστικά οι peers του 1ου επιπέδου. Έτσι, ο κόμβος 37 ζητάει από τον κόμβο 1 το πακέτο 1, αφού το πάρει ζητάει από τον κόμβο 2 το πακέτο 2, μετά από τον κόμβο 3 το πακέτο 3, από τον κόμβο 4 το πακέτο 4 και από τον κόμβο 5 το πακέτο 5. Σε αυτό το σημείο, έχουμε το παρακάτω γράφημα για τα πρώτα πακέτα του βίντεο.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Με κόκκινο χρώμα φαίνονται όσα πακέτα έχει αποκτήσει ο κόμβος 37 στη φάση swarming και με κίτρινο χρώμα αυτά που έχει αποκτήσει στη φάση diffusion. Στην υλοποίησή μας θεωρούμε ότι όταν υπάρχουν 5 πακέτα του βίντεο στη σωστή σειρά, μπορεί να αρχίσει η ζωντανή μετάδοσή τους. Έτσι σε αυτό ακριβώς το σημείο αρχίζει το streaming στον κόμβο 37, με τη μετάδοση των πρώτων 5 πακέτων του βίντεο. Όσον αφορά το buffering που πρέπει να κάνει ο κόμβος 37, ξεκινάει από τη χρονική στιγμή που αρχίζει η φάση diffusion μέχρι και τη δεδομένη χρονική στιγμή που αρχίζει το streaming. Αυτή τη σχέση την έχουμε προσδιορίσει θεωρητικά παραπάνω ω= depth + K min , όπου ω το μέγεθος του buffer, depth το βάθος του diffusion tree και K min τα ελάχιστα swarming intervals. Στην περίπτωση του κόμβου 37 το depth των diffusion subtrees που συμμετέχουν είναι 10 και τα ελάχιστα swarming intervals (K min) που χρειάζονται για να αρχίσει το streaming είναι 5, αφού είναι σίγουρο ότι κάθε swarming interval θα είναι πετυχημένο, εφόσον ξέρουμε εξαρχής ότι ο swarming parent έχει το πακέτο που του ζητάμε. Άρα το μέγεθος του buffer που χρειάζεται ο κόμβος 37 για το χρονικό διάστημα από τη στιγμή που αρχίζει η αποστολή του βίντεο με τη φάση diffusion μέχρι να αρχίσει το streaming είναι ω_first= 10 + 5 = 15.

Η φάση του swarming έπειτα συνεχίζεται κανονικά, ζητώντας τα επόμενα στη σειρά πακέτα που λείπουν. Έτσι ο κόμβος 37 ζητάει από τον κόμβο 6 το πακέτο 6, αφού το πάρει ζητάει από τον κόμβο 8 το πακέτο 8, μετά από τον κόμβο 9 το πακέτο 9 και τέλος το πακέτο από τον κόμβο 10.

Σε αυτό το σημείο έχουμε τα επόμενα 5 πακέτα του βίντεο (6,7,8,9,10) και μπορούμε να συνεχίσουμε με τη ζωντανή μετάδοση τους.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Συνεχίζοντας με τη φάση swarming, ο κόμβος 37 ζητάει από τον κόμβο 1 το πακέτο 11, αφού το πάρει ζητάει από τον κόμβο 2 το πακέτο 12, μετά από τον κόμβο 3 το πακέτο 13, από τον κόμβο 4 το πακέτο 14 και από τον κόμβο 5 το πακέτο 15. Παρατηρούμε, ότι οι swarming parents έχουν σίγουρα τα ζητούμενα πακέτα.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Τώρα υπάρχουν διαθέσιμα τα επόμενα 5 πακέτα (11,12,13,14,15) και μπορούν να μεταδοθούν. Παρατηρούμε επίσης ότι το μέγεθος του buffer που χρειάζεται για το χρονικό διάστημα που μεταδίδεται μια πεντάδα πακέτων μέχρι τη μετάδοση της επόμενης πεντάδας είναι ω= K min=5, όσα δηλαδή είναι και τα swarming intervals που απαιτούνται για να φτάσει η επόμενη πεντάδα πακέτων (η οποία θα είναι στη σωστή σειρά).

Στη συνέχεια, ο κόμβος 37 ζητάει από τον κόμβο 6 το πακέτο 16, από τον κόμβο 8 το πακέτο 18 , από τον κόμβο 9 το πακέτο 19 και από τον κόμβο 10 το πακέτο 20. Εφόσον αποκτήσει τα παραπάνω πακέτα στη σωστή σειρά (16,17,18,19,20) (το 17 το έχει ήδη από τη φάση diffusion), μπορεί πλέον να τα μεταδώσει.

Το swarming συνεχίζεται με το παραπάνω τρόπο μέχρι ο κόμβος 37 να αποκτήσει όλα τα πακέτα του βίντεο και μάλιστα στη σωστή σειρά από την αρχή προς το τέλος, έτσι ώστε να συνεχίζεται αποτελεσματικά το streaming. Τους swarming parents αποτελούν αποκλειστικά οι peers (1,2,3,4,5,6,7,8,9,10) του $1^{ου}$ επιπέδου και οι peers ζητούν από τον πλέον κατάλληλο από αυτούς όποιο πακέτο χρειαστούν, εφόσον γνωρίζουν ποια πακέτα έχει αποκτήσει ο καθένας στη φάση diffusion. Παρατηρούμε, τέλος , ότι θα χρειαστούν (300-30=270) swarming intervals για να φτάσουν όλα τα πακέτα του βίντεο στον κόμβο 37, όπου στην παρένθεση αφαιρούμε από τα 300 πακέτα του βίντεο συνολικά τα 30 που έχουν αποκτηθεί στο diffusion.

3) ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΥΛΟΠΟΙΗΣΕΩΝ ΤΩΝ ΕΦΑΡΜΟΓΩΝ

Γενικές Προδιαγραφές Και Κοινά Χαρακτηριστικά Των Εφαρμογών:

Η υλοποίηση των παρακάτω εφαρμογών έγινε στο προσομοιωτή oversim. Όλες οι υλοποιήσεις μας και οι αλλαγές έγιναν αποκλειστικά στο επίπεδο εφαρμογής. Στο επίπεδο δικτύου για τη δημιουργία του overlay χρησιμοποιήσαμε την έτοιμη εφαρμογή του oversim <<MyOverlay>> και πιο συγκεκριμένα το αρχείο <<MyOverlay.h>>. Η κατασκευή του overlay γίνεται σε αυτό το αρχείο, προσθέτοντας κάθε καινούργιο peer σειριακά μετά από αυτόν που έχει δημιουργηθεί τελευταίος, όπως προσθέτονται τα στοιχεία σε μια αλυσίδα FIFO. Έτσι η δομή του overlay μοιάζει με μια αλυσίδα με πρώτο στοιχείο της τον peer που έχει δημιουργηθεί πρώτος και του ανατίθεται το identifier 1 και τελευταίο στοιχείο της τον peer που έχει προστεθεί τελευταίος στο overlay και του ανατίθεται το identifier n, όπου n ο συνολικός αριθμός των peers στο overlay. Με αυτόν τον τρόπο παρατηρούμε ότι στο overlay που σχηματίζεται peers με κοντινά identifiers (π.χ. 1, 2,3) είναι και γείτονες στο overlay. Όταν θέλουμε να αποστείλουμε ένα μήνυμα μέσω overlay καλείται η συνάρτηση << callRoute()>>, την οποία η εφαρμογή <<MyOverlay>> την υλοποιεί ψάχνοντας τον peer, στον οποίο πρέπει να αποσταλεί το μήνυμα με τη τακτική του κοντινότερου γείτονα στο overlay. Για παράδειγμα, όταν ο peer με identifier 1 θέλει να στείλει ένα μήνυμα μέσω overlay στον peer με identifier 4, ο peer 1 στέλνει αναγνωριστικά μηνύματα στους peers με identifiers 2,3 μέχρι να φτάσει και να βρει τον peer με identifier 4. Η παραπάνω τακτική αναζήτησης δείχνει και τον λόγο, για τον οποίο κάποιος peer αργεί να βρει έναν άλλο peer στο overlay που βρίσκεται σε αρκετή μεγάλη απόσταση. Προκειμένου να βρει ο πρώτος peer τον τελευταίο, θα χρειαστεί να στείλει αναγνωριστικά μηνύματα σε όλους τους ενδιάμεσους peers, δηλαδή σε όλους τους υπόλοιπους peer του overlay, γεγονός που θα καθυστερήσει αρκετά την αναζήτηση.

Αρχίσαμε τις προσομοιώσεις μας στον προσομοιωτή oversim υλοποιώντας την **ΕΦΑΡΜΟΓΗ** 1, μία εφαρμογή πολύ απλή στην υλοποίηση της, όπως θα δούμε και παρακάτω, κυρίως για να δούμε τον τρόπο που δουλεύει το oversim.

Όσον αφορά τις υλοποιήσεις των εφαρμογών 2,3 που παρουσιάζουμε παρακάτω, κύριος στόχος μας είναι να βρεθούν μέθοδοι σε μια μετάδοση ενός βίντεο σε κάποιους κόμβους σε ένα overlay, έτσι ώστε να βελτιωθεί όσο το δυνατόν περισσότερο η ποιότητα του streaming και να ελαχιστοποιηθούν στο ελάχιστο οι διακοπές που θα προκληθούν σε αυτό. Γι' αυτόν τον λόγο, θα παρατηρήσουμε ότι και στις δύο εφαρμογές η υλοποίησή μας εστιάζει στο να βρει τα σημαντικότερα πακέτα που λείπουν από κάθε peer έτσι ώστε να συνεχίζει αποτελεσματικά το streaming και να τα ζητήσει από έναν peer που έχει αρκετές πιθανότητες να τα έχει και στις δύο εφαρμογές. Έτσι δύο σημαντικά θέματα τα οποία γίνεται προσπάθεια να λυθούν και στις δύο εφαρμογές είναι τα εξής:

1) Να βρεθεί ποιο είναι το σημαντικότερο πακέτο που λείπει από κάθε peer, έτσι ώστε να συνεχίζεται αποτελεσματικά η μετάδοση του βίντεο, σταδιακά ανά πεντάδες πακέτων.
2) Να βρεθούν κάποιοι peers σε κάθε εφαρμογή που συγκεντρώνουν αρκετές πιθανότητες να διαθέτουν τα ζητούμενα πακέτα. Έτσι για την εφαρμογή 2 επιλέγονται ως τέτοιοι peers οι λεγόμενοι last senders (δηλαδή οι peers που έχουν στείλει τελευταίοι κάποιο πακέτο σε κάθε peer), υποθέτοντας ότι τους last senders αποτελούν οι αρχικές πηγές, αφού από αυτές ξεκινάει η μετάδοση του βίντεο και επομένως συγκεντρώνουν αρκετές πιθανότητες να έχουν τα ζητούμενα πακέτα. Στην εφαρμογή 3, επιλέγουμε ως κατάλληλους peers για να ζητούνται πακέτα, τους peers του 1^{ου} επιπέδου του diffusion tree, για τους οποίους είμαστε σίγουροι ότι έχουν τα ζητούμενα πακέτα, εφόσον τα έχουν αποκτήσει στη φάση diffusion, για την οποία ελέγχουμε ότι έχει τελειώσει για όλους τους peers, όπως θα εξηγήσουμε και παρακάτω.

Ειδικότερα για την υλοποίηση της εφαρμογής 3, στόχος μας είναι να ερευνηθεί το κατά πόσο μπορεί να χρησιμοποιηθεί ο αλγόριθμος Prime έτσι ώστε να βελτιωθεί η ποιότητα του streaming. Γι' αυτόν τον λόγο χρησιμοποιήθηκε αποκλειστικά η φάση του swarming για να ζητούνται πακέτα που είναι σημαντικά για την εξέλιξη του streaming. Ουσιαστικά, δηλαδή το scheduling των ζητούμενων πακέτων στη φάση του swarming γίνεται με κριτήριο την αποτελεσματική εξέλιξη του streaming για κάθε peer. Κάποια επιπλέον χαρακτηριστικά που προστέθηκαν στην υλοποίηση του αλγορίθμου Prime είναι τα εξής :

1) Ο διαχωρισμός των φάσεων diffusion- swarming, έτσι ώστε όταν αρχίζει η φάση του swarming να έχουν φτάσει όλα τα πακέτα της φάσης diffusion. Με αυτόν τον τρόπο γίνεται πολύ αποτελεσματική η φάση του swarming και κατά επέκταση και το streaming, αφού γι' αυτόν τον σκοπό χρησιμοποιείται η φάση swarming.
2) Η επιλογή των swarming parents γίνεται από τους peers, έχοντας τη γνώση για το ποια πακέτα φτάνουν σε ποια υποδέντρα. Έτσι σε κάθε swarming interval καθιστούμε σίγουρο ότι ο swarming parent θα έχει το ζητούμενο πακέτο και θα το στείλει στο peer που το ζητάει, μειώνοντας έτσι τις καθυστερήσεις και τις διακοπές στην εξέλιξη του streaming στον συγκεκριμένο peer.

3.1) Εφαρμογή 1

Η συγκεκριμένη εφαρμογή είναι αυτή που υλοποιήθηκε αρχικά στο oversim. Σε αυτή την εφαρμογή δεν έχει χρησιμοποιηθεί καμία τεχνική για τη βελτίωση της ποιότητας του streaming όπως στις παρακάτω εφαρμογές, γι' αυτό και όλα τα αποτελέσματα σε όλες τις παραμέτρους που καθορίζουν την ποιότητα του streaming είναι πολύ χειρότερα από όλες τις υπόλοιπες εφαρμογές και δεν θα αναλυθούν στο αντίστοιχο κεφάλαιο της ανάλυσης των αποτελεσμάτων. Το μόνο που γίνεται σε αυτή την εφαρμογή είναι ότι υπάρχουν 2 time schedulers, οι ίδιοι που χρησιμοποιούνται και στην πιο κάτω εφαρμογή. Ο πρώτος ο send_period, στέλνει κάθε 1 second ένα send_period event στις αρχικές πηγές, οι οποίες έχουν όλα τα πακέτα του βίντεο και από αυτές θα αρχίσει η αποστολή του video, να επιλέξουν τυχαία έναν κόμβο του δικτύου και να του αποστείλουν όλα τα πακέτα του βίντεο μέσω του overlay. Ο δεύτερος time scheduler είναι ο check_time, ο οποίος στέλνει κάθε 0,4 seconds ένα check_time event σε όλους τους peers για να ελέγξουν πόσα από τα συνολικά πακέτα του βίντεο έχουν λάβει καθώς και πόσα από αυτά βρίσκονται στη σωστή σειρά και μπορούν να παιχτούν (δηλαδή να προχωρήσει αποτελεσματικά το streaming). Ο τρόπος που γίνονται οι παραπάνω έλεγχοι είναι ακριβώς ίδιος με αυτόν που χρησιμοποιείται στην Εφαρμογή 2 με τη μόνη διαφορά ότι όταν οι peers δέχονται ένα check_time event δεν έχουν τη δυνατότητα να βρουν ποιο είναι το σημαντικότερο πακέτο που τους λείπει για να συνεχίσουν την αποτελεσματική εξέλιξη του streaming και να το ζητήσουν από κάποιον peer του overlay που έχει αυξημένες πιθανότητες να το έχει, διαδικασία η οποία γίνεται στην Εφαρμογή 2 κατά τη διάρκεια των check_time events, βελτιώνοντας έτσι σημαντικά τις επιδόσεις στην ποιότητα του streaming.

3.2) Εφαρμογή 2(Σύστημα P2P Που Μοιάζει Με Τη Λειτουργία Ενός Τύπου Servers- Clients Συστήματος)

Ξεκινώντας την περιγραφή της υλοποίησης της εφαρμογής 2 θα ξεκινήσουμε με την περιγραφή της λειτουργίας των 2 time schedulers που χρησιμοποιήθηκαν. Ο πρώτος time scheduler είναι ο send_period, η λειτουργία του οποίου είναι κάθε 1 second να στέλνει ένα send_period event στους αρχικούς αποστολείς, ώστε να επαναλαμβάνουν την αποστολή των κατάλληλων πακέτων. Ο δεύτερος time scheduler είναι ο check_time, ο οποίος έχει στόχο κάθε 0,4 seconds όλοι οι peers να ελέγχουν: α) πόσα πακέτα του βίντεο έχουν λάβει συνολικά, και β) πόσα από αυτά τα πακέτα βρίσκονται στη σωστή σειρά και επομένως μπορούν να παιχτούν. Παρακάτω θα περιγράψουμε πιο αναλυτικά τι γίνεται όταν ένας peer λαμβάνει ένα check_time event.

Συμπεριφορά των πηγών: Όταν κάποια από τις αρχικές πηγές λαμβάνει ένα send_period event επιλέγει με τυχαίο τρόπο έναν κόμβο του δικτύου και του αποστέλλει μέσω του overlay όλα τα πακέτα του βίντεο.

Λήψη πακέτων από το overlay: Όταν ένας peer λαμβάνει ένα πακέτο από το overlay (δηλαδή από τις αρχικές πηγές) αποθηκεύει αρχικά το identifier του πακέτου για να ξέρει ποια πακέτα έχει λάβει. Στη συνέχεια, ελέγχει αν είναι η πρώτη φορά που λαμβάνει το συγκεκριμένο πακέτο ή αν το έχει λάβει ξανά. Αν ένα πακέτο λαμβάνεται για παραπάνω από μία φορά, αυξάνουμε την αντίστοιχη μεταβλητή που έχουμε για τη μέτρηση των διπλών πακέτων. Επίσης στη μεταβλητή last_sender, αποθηκεύεται για κάθε κόμβο το αναγνωριστικό του peer που του έστειλε το τελευταίο πακέτο (πιο πρόσφατο) μέσω overlay.

Check_time Event: Όπως είπαμε και παραπάνω, όταν ένας peer λαμβάνει ένα check time event, ελέγχει αρχικά πόσα πακέτα έχει συνολικά. Την πρώτη φορά που κάποιος peer λαμβάνει check time event ελέγχει εάν έχει τα πρώτα 5 πακέτα του βίντεο. Αν τα έχει, αρχίζει το streaming, μεταδίδοντάς τα και κρατάει ταυτόχρονα τη συγκεκριμένη χρονική στιγμή για να βρεθεί η startup delay (η καθυστέρηση από τη χρονική στιγμή που δημιουργείται το overlay και ζητείται το βίντεο από τους peers μέχρι τη στιγμή που αρχίζει το streaming). Αν όμως κάποιο από αυτά λείπει, κάτι πρέπει να γίνει για να καλυτερεύσει η ποιότητα του streaming. Υπολογίζεται λοιπόν ποιο είναι το πρώτο πακέτο από την πεντάδα πακέτων, στην οποία γίνεται ο έλεγχος, που λείπει. Για παράδειγμα, αν ένας peer έχει τα πακέτα 1,2 και 5 και δεν έχει τα πακέτα 3 και 4 από την πρώτη πεντάδα πακέτων του βίντεο, τότε το πακέτο 3 είναι το πρώτο πακέτο που του λείπει από τα πέντε και αυτό που χρειάζεται πρώτα για να τα βάλει στη σωστή σειρά. Αυτό που κάνει ένας peer, όταν έχει βρει το πρώτο πακέτο από τα πέντε που του λείπει είναι να το ζητήσει από κάποιον άλλο peer που ίσως να το έχει. Ποιοι peers συγκεντρώνουν τις περισσότερες πιθανότητες να έχουν τα ζητούμενα πακέτα; Προφανώς ο peer που του έχει στείλει τελευταίος κάποιο πακέτο και τον ονομάζουμε last_sender. Αν σκεφτούμε πιο προσεκτικά θα δούμε ότι οι last_senders ταυτίζονται με τις αρχικές πηγές του overlay. Άρα εφόσον και οι last_senders αποτελούνται από τους αρχικούς αποστολείς, μπορούμε να θεωρήσουμε αυτούς τους αποστολείς ως servers από τους οποίους στέλνονται αποκλειστικά όλα τα πακέτα του βίντεο στους υπόλοιπους peers του overlay, τους οποίους θεωρούμε ως clients. Αυτός είναι και ο λόγος που, όπως αναφέρεται και στον τίτλο, το συγκεκριμένο σύστημα αν και είναι ένα P2P σύστημα λειτουργεί σαν ένα σύστημα τύπου Servers- Clients. Η ζήτηση κάποιου πακέτου από τους last_senders γίνεται μέσω UDP.

Η ίδια διαδικασία συνεχίζεται και στα επόμενα check time events. Αν κάποιος peer έχει τα πρώτα 5 πακέτα, ελέγχει τα 5 επόμενα και η διαδικασία αυτή συνεχίζεται με αυτόν τον τρόπο, ελέγχοντας αν υπάρχουν τα επόμενα 5 πακέτα κάθε φορά μέχρι να ολοκληρωθεί η μετάδοση ολόκληρου του βίντεο.

Συμπεριφορά των last_senders: Όταν κάποιος last_sender λαμβάνει ένα αίτημα για την αποστολή κάποιου πακέτου του βίντεο, ελέγχει αρχικά αν έχει το συγκεκριμένο πακέτο. Εφόσον το έχει το αποστέλλει απευθείας στον peer που το ζήτησε μέσω UDP.

Λήψη πακέτων μέσω UDP: Ένας peer λαμβάνει κάποιο πακέτο μέσω του UDP μόνο στην περίπτωση που έχει λάβει το συγκεκριμένο πακέτο από κάποιο last_sender. Σε αυτή την περίπτωση, λοιπόν, ακολουθούνται ακριβώς τα ίδια βήματα που κάναμε όταν κάποιος peer λάμβανε κάποιο πακέτο μέσω του overlay και τα περιγράψαμε παραπάνω. Στην ουσία χρησιμοποιείται ο ίδιος ακριβώς κώδικας.

3.3)Εφαρμογή 3 (Υλοποίηση του Αλγορίθμου Prime)

Σε αντίθεση με την Εφαρμογή 2 που είχαμε 2 time schedulers, στην Εφαρμογή 3 έχουμε 3 time schedulers. Ο πρώτος time scheduler είναι και πάλι ο send_period, ο οποίος κάνει ουσιαστικά το scheduling της μετάδοσης των κατάλληλων πακέτων από τις αρχικές πηγές στους peers του 1^{ου} επιπέδου, στην αρχή της φάσης του diffusion. Κάθε 1 second, λοιπόν, στέλνεται ένα send_period event στις αρχικές πηγές να μεταδώσουν μέσω του overlay ένα διαφορετικό σύνολο πακέτων σε κάθε peer του 1^{ου} επιπέδου. Ο δεύτερος time scheduler είναι ο check_timerMsg, ο οποίος όσο διαρκεί η φάση diffusion στέλνει ένα check_timerMsg event κάθε 0,4 seconds σε όλους τους peers, έτσι ώστε να ελέγξουν ποια από τα 30 πακέτα που πρέπει να λάβουν στη φάση diffusion έχουν πάρει μέχρι τη δεδομένη χρονική στιγμή και ποια από αυτά τους λείπουν. Ο τρίτος time scheduler είναι ο swaring_timerMsg και χρησιμοποιείται για τη φάση του swarming. Ο συγκεκριμένος scheduler ενεργοποιείται 0,2 sec πριν τον check_timerMsg και στέλνει ένα swarming_timerMsg event κάθε 0,4 seconds. Όσο η φάση diffusion δεν έχει τελειώσει, όταν οι peers λαμβάνουν ένα swarming_timerMsg event δεν κάνουν απολύτως τίποτα. Από τη στιγμή όμως που η φάση diffusion τελειώνει (θα αναφέρουμε αναλυτικά παρακάτω πώς γίνεται γνωστό σε όλους τους peers ότι τελειώνει η φάση diffusion), όταν κάποιος peer λαμβάνει ένα swarming_timerMsg event κάνει ακριβώς τις ίδιες ενέργειες που έκαναν οι peers σε ένα check_time event στην εφαρμογή 2. Δηλαδή, ελέγχει πόσα πακέτα έχει συνολικά τη δεδομένη χρονική στιγμή και πόσα πακέτα έχει στη σωστή σειρά ανά πεντάδες έτσι ώστε να μπορούν να παιχτούν. Η μόνη διαφορά με το check_time event της εφαρμογής 2 είναι ο peer από τον οποίο ζητούν τα πακέτα που χρειάζονται για το streaming. Ενώ στην εφαρμογή 2 τα ζητούν από τον last_sender, στην εφαρμογή 3 τα ζητούν και πάλι μέσω UDP από τον peer του 1^{ου} επιπέδου του diffusion tree που έχει το ζητούμενο πακέτο (λόγω της εφαρμογής του swarming). Όπως ξέρουμε κατά τη διάρκεια της φάσης του diffusion κάθε peer του 1^{ου} επιπέδου του diffusion tree λαμβάνει ένα διαφορετικό σύνολο πακέτων. Όπως έχει επίσης αναφερθεί, κατά τη διάρκεια του swarming ελέγχονται τα πακέτα ανά πεντάδες και ζητούνται τα πακέτα που λείπουν (από την αρχή προς το τέλος)σε διαδοχικά swarming intervals από τους κατάλληλους swarming parents. Ο λόγος για τον οποίο ζητούνται σειριακά τα πρώτα πακέτα και στη συνέχεια τα επόμενα στη σειρά μέχρι να φτάσουμε στα τελευταία είναι γιατί στη συγκεκριμένη υλοποίηση στόχος μας είναι να εκμεταλλεύεται η φάση swarming, έτσι ώστε να προχωράει όσο το δυνατόν ποιο αποτελεσματικά το streaming. Ουσιαστικά ο ρόλος του swarming είναι ο κάθε peer να αποκτήσει όλα τα πακέτα του βίντεο που του λείπουν αλλά στη σωστή σειρά.

Συμπεριφορά των πηγών: Όταν οι πηγές λαμβάνουν ένα send_period event , στέλνουν μέσω του overlay ένα διαφορετικό σύνολο πακέτων αποκλειστικά στους peers του 1^{ου} επιπέδου του diffusion tree. Ουσιαστικά σε αυτό το σημείο αρχίζει η φάση diffusion, αλλά δεν ολοκληρώνεται.

Λήψη πακέτων από το overlay: Αρχικά όταν ένας peer λαμβάνει ένα πακέτο από το overlay είναι σίγουρος ότι είναι ένα πακέτο που στέλνεται κατά τη διάρκεια της φάσης diffusion, αφού μόνο σε αυτή τη φάση χρησιμοποιείται η μεταφορά πακέτων μέσω overlay. Οπότε, όταν ένας peer λαμβάνει ένα πακέτο μέσω overlay το προωθεί απευθείας (και πάλι μέσω του overlay) στον peer του αμέσως επόμενου επιπέδου που βρίσκεται ακριβώς από κάτω του στο ίδιο υποδέντρο. Για παράδειγμα, κοιτώντας παραπάνω την εικόνα του σχήματος Α της δομής του overlay στη φάση diffusion, οποιοδήποτε πακέτο φτάνει στον κόμβο 61 μέσω overlay το προωθεί απευθείας στον κόμβο 71, ο οποίος βρίσκεται στο αμέσως επόμενο επίπεδο ακριβώς κάτω από τον κόμβο 61. Με τον παραπάνω τρόπο, καταφέραμε να υλοποιήσουμε τη φάση του diffusion και να φτάσουν και τα 30 πακέτα του diffusion σε όλους τους peers.

Επικοινωνία των peers για τον διαχωρισμό των φάσεων Diffusion- Swarming:

Ένα χαρακτηριστικό στοιχείο που έγινε προσπάθεια να επιτευχθεί στην υλοποίηση του αλγορίθμου Prime είναι να διαχωρίζεται η φάση diffusion με τη φάση swarming. Δηλαδή η φάση swarming αρχίζει μόνο όταν έχει τελειώσει η φάση diffusion σε όλους τους peers. Αλλά για ποιο λόγο θα βοηθούσε η κάλυψη της παραπάνω συνθήκης; Ας φανταστούμε την περίπτωση που το swarming ξεκινάει πριν να έχει τελειώσει το diffusion. Κάποιος peer ζητάει κάποιο πακέτο στη φάση swarming από κάποιο swarming parent, για τον οποίο ξέρει ότι θα αποκτήσει το ζητούμενο πακέτο στη φάση diffusion. Λόγω, όμως του γεγονότος, ότι η φάση diffusion δεν έχει τελειώσει ακόμα, υπάρχει η περίπτωση να μην έχει φτάσει τη δεδομένη χρονική στιγμή το ζητούμενο πακέτο. Έτσι χάνεται ένα swarming interval, λόγω του content bottleneck που προκαλεί η παραπάνω κατάσταση. Αυτή ακριβώς η κατάσταση θέλουμε να αποφευχθεί διαχωρίζοντας τις φάσεις diffusion και swarming. Αυτό επιτυγχάνεται με μία επικοινωνία μεταξύ των peers, έτσι ώστε να σιγουρευτούν ότι έχουν όλοι τελειώσει το diffusion.

Όπως είπαμε παραπάνω, όταν ένας peer λαμβάνει ένα check_timerMsg event, ελέγχει την πορεία της φάσης diffusion, δηλαδή πόσα πακέτα από τα συνολικά 30 που πρέπει να λάβει μετά το τέλος της φάσης diffusion έχει τη δεδομένη χρονική στιγμή. Στην περίπτωση όμως που έχει λάβει και τα 30 πακέτα, στέλνει ένα μήνυμα τύπου (DIFFUSION_OVER) σε όλους τους υπόλοιπους peers μέσω UDP. Αποθηκεύουμε τη χρονική στιγμή του peer που τελειώνει τελευταίος τη φάση diffusion,στη μεταβλητή **last diffusion_over time**.

Λήψη μηνυμάτων τύπου (DIFFUSION_OVER) μέσω UDP: Κάθε peer πρέπει να λάβει μηνύματα τύπου (DIFFUSION_OVER) από όλους τους υπόλοιπους peers εκτός από τον εαυτό του. Όταν λάβει ένα τέτοιο μήνυμα και από το τελευταίο peer (n-1), αν n είναι όλοι οι peers, κρατάμε τη συγκεκριμένη χρονική στιγμή ως period_swarm. Όπως είναι λογικό η τιμή period_swarm διαφέρει ελάχιστα για κάθε peer. Αποθηκεύουμε τη χρονική στιγμή του peer που λαμβάνει τελευταίος όλα τα μηνύματα «DIFFUSION_OVER» από όλους τους υπόλοιπους peers του overlay στη μεταβλητή **last period_swarm**.

Λήψη swarming_timerMsg events: Όταν κάποιος peer λαμβάνει swarming_timerMsg event πριν τη χρονική στιγμή period swarm που του έχει ανατεθεί, δεν εκτελεί καμία απολύτως ενέργεια, αφού αυτό σημαίνει ότι δεν έχουν φτάσει οι ειδοποιήσεις από όλους τους υπόλοιπους peers ότι έχουν τελειώσει το diffusion. Όταν κάποιος peer, όμως , λαμβάνει ένα swarming_timerMsg event μετά τη χρονική στιγμή period_swarm (το period_swarm είναι διαφορετικό για κάθε peer), αρχίζει τη φάση swarming. Δηλαδή ελέγχει ποια πακέτα του βίντεο του λείπουν ανά πεντάδες από την αρχή προς το τέλος, όπως ακριβώς στην εφαρμογή 2. Αν έχει μια ολόκληρη πεντάδα πακέτων στη σωστή σειρά αρχίζει τη ζωντανή μετάδοσή τους (streaming), αλλιώς ζητάει το πρώτο πακέτο της πεντάδας που του λείπει μέσω της διαδικασίας του swarming. Δηλαδή ζητάει το πακέτο που του χρειάζεται για το streaming από τον swarming parent του $1^{ου}$ επιπέδου του diffusion tree, που ξέρει ότι έχει αποκτήσει το συγκεκριμένο πακέτο στη φάση του diffusion. Αυτό που είναι σημαντικό είναι ότι ξέρει εξαρχής ότι ο συγκεκριμένος swarming parent θα έχει το ζητούμενο πακέτο, γεγονός που δείχνει πόσο σημαντική είναι η προεργασία που έχει κάνει η φάση diffusion. Με το παραπάνω τρόπο, ο κάθε peer συνεχίζει να ζητάει και να παίρνει σε κάθε swarming interval τα πακέτα του βίντεο στη σωστή σειρά από τον κατάλληλο swarming parent, πετυχαίνοντας έτσι αρκετά καλή ποιότητα στην εξέλιξη του streaming. Η ζήτηση και η αποστολή των πακέτων από τους swarming parents στη φάση του swarming γίνεται και πάλι μέσω UDP. Τέλος, αποθηκεύουμε τη χρονική στιγμή που κάποιος peer λαμβάνει για πρώτη φορά κάποιο swarming_timerMsg event στη μεταβλητή **first swarming_event**. Η μεταβλητή first swarming_event, επομένως, δείχνει τη χρονική στιγμή που κάποιος peer λαμβάνει για πρώτη φορά swarming_timerMsg event, η οποία συμβαίνει μετά τη χρονική στιγμή period swarm για αυτόν τον peer (δηλαδή τη στιγμή που λαμβάνει και το τελευταίο μήνυμα «DIFFUSION_OVER» από όλους τους υπόλοιπους peer του overlay) και αρχίζει για πρώτη φορά τη διαδικασία του swarming.

Λήψη check_timerMsg events: Έχει αναφερθεί ότι όσο διαρκεί η φάση του diffusion, όλοι οι peers παίρνουν κάθε 0,4 seconds check_timerMsg events, ελέγχοντας τη πορεία του diffusion, δηλαδή πόσα και ποιά πακέτα

πρέπει να αποκτήσουν για να συμπληρώσουν τα 30 πακέτα που πρέπει να λάβουν συνολικά στη φάση diffusion. Αν λάβουν και τα 30 πακέτα στέλνουν ένα μήνυμα τύπου (DIFFUSION_OVER) σε όλους τους υπόλοιπος peers. Από τη χρονική στιγμή όμως που κάποιος peer λαμβάνει μηνύματα τύπου (DIFFUSION_OVER) από όλους τους υπόλοιπους peers, η οποία έχει καταχωρηθεί στη μεταβλητή period_swarm, σταματάει να παίρνει check_timerMsg events, αφού έχει πιστοποιηθεί ότι η φάση diffusion έχει τελειώσει για όλους τους peers.

Total Delay: Στις παραπάνω φάσεις τονίστηκε ότι αποθηκεύτηκαν κάποιες σημαντικές χρονικές στιγμές. Αυτό έγινε για να μετρηθεί η καθυστέρηση total delay. Με τη συγκεκριμένη μεταβλητή έγινε προσπάθεια να προσδιοριστεί η καθυστέρηση που επιφέρει στην υλοποίησή μας η απαιτούμενη επικοινωνία μεταξύ των peers, προκειμένου να διαχωρίσουμε τις φάσεις diffusion-swarming. Η συγκεκριμένη καθυστέρηση μετράει τον χρόνο που περνάει από τη στιγμή που τελειώνει το diffusion ο τελευταίος χρονικά peer, μέχρι τη χρονική στιγμή που κάποιος peer στο overlay αρχίζει το swarming (δηλαδή το πρώτο swarming_timerMsg που εκτελεί swarming και δεν είναι αδρανές). Η καθυστέρηση total delay υπολογίζεται, επομένως, αφαιρώντας από τη χρονική στιγμή first swarming_event τη χρονική στιγμή last diffusion_over time (έχουμε εξηγήσει τη σημασία και των δύο χρονικών στιγμών παραπάνω). Άρα, ισχύει **ότι total delay= first swarming_event – last diffusion_over time**. Έχοντας, δημιουργήσει τη μεταβλητή total delay, θα δούμε παρακάτω στην ανάλυση των αποτελεσμάτων το ποσοστό της καθυστέρησης που επιφέρει στον αλγόριθμο Prime, η επικοινωνία των peers που έχουμε δημιουργήσει για να διαχωρίσουμε τις φάσεις diffusion-swarming. Τέλος, θα είναι δυνατό να διαπιστωθεί κατά πόσο η συγκεκριμένη καθυστέρηση είναι σημαντική ή αμελητέα σε σχέση με τα προβλήματα που δημιουργούνται όταν δε διαχωρίζονται οι φάσεις diffusion-swarming.

3.4) Υλοποίηση Pull-Based Αλγορίθμου για P2P-TV Streaming Systems

Ο αλγόριθμος Prime είναι ένας τύπου push αλγόριθμος, όπως και όλοι οι αλγόριθμοι που δημιουργούν διάφορα distribution trees και προωθούν τα δεδομένα μέσα από αυτά. Εκτός από τους τύπου push αλγορίθμους, υπάρχουν και οι τύπου pull αλγόριθμοι για P2P συστήματα. Σε αυτή την κατηγορία αλγορίθμων, προηγείται μια προκαταρκτική trading φάση όπου κάθε peer ανακοινώνει σε έναν αριθμό γειτόνων του κάποια από τα πακέτα που διαθέτει, έτσι ώστε οι γείτονες να επιλέξουν ποιο πακέτο τους χρειάζεται και να το ζητήσουν (γι' αυτό και ονομάζονται pull αλγόριθμοι).

Σε τέτοιου είδους αλγορίθμους 3 είναι οι σημαντικότεροι παράμετροι:

1) Η συχνότητα με την οποία οι peers ανακοινώνουν τα πακέτα που έχουν στους γείτονές τους.
2) Ο αριθμός των γειτόνων στους οποίους κάθε peer ανακοινώνει τα πακέτα του.
3) Ο αριθμός των πακέτων που θα ζητούν οι peers από τους γείτονες που τους έχουν ανακοινώσει πακέτα.

Στη δική μας υλοποίηση για την παράμετρο 1) επιλέγεται η στατική τιμή των 0,3 seconds. Γι' αυτόν τον σκοπό χρησιμοποιήθηκε ο time scheduler (trading_time), ο οποίος στέλνει κάθε 0,3 seconds ένα trading_time event σε όλους τους peers, έτσι ώστε να ανακοινώνουν ποια πακέτα έχουν στους γείτονές τους. Για την παράμετρο 3) επιλέγεται η σταθερή τιμή 1, δηλαδή κάθε peer ζητάει μόνο 1 πακέτο από τους γείτονες που τους έχουν ανακοινώσει πακέτα. Επίσης κάθε peer ανακοινώνει στους γείτονες 3 πακέτα από αυτά που έχει, δηλαδή τα πακέτα που έχει λάβει πιο πρόσφατα. Ο λόγος που επιλέγεται η τιμή 1 για την παράμετρο 3) είναι ότι όλες αυτές οι ανακοινώσεις γίνονται μέσω του overlay και όσο πιο πολλές ανακοινώσεις γίνονται τόσο πιο πολύ συμφόρηση δημιουργείται στο overlay, επιφέροντας μία επιπλέον καθυστέρηση. Για την παράμετρο 2), όπως θα

δούμε παρακάτω στην ανάλυση των αποτελεσμάτων, αυξομειώνεται ο αριθμός των γειτόνων, στους οποίους οι peers ανακοινώνουν πακέτα, για να αποδειχθούν οι συνέπειες που προκαλούνται στα αποτελέσματα.

Πέρα από τον πρώτο time scheduler που περιγράψαμε παραπάνω , υπάρχει και ο time scheduler send_period, ο οποίος υπάρχει και στην εφαρμογή 2. Η λειτουργία του είναι να στέλνει 1 second ένα send_period event στις αρχικές πηγές να επιλέξουν τυχαία έναν peer από το overlay και να του στείλουν όλα τα πακέτα του βίντεο και πάλι μέσω του overlay. Τέλος χρησιμοποιήθηκε και πάλι ο check_time scheduler, ο οποίος κάθε 0,4 seconds στέλνει ένα check_time event σε όλους τους peer για να ελέγξουν πόσα πακέτα του βίντεο έχουν συνολικά, καθώς και πόσες πεντάδες πακέτων υπάρχουν στη σωστή σειρά από την αρχή μέχρι το τέλος, άρα μπορούν και να παιχτούν. Και πάλι το πρώτο πακέτο που λείπει από κάθε πεντάδα ζητείται από κάποιον peer. Αυτό, λοιπόν, το πρώτο πακέτο μιας πεντάδας που μας λείπει συγκρίνεται με τις ανακοινώσεις όλων των πακέτων που μας έχουν στείλει οι γείτονες και αν υπάρχει σε κάποιες από αυτές ζητείται από τον αντίστοιχο γείτονα.

Γενικότερα ο αλγόριθμος εκτελείται με την παρακάτω ροή. Αρχικά κάθε peer ανακοινώνει σε έναν αριθμό γειτόνων του τα 3 πακέτα του βίντεο που έχει λάβει πιο πρόσφατα μέσω overlay από τις αρχικές πηγές. Οι peers που λαμβάνουν αυτές τις ανακοινώσεις ελέγχουν ποιο πακέτο τους λείπει για να συνεχιστεί αποτελεσματικά το streaming και αν αυτό ταυτίζεται με κάποιο από αυτά που τους έχουν ανακοινώσει οι γείτονές τους ότι κατέχουν το ζητούν με ένα μήνυμα τύπου reply στις αρχικές ανακοινώσεις από τον γείτονα που το διαθέτει. Όταν ένας peer λάβει το μήνυμα τύπου reply από κάποιο γείτονα του στέλνει το πακέτο που του ζητάει, βοηθώντας τον έτσι να συνεχίσει το streaming. Ας δούμε τώρα πως υλοποιήθηκε ο αλγόριθμος που μόλις περιγράψαμε.

Λήψη send_period event: Τέτοιου τύπου events στέλνονται στις αρχικές πηγές κάθε 1 second, έτσι ώστε να επιλέγουν τυχαία έναν peer και να του στέλνουν όλα τα πακέτα του βίντεο. Οι αποστολές των πακέτων σε αυτό το επίπεδο γίνονται μέσω του overlay.

Λήψη μηνυμάτων μέσω overlay: Κάθε peer που λαμβάνει ένα μήνυμα μέσω του overlay, σημαίνει ότι αυτό το πακέτο το έχουν στείλει οι αρχικές πηγές, αφού μόνο αυτές στέλνουν πακέτα μέσω του overlay. Αποθηκεύτηκαν τα αναγνωριστικά (identifiers) των 3 πιο πρόσφατων πακέτων που φτάνουν από το overlay σε μια δομή πίνακα, έτσι ώστε να στείλουμε ειδοποιήσεις στους γείτονες με αυτά τα αναγνωριστικά των πακέτων, όταν συμβεί ένα trade_time event. Φυσικά ελέγχεται μέσα σε αυτά τα 3 αναγνωριστικά των πακέτων να μην υπάρχουν ταυτίσεις, έτσι ώστε να μη σταλούν ειδοποιήσεις για ίδια πακέτα στον ίδιο γείτονα.

Λήψη trade_time event: Όταν ένας peer λαμβάνει ένα trade_time event στέλνει ειδοποιήσεις τύπου «VIDEO_DECLARE» με τα αναγνωριστικά των 3 πακέτων που έχει λάβει πιο πρόσφατα (εφόσον έχουμε τα αναγνωριστικά των 3 πακέτων που λαμβάνει πιο πρόσφατα από το overlay) στους γείτονές του (τον αριθμό των γειτόνων τον καθορίζουμε εμείς ανά περίπτωση).

Λήψη ειδοποιήσεων τύπου «VIDEO_DECLARE»: Σε αυτό το σημείο οι peers κρατούν σε μια δομή πίνακα τα αναγνωριστικά των πακέτων που τους αναφέρουν ότι διαθέτουν οι γείτονές τους και σε μια αντίστοιχη δομή πίνακα τους γείτονες που αντιστοιχούν σε κάθε αναγνωριστικό πακέτου, δηλαδή το ποιός γείτονας κατέχει κάθε

πακέτο που έχει αναφερθεί στον συγκεκριμένο peer. Να σημειωθεί ότι τα πεδία των παραπάνω πινάκων ανανεώνονται κάθε φορά που λαμβάνεται ένα καινούργιο μήνυμα τύπου “VIDEO_DECLARE”.

Λήψη check_time event: Όπως και στην εφαρμογή 2, όταν ένας peer λαμβάνει ένα check_time event στην ουσία υποχρεούται να κάνει έναν έλεγχο στη πορεία του streaming. Δηλαδή, ελέγχει πόσα πακέτα κατέχει από τα συνολικά πακέτα του βίντεο, καθώς και πόσα από αυτά είναι στη σωστή σειρά (έτσι ώστε να προχωρήσει το streaming). Ελέγχοντας τα πακέτα του βίντεο ανά πεντάδες από την αρχή μέχρι το τέλος κάθε πεντάδα που βρίσκει συμπληρωμένη και με τα πακέτα της στη σωστή σειρά τη μεταδίδει, αλλιώς ζητάει από κάποιον άλλο peer το πρώτο πακέτο της πεντάδας που του λείπει. Η διαφορά με την εφαρμογή 2 είναι οι peers από τους οποίους ζητούνται τα πακέτα για τη βελτίωση της ποιότητας του streaming. Στην παρούσα εφαρμογή οι peers από τους οποίους ζητούνται πακέτα είναι οι γείτονες που προηγουμένως είχαν στείλει ειδοποιήσεις ότι κατέχουν κάποια πακέτα. Κάθε peer ελέγχει αν το πακέτο που του χρειάζεται για το streaming και θέλει να το ζητήσει ταυτίζεται με τα αναγνωριστικά των πακέτων που έχουν αναφέρει ότι κατέχουν οι γείτονες και τα έχουν αποθηκευτεί σε μια δομή πίνακα όταν λήφθηκαν οι ειδοποιήσεις τύπου “VIDEO_DECLARE”. Αν ναι, ζητάει το συγκεκριμένο πακέτο από τον γείτονα που το κατέχει, ο οποίος μπορεί να βρεθεί και πάλι από τον αντίστοιχο πίνακα που έχουμε δημιουργήσει στη λήψη μηνυμάτων “VIDEO_DECLARE” με τις αντιστοιχήσεις αναγνωριστικών πακέτων στους γείτονες που τα κατέχουν. Αυτά τα αιτήματα για κάποιο πακέτο τα ονομάζουμε μηνύματα τύπου “VIDEO_REQUEST” και στέλνονται μέσω UDP.

Λήψη αιτημάτων τύπου “VIDEO_REQUEST”: Τα συγκεκριμένα αιτήματα “VIDEO_REQUEST” λαμβάνονται μέσω UDP από τους peers που έχουν στείλει αρχικά ειδοποιήσεις σε έναν αριθμό γειτόνων τους με τα 3 πακέτα που έχουν λάβει πιο πρόσφατα. Οι γείτονες που λαμβάνουν αυτές τις αρχικές ειδοποιήσεις από τη μεριά τους ελέγχουν ποιο είναι το σημαντικότερο πακέτο που τους λείπει για την αποτελεσματική εξέλιξη του streaming και αν αυτό ταυτίζεται με κάποιο από αυτά που τους έχουν αναφερθεί στις ειδοποιήσεις που πήραν, το ζητούν από τον γείτονα που το κατέχει με ένα μήνυμα που έχει το ρόλο του reply στις αρχικές ειδοποιήσεις. Τον ρόλο αυτών των reply μηνυμάτων παίζουν τα μηνύματα “VIDEO_REQUEST”. Έτσι όταν ένας peer παίρνει ένα αίτημα τύπου “VIDEO_REQUEST”, στέλνει στον γείτονα που στέλνει αυτό το αίτημα το πακέτο που ζητάει.

4) ΑΝΑΛΥΣΗ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΤΩΝ ΥΛΟΠΟΙΗΣΕΩΝ

4.1) Εισαγωγή

α) Μέθοδοι μέτρησης της ποιότητας του Streaming

Σκοπός μας είναι μέσα από τις διάφορες εφαρμογές που δημιουργήθηκαν να βρεθεί ένας τρόπος να μετρηθεί η ποιότητα του streaming κατά τη διάρκεια μιας ζωντανής μετάδοσης ενός βίντεο. Αυτό θα προσπαθήσουμε να το κάνουμε μελετώντας τα παρακάτω μεγέθη που παίζουν καθοριστικό λόγο για την αποτελεσματικότητα ενός streaming μηχανισμού.

1)Chunk Delivery Time (Ο χρόνος για τη μετάδοση ολόκληρου του βίντεο σε όλους τους peers.)

2)% Average Rate Of Streaming (Για κάθε peer σε κάποια χρονική στιγμή στα μέσα του simulation μετράμε την ποιότητα του streaming, δηλαδή πόσα πακέτα είναι στη σωστή σειρά και μπορούν να παιχτούν επί του συνόλου των πακέτων που έχει λάβει. Στην ουσία βρίσκουμε ένα ποσοστό της

ποιότητας του Streaming για κάθε peer. Από αυτά τα ποσοστά βγάζουμε ένα % μέσο όρο της ποιότητας toustreaming για όλους τους peers.)

3)Average Number Of Double Packets (Για κάθε peer πόσα ίδια (διπλά) πακέτα του βίντεο έχει λάβει κατά τη διάρκεια του simulation και βγάζουμε το μέσο όρο για όλους τους peers.))

4)Average Startup Delay (Για κάθε peer μετράμε τον χρόνο που πρέπει να περιμένει από τη στιγμή που αρχίζει να του στέλνεται το βίντεο μέχρι να μεταδώσει τα πρώτα chunks (δηλαδή να αρχίσει το streaming.)

Απ' εκεί και πέρα ανάλογα με την εφαρμογή που θα μελετάται θα προστίθενται και κάποιες επιπλέον σημαντικές παράμετροι που μας ενδιαφέρουν.

β) Γενικές προδιαγραφές των αρχικών συστημάτων

Όπως αναφέραμε και πριν θέλουμε να υλοποιήθούν κάποια συστήματα για το real time streaming ενός βίντεο που αποτελείται από N chunks (π.χ. 300 chunks στη δική μας προσομοίωση). Ο αριθμός των peers στους οποίους θέλουμε να γίνει η μετάδοση είναι 100. Για όλες τις εφαρμογές που θα περιγραφούν παρακάτω υπάρχει ένα check time που γίνεται κάθε 0,4 seconds κατά τη διάρκεια του simulation. Στόχος του check time είναι σε μια δεδομένη χρονική στιγμή για όλους τους peers να ελέγξει πόσα chunks του βίντεο έχουν λάβει συνολικά και πόσα από αυτά έχουν παίξει. Επίσης σε όλες τις παρακάτω εφαρμογές κατά τη διάρκεια του check time ζητάει από άλλους peers το πιο σημαντικό chunk που του λείπει για να συνεχίσει αποτελεσματικά το streaming. Το από ποιο peer το ζητάει και πώς αλλάζει από εφαρμογή σε εφαρμογή θα το δούμε παρακάτω.

Ακόμη, αρχικά η μετάδοση του βίντεο στους peers γίνεται μέσω του overlay, ενώ οι επαναποστολές chunks για τη βοήθεια του streaming γίνονται μέσω του πρωτοκόλλου UDP.

Το bandwidth που χρησιμοποιούμε αρχικά για όλες τις συνδέσεις είναι αυτό της default περίπτωσης και είναι ίσο με 10 Mbps. Στη συνέχεια προσπαθούμε να προσομοιώσομε μια πραγματική ADSL σύνδεση χρησιμοποιώντας δύο διαφορετικές κατηγορίες bandwidth για τους peers. Η πρώτη κατηγορία έχει download bandwidth=8 Mbps και upload bandwidth= 2 Mbps για κάθε peer. Η δεύτερη κατηγορία παρέχει download bandwidth=4 Mbps και upload bandwidth= 1 Mbps για τους peers.

Τέλος, σε όλες τις παρακάτω προσομοιώσεις υποτέθηκε ότι δεν υπάρχουν απώλειες πακέτων στο overlay, καθώς και η μη ύπαρξη churn.

4.2) ΕΦΑΡΜΟΓΗ 2

α) Αρχική περιγραφή

Σε αυτή την εφαρμογή υπάρχουν κάποιοι servers που έχουν είτε ολόκληρο το βίντεο, είτε κάποια chunks του βίντεο τα οποία θέλουν να στείλουν στους clients. Από την πλευρά τους οι clients θέλοντας να κάνουν streaming ζητούν το πιο σημαντικό chunk που χρειάζονται για να συνεχίσουν τη live μετάδοση του βίντεο. Η αποστολή του βίντεο από τους servers στους clients γίνεται με τον παρακάτω τρόπο. Σε κάθε 1 second κατά τη διάρκεια του simulation όλοι οι servers επιλέγουν με τυχαίο τρόπο έναν client και του στέλνουν (μέσω του overlay) όσα chunks του βίντεο διαθέτουν.

β) Οι διάφορες παραλλαγές της εφαρμογής 2.

1) APP2(F-2S)

Στην πρώτη παραλλαγή υπάρχουν 2 servers (ο 50 και ο 100), που έχουν και οι δύο όλα τα chunks του βίντεο. Οι clients αποτελούνται από όλους τους υπόλοιπους 100 κόμβους από τον 1 μέχρι τον 102. Κατά τη διάρκεια του check time, οι clients βλέπουν ποιο chunk τους χρειάζεται για να συνεχίσουν το streaming και το ζητούν από τον last_sender (τον server που τους έστειλε πιο πρόσφατα ένα chunk του βίντεο). Αυτή η ζήτηση αλλά και η μετάδοση γίνεται μέσω UDP.

2) APP2(N-2S)

Η πρώτη παραλλαγή της application 2 έχει την παρακάτω διαφορά σε σχέση με την παραπάνω $1^{η}$ παραλλαγή. Οι 2 servers (και πάλι ο 50 με τον 100) δεν έχουν όλα τα chunks του βίντεο, αλλά ο $1^{ος}$ server (ο 50) έχει τα μισά chunks του βίντεο (δηλαδή τα chunks από το 1 μέχρι το 150) και ο $2^{ο}$ς (ο 100) έχει τα υπόλοιπα chunks του βίντεο (τα chunks από το 151 μέχρι το 300). Εξαιτίας της παραπάνω αλλαγής, σε αυτή την εφαρμογή μπορούμε να έχουμε την εξής περίπτωση. Όταν στο check time ένας client ζητάει ένα chunk για να συνεχίσει το streaming (π.χ. το chunk 5) και ο last_sender από τον οποίο το ζητάει είναι ο 100, τότε συγκεκριμένος server (100) δεν μπορεί να βοηθήσει τον client, εφόσον δεν έχει τα πρώτα 150 chunks του βίντεο.

3) APP2(4S-MIXED)

Σε αυτή την παραλλαγή υπάρχουν 4 servers (οι 25,50,75,100). Ονομάζεται η συγκεκριμένη παραλλαγή (mixed) γιατί μοιράζονται πιο ομοιόμορφα τα chunks του βίντεο που έχουν οι servers. Έτσι οι 50 και 100 έχουν τα πρώτα 150 chunks του βίντεο και οι 25,75 έχουν τα τελευταία 150 chunks του βίντεο (από 151 μέχρι 300). Εφόσον το overlay που σχηματίζει το oversim έχει τη μορφή μιας αλυσίδας με γείτονες τους κόμβους με συνεχόμενα identifiers (π.χ. ο κόμβος 2 έχει γείτονες τον κόμβο 1 και 3), είναι πιο εύκολο να φτάσουν όλα τα chunks του βίντεο σε όλους τους clients με περισσότερη ευκολία εφόσον οι αποστάσεις με τους servers ελαχιστοποιούνται σημαντικά. Τέλος, μια ακόμα διαφορά της συγκεκριμένης εφαρμογής σε σχέση με τις παραπάνω είναι ότι οι 4 servers έχουν και τον ρόλο του server και τον ρόλο του client. Αυτό γίνεται γιατί οι 4 servers ζητούν και παίρνουν και τα υπόλοιπα chunks του βίντεο εκτός από αυτά που έχουν αρχικά.

4) APP2(4S-NO MIXED)

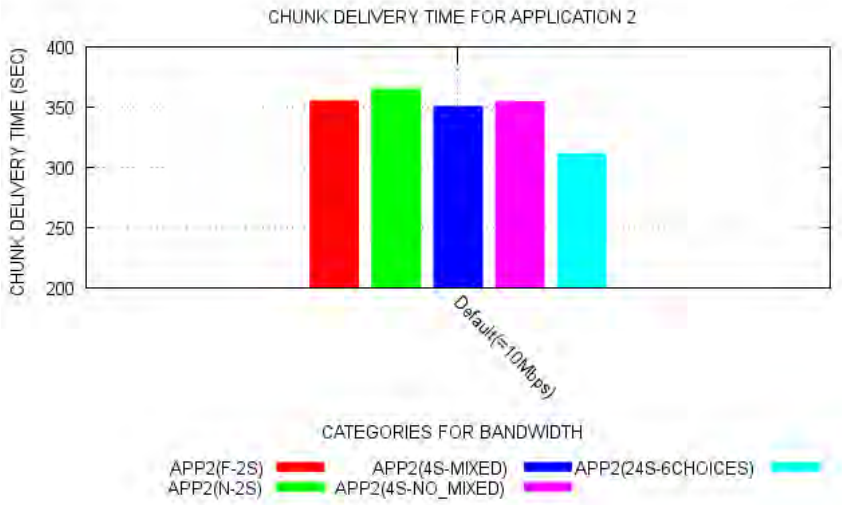
Και πάλι εδώ υπάρχουν οι ίδιοι 4 servers. Εκείνο που αλλάζει είναι τα chunks που έχουν οι servers. Οι 25, 50 έχουν τα πρώτα 150 chunks του βίντεο και οι 75,100 τα 150 τελευταία chunks του βίντεο. Όπως θα δούμε και παρακάτω στην ανάλυση των αποτελεσμάτων είναι αρκετά πιο δύσκολο για τους clients από τον 70 μέχρι τον 102 να έχουν καλές επιδόσεις στο streaming, αφού τα πρώτα 150 chunks του βίντεο που χρειάζονται για να αρχίσει το streaming βρίσκονται σε servers (25,50) που απέχουν σημαντικά στο overlay από τους συγκεκριμένους clients. Τέλος κι εδώ οι 4 servers παίρνουν στο τέλος τα πακέτα του βίντεο που τους λείπουν.

5) APP2(24S-6 CHOICES)

Εδώ υπάρχουν πλέον 24 servers, καθένας από τους οποίους έχει το 1/6 των συνολικών chunks του βίντεο. Κι εδώ οι 24 servers ζητούν και παίρνουν στο τέλος τα πακέτα του βίντεο που τους λείπουν.

γ) ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ 2

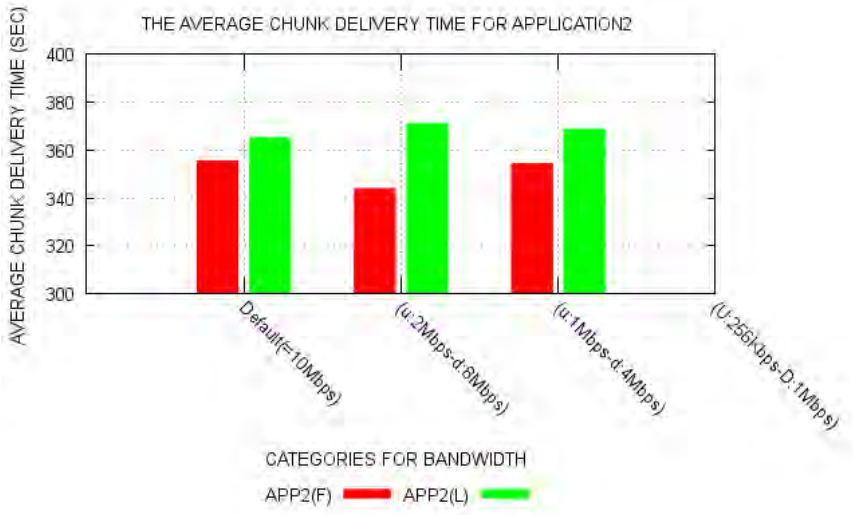
1) chunk delivery time



Στην παραπάνω εικόνα βλέπουμε ότι το μικρότερο chunk delivery time (311.6 seconds) το έχει η APP2(24S-6 CHOICES), αφού η αρχική μετάδοση του βίντεο γίνεται από περισσότερους servers (24), παρόλο που καθένας από αυτούς έχει αρχικά λιγότερα chunks (μόνο το 1/6 των συνολικών chunks του βίντεο). Όταν ο αριθμός των αρχικών servers είναι μεγαλύτερος το δίκτυό μας έχει περισσότερη ευελιξία(flexibility) και μειώνεται η πιθανότητα του content bottleneck. Το content bottleneck συμβαίνει όταν δε χρησιμοποιείται επαρκώς (utilization) το upload bandwidth των server. Το default bandwidth μεταξύ των συνδέσεων όλων των κόμβων στην παραπάνω εικόνα είναι 10Mbps. Η βέλτιστη επίδοση της εφαρμογής με τους 24 servers δείχνει και κάτι άλλο πολύ σημαντικό. Εφόσον έχουμε 24 servers έχουμε και 24 last_senders από τους οποίους ζητούν οι clients τα πακέτα που τους χρειάζονται για το streaming. Έτσι ο φόρτος του δικτύου μοιράζεται σε περισσότερους κόμβους σε σχέση με τις εφαρμογές με τους λιγότερους servers. Από την άλλη η APP2(4S-MIXED) έχει καλύτερη επίδοση στο chunk delivery time (350.8 seconds) σε σχέση με την APP2(4S- NO MIXED)(354.8 seconds), γιατί οι servers που έχουν τα διαφορετικά chunks του βίντεο είναι κατανεμημένοι πιο ομοιόμορφα μέσα στο overlay. Αυτό το γεγονός διευκολύνει στο να φτάνουν πιο γρήγορα όλα τα chunks του βίντεο σε όλους τους clients. Συγκρίνοντας τις δύο πρώτες εφαρμογές, η APP2(F-2S) (355,6 seconds), υπερτερεί σε σχέση με την APP2(N-2S) (365,2 seconds) , γιατί οι 2 servers έχουν όλα τα πακέτα του βίντεο και δεν υπάρχουν αστοχίες όταν ένας client ζητάει από έναν last_sender ένα πακέτο.

Συνολικά, μπορούμε να πούμε ότι καλύτερες επιδόσεις στο chunk delivery time έχουν οι εφαρμογές που χρησιμοποιούν τους περισσότερους servers λόγω του ότι ο φόρτος δικτύου μοιράζεται σε μεγαλύτερο βαθμό ανάμεσα σε περισσότερες συνδέσεις. Παρόλο που μπορούν να υπάρχουν αρκετές αστοχίες, όταν ένας client ζητάει κάποιο πακέτο, εφόσον οι last_senders δεν έχουν όλα τα πακέτα του βίντεο βλέπουμε ότι αυτό δεν επηρεάζει αρκετά αρνητικά το chunk delivery time των εφαρμογών.

Διαφορετικές κατηγορίες bandwidth



Στο παραπάνω γράφημα υπάρχει η παρακάτω αντιστοιχία των εφαρμογών:

- APP2(F)=APP2(F-2S)
- APP2(L)=APP2(N-2S)

Στο παραπάνω γράφημα χρησιμοποιούνται οι παρακάτω κατηγορίες bandwidth:

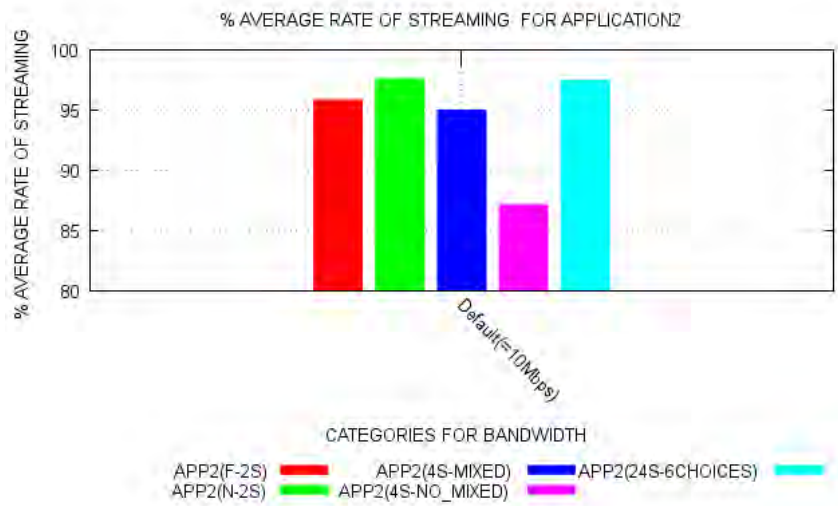
- Default (=10Mbps)

1. (Download=8Mbps , Upload=2Mbps)

2. (Download=4Mbps , Upload=1Mbps)

Συγκρίνοντας σε αυτές τις κατηγορίες τις εφαρμογές APP2(F-2S) και APP2(N-2S), αξίζει να προσέξουμε ότι για την APP2(F-2S) από την κατηγορία 1 (344 seconds) στην κατηγορία 2 (354.4 seconds) χειροτερεύει το chunk delivery time εφόσον στη $2^{η}$ κατηγορία δεν φτάνει πλέον το upload bandwidth και για τους 2 servers να μεταδώσουν όλα τα πακέτα του βίντεο σε όλους τους clients. Για τον παραπάνω λόγο στη συγκεκριμένη εφαρμογή παρατηρούνται μεγαλύτερες αυξομειώσεις στο chunk delivery time μεταβαίνοντας σε διαφορετική κατηγορία bandwidth κάθε φορά. Από την άλλη για την APP2(N-2S) στην $1^{η}$ κατηγορία έχουμε chunk delivery time (371.2 seconds) ενώ στη $2^{η}$ κατηγορία έχουμε (368.8 seconds). Αυτό γίνεται γιατί στη συγκεκριμένη εφαρμογή οι 2 servers έχουν τα μισά πακέτα του βίντεο ο καθένας, άρα δεν χρειάζεται τόσο upload bandwidth για τους 2 servers αυτής της εφαρμογής όσο για αυτούς της πρώτης εφαρμογής. Παρόλα αυτά η διαφορά αυτή μεταξύ των δύο κατηγοριών bandwidth μπορεί να θεωρηθεί αμελητέα και να πούμε ότι το chunk delivery time παραμένει σχεδόν σταθερό σε όλες τις κατηγορίες του bandwidth στη συγκεκριμένη εφαρμογή λόγω καλύτερης scalability στο overlay, αφού από τις 2 αρχικές πηγές η μία μεταδίδει τα μισά πακέτα του βίντεο στους υπόλοιπους κόμβους και η άλλη τα υπόλοιπα μισά.

2) % Average Rate Of Streaming

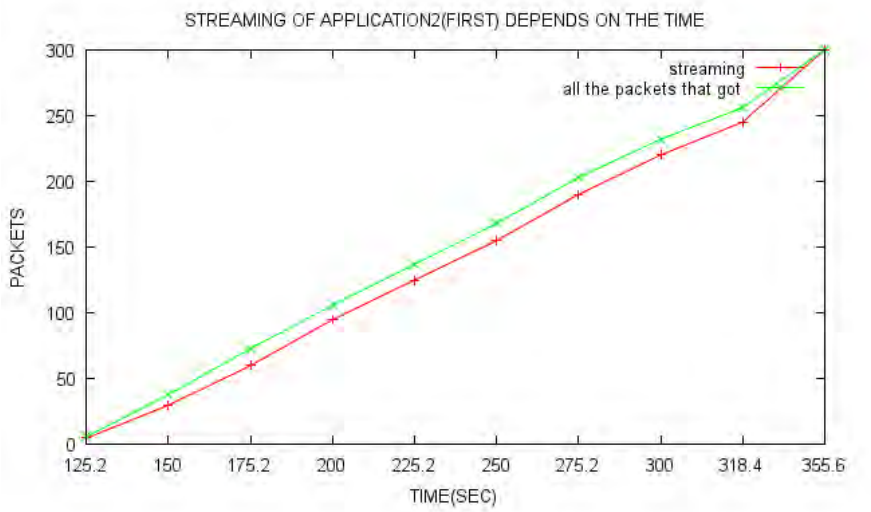


Ενώ για το chunk delivery time, όπως είπαμε και παραπάνω, δεν έχουν και τόση μεγάλη σημασία οι αστοχίες που έχουν οι clients όταν ζητούν πακέτα από τους last_senders, στο streaming αυτό παίζει πιο καθοριστικό ρόλο. Άλλωστε γι' αυτό το λόγο ένας client ζητάει πακέτο από το last_sender, για να έχει καλύτερες επιδόσεις στο streaming. Ακριβώς για τον παραπάνω λόγο η APP2(N-2S) και η APP2(24S-6CHOICES) έχουν τις καλύτερες επιδόσεις στο streaming (97,6%) και (97.49%) αντίστοιχα, εφόσον αρχικά είναι μοιρασμένα τα πακέτα του βίντεο σε 2 και 24 servers αντίστοιχα για τις δύο εφαρμογές (καλύτερη scalability του δικτύου) . Στη συνέχεια έρχεται η APP2(F-2S) (95.87%), αφού οι 2 servers έχουν όλα τα πακέτα του βίντεο, με αποτέλεσμα οι clients να μην έχουν καμία αστοχία όταν ζητούν πακέτα από τους servers. Έπειτα έρχεται η APP2(4S-MIXED)(95.03%), γιατί σε σχέση με την APP2(4S-NO MIXED)(87.21%), η κατανομή των πακέτων που έχουν οι 4 servers είναι πιο ομοιόμορφη στο overlay με αποτέλεσμα να προχωράει και με πιο γρήγορους ρυθμούς και το streaming. Για την εφαρμογή APP2(4S-NO MIXED) έχει ενδιαφέρον ότι οι πρώτοι 51 κόμβοι έχουν average rate of streaming (95,3 %), ενώ οι τελευταίοι 51 κόμβοι έχουν επίδοση (77.39%). Αυτό δικαιολογείται από τη δομή του overlay που έχουμε επιλέξει και είναι η παρακάτω: οι 2 servers που βρίσκονται ανάμεσα στους πρώτους 51 κόμβους (οι 25, 50) έχουν τα πρώτα 150 πακέτα του βίντεο με συνέπεια να βοηθιούνται οι κοντινοί clients στο streaming. Από την άλλη οι τελευταίοι 51 κόμβοι έχουν ανάμεσα τους 2 servers (τους 75,100) που έχουν τα τελευταία 150 πακέτα του βίντεο, ενώ βρίσκονται σε αρκετά μακρινή απόσταση από τους servers που έχουν τα πρώτα 151 πακέτα του βίντεο, με συνέπεια να μειονεκτούν στην ποιότητα του streaming. Συμπερασματικά μπορούμε να πούμε ότι καλύτερες επιδόσεις έχουμε στις εφαρμογές που λόγω της δομής του overlay που έχουμε επιλέξει, συνδυάζουμε καλή scalability και λιγότερες αστοχίες.

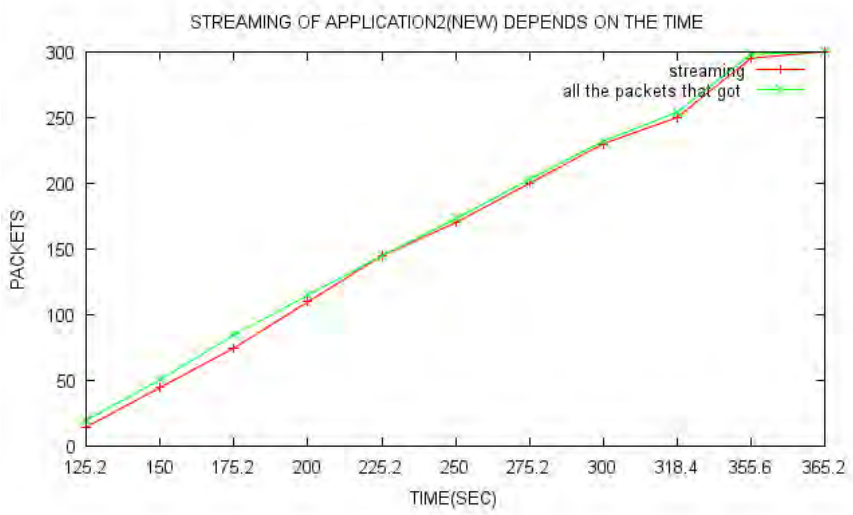
Γραφήματα του Streaming στη πορεία του χρόνου

Είδαμε παραπάνω το % Average rate of streaming για όλες τις εφαρμογές. Παρακάτω για κάθε εφαρμογή θα δούμε ένα γράφημα που θα μας δείχνει πόσα πακέτα έχει λάβει ο κόμβος 21 και πόσα έχουν παίξει σε δεδομένες χρονικές στιγμές. Με αυτό τον τρόπο θα εξεταστεί η εξέλιξη του streaming στη πορεία του χρόνου. Συγκρίνοντας τις εφαρμογές APP2(F-2S) και APP2(N-2S) θα φανεί και στα παρακάτω γραφήματα η διαφορά στην ποιότητα του streaming.

1)APP2(F-2S)



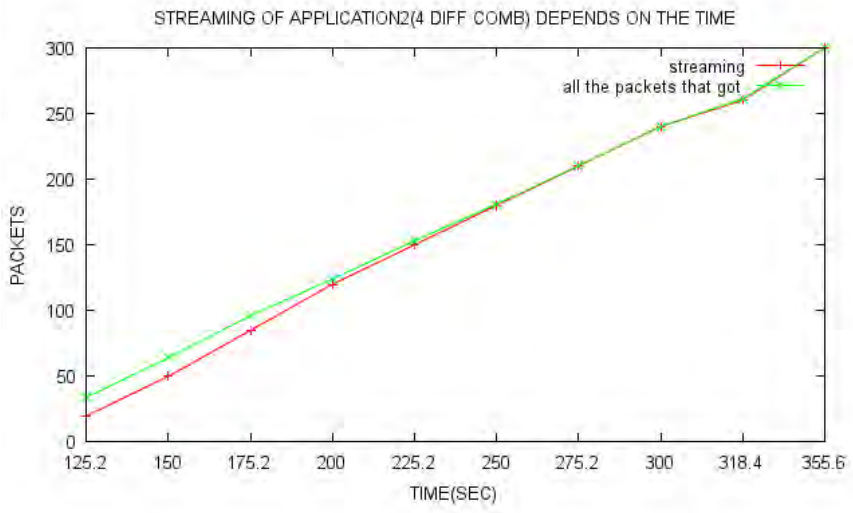
2) APP2(N-2S)



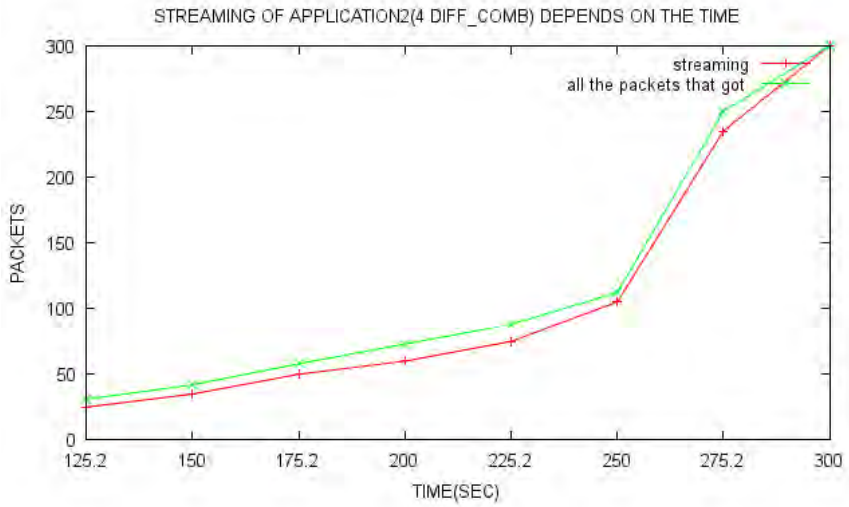
3)APP2(4S- NO MIXED)

Εδώ θα δείξουμε την πορεία του streaming σε 2 κόμβους (στον 21 και στον 75). Θα παρατηρήσουμε ότι ο κόμβος 21 έχει καλύτερη εξέλιξη του streaming στην πορεία του χρόνου αφού οι πιο κοντινοί του servers (25 και 50) έχουν τα πρώτα 151 πακέτα του βίντεο. Έτσι ο κόμβος 21 ξεκινάει πιο αποτελεσματικά τη live μετάδοση του βίντεο σε σχέση με τον κόμβο 75.

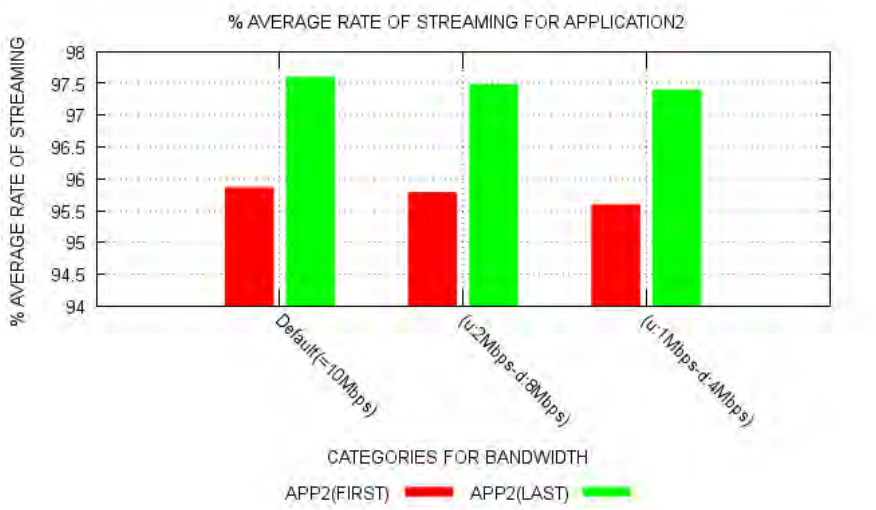
ΚΟΜΒΟΣ 21



ΚΟΜΒΟΣ 75


STREAMING OF APPLICATION2(4 DIFF_COMB) DEPENDS ON THE TIME
PACKETS
TIME(SEC)
streaming
all the packets that got
0
50
100
150
200
250
300
125.2
150
175.2
200
225.2
250
275.2
300

Διαφορετικές Κατηγορίες Bandwidth



Στο παραπάνω γράφημα υπάρχει η παρακάτω αντιστοιχία των εφαρμογών:

- APP2(F)=APP2(F-2S)
- APP2(L)=APP2(N-2S)

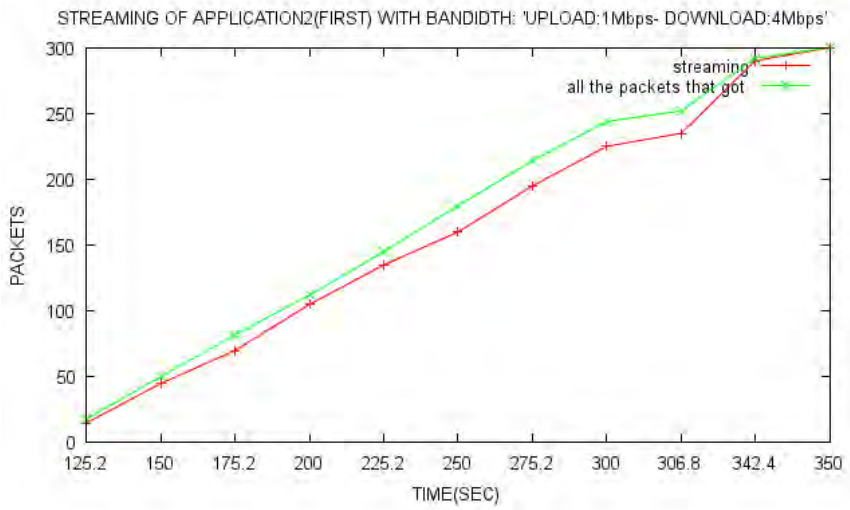
Για τις διαφορετικές κατηγορίες bandwidth που αναφέραμε παραπάνω, η APP2(N-2S) έχει καλύτερες επιδόσεις στο streaming σε σχέση με την APP2(F-2S) στο default bandwidth για τους λόγους που είπαμε παραπάνω. Από εκεί και πέρα βλέπουμε ότι όσο μειώνουμε το bandwidth οι επιδόσεις και των δύο εφαρμογών μειώνονται ανάλογα. Αξιοσημείωτο είναι επίσης ότι στην εφαρμογή APP2(N-2S) παρόλο τις μειώσεις του bandwidth η ποιότητα του streaming παραμένει σχεδόν σταθερή. Αυτό οφείλεται κυρίως στην καλύτερη scalability της συγκεκριμένης εφαρμογής.

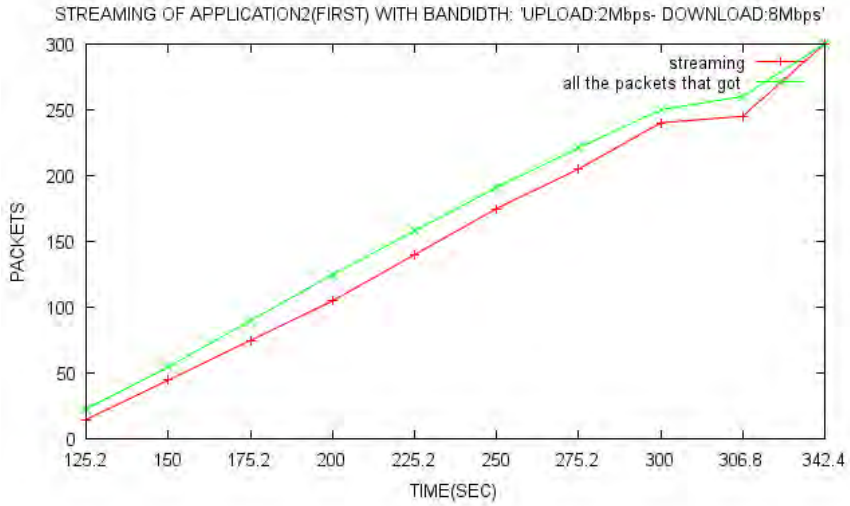
Γραφήματα του Streaming στην πορεία του χρόνου

1)APP2(F-2S)

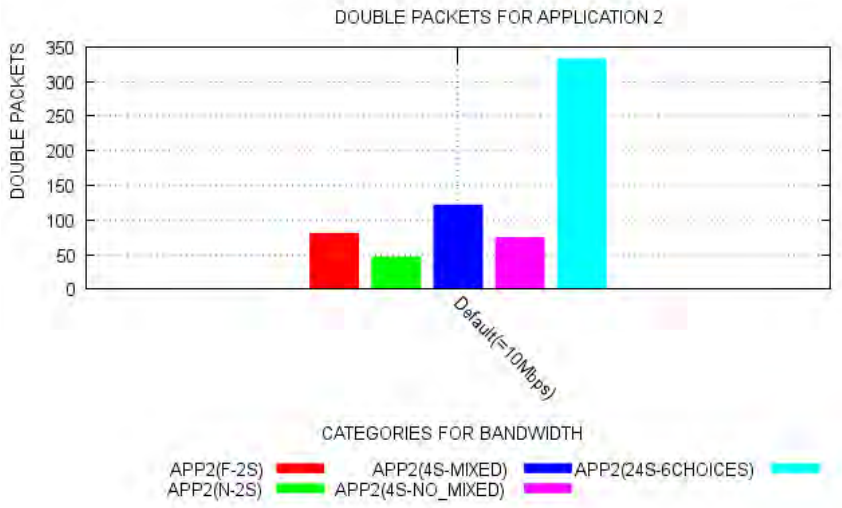
Συγκρίνοντας αυτή την εφαρμογή στις 2 κατηγορίες bandwidth παρατηρούμε ότι στην 1^{η} κατηγορία bandwidth (upload: 2Mbps- download: 8Mbps) η γραφική παράσταση της εξέλιξης του streaming στην εξέλιξη του χρόνου είναι πιο σταθερή (σχεδόν μία ευθεία γραμμή). Αντίθετα όταν μειώνουμε το bandwidth στη 2^{η} κατηγορία

bandwidth (upload: 1Mbps- download: 4Mbps) βλέπουμε ότι η γραφική παράσταση έχει περισσότερες καμπυλώσεις λόγω της αστάθειας που προκαλεί στο σύστημα η μείωση του bandwidth.



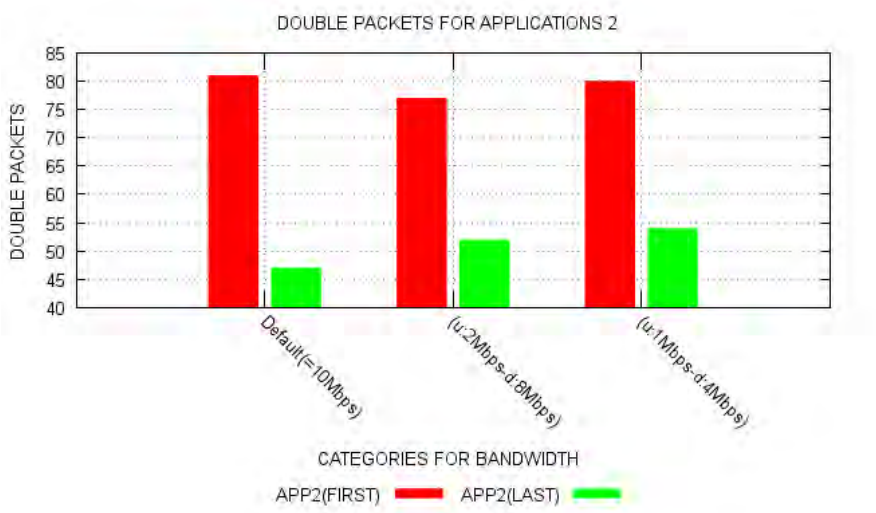


3) Average Number Of Double Packets



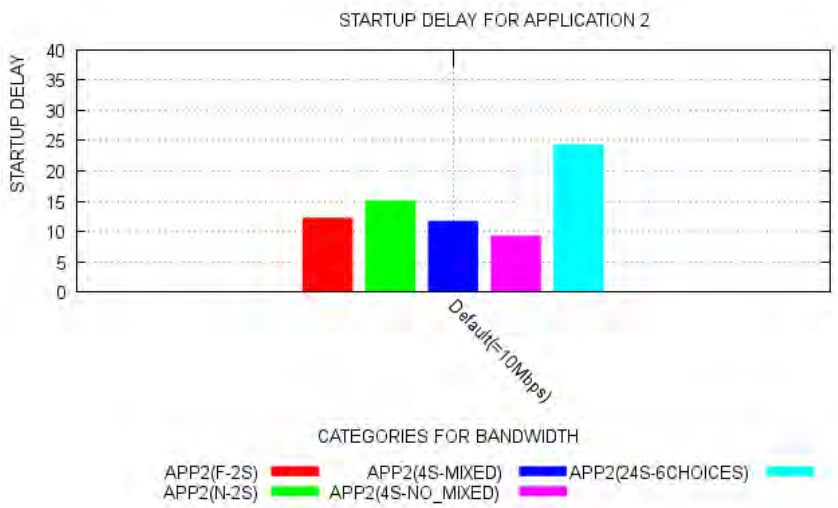
Από την παραπάνω εικόνα βλέπουμε ότι τα λιγότερα διπλά πακέτα τα έχει η APP2(N-2S)(42 πακέτα), λόγω του ότι υπάρχουν μόνο 2 servers οι οποίοι έχουν τα μισά πακέτα του βίντεο ο καθένας. Έτσι κάθε 1 sec που ο κάθε server επιλέγει τυχαία έναν client και του στέλνει όλα τα πακέτα που έχει, δεν τροφοδοτεί τους clients με πολλά διπλά πακέτα. Στην APP2(F-2S) (81 διπλά πακέτα) πάλι έχουμε μόνο 2 servers αλλά και οι 2 έχουν όλα τα πακέτα του βίντεο, τα οποία στέλνουν στους ίδιους clients με συνέπεια να τροφοδοτούνται με περισσότερα διπλά πακέτα. Έπειτα οι εφαρμογές APP2(4S-MIXED)(122 διπλά πακέτα) και APP2(4S-NO MIXED)(75 διπλά πακέτα) με τους 4 servers έχουν περισσότερα διπλά πακέτα από αυτές με τους 2 servers. Τέλος η APP2(24S-6CHOICES)(333 διπλά πακέτα) που έχει πολύ περισσότερους servers (24) από όλες τις υπόλοιπες εφαρμογές, έχει τα περισσότερα διπλά πακέτα. Συμπερασματικά μπορούμε να πούμε ότι δύο λόγοι που συμβάλλουν στην αύξηση των διπλών πακέτων είναι ο μεγαλύτερος αριθμός των servers που έχουμε και ο μεγαλύτερος αριθμός των πακέτων του βίντεο που έχουν αρχικά οι servers.

Διαφορετικές Κατηγορίες Bandwidth



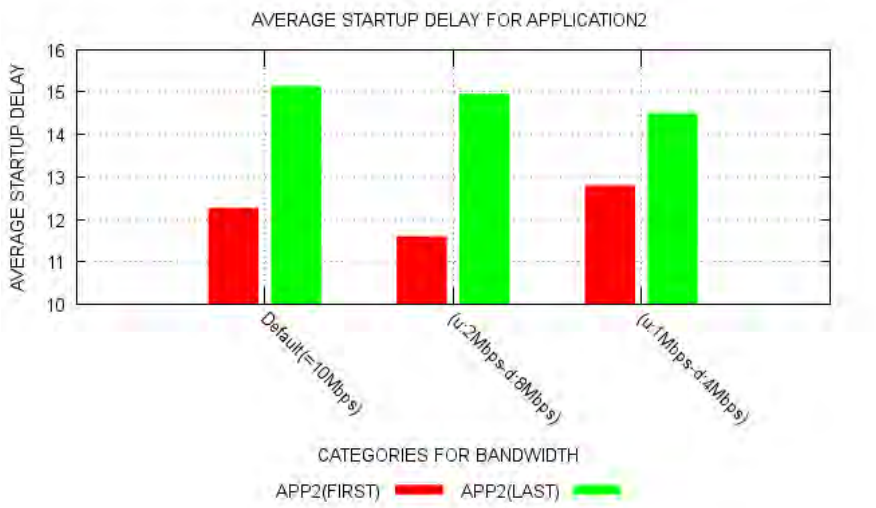
Παρατηρούμε ότι για την APP2(F-2S) μειώνονται για λίγο τα διπλά πακέτα στην 1η κατηγορία bandwidth σε σχέση με τη default. Αυτό συμβαίνει λογικά γιατί έχουμε άλλο bandwidth για upload και άλλο για download με συνέπεια να μην φτάνουν στους clients όλα τα διπλά πακέτα που έφταναν στη default περίπτωση. Γενικότερα, όσο μειώνουμε το bandwidth τόσο περισσότερα διπλά πακέτα έχουμε και στις δύο εφαρμογές. Αυτό συμβαίνει γιατί μειώνοντας το bandwidth δυσκολεύει η επικοινωνία μεταξύ των 2 αρχικών αποστολέων και των μακρινών peer στο overlay. Έτσι οι αρχικοί αποστολείς επικοινωνούν πολύ πιο συχνά με τους κοντινούς peers με συνέπεια να τροφοδοτούνται και με περισσότερα διπλά πακέτα.

4) Average Startup Delay



Το μικρότερο startup delay το έχει η εφαρμογή APP2(4S-NO MIXED)(9.29seconds)(λόγω της δομής του overlay), αφού οι πρώτοι 51 clients που έχουν κοντά τους 2 servers έχουν τα πρώτα 151 πακέτα του βίντεο, με αποτέλεσμα να παίρνουν γρήγορα τα πρώτα πακέτα του βίντεο και να αρχίζουν να το μεταδίδουν. Στη συνέχεια η APP2(4S-MIXED)(11.76 seconds) με τους 4 servers έχει μικρότερο startup delay από τις εφαρμογές με 2 servers, αφού η ζήτηση για πακέτα που χρειάζονται οι clients για τν εξέλιξη του streaming κατανέμεται σε περισσότερους last senders. Η APP2(F-2S)(12.27 seconds) έχει καλύτερη επίδοση από την APP2(N-2S)(15.14 seconds) γιατί οι 2 servers της έχουν όλα τα πακέτα του βίντεο με συνέπεια να μην υπάρχουν αστοχίες όταν 1 client ζητάει το πακέτο που του χρειάζεται για να αρχίσει το streaming. Τέλος η APP2(24S-6CHOICES)(24.37 seconds), έχει το μεγαλύτερο startup delay αφού ο κάθε client είναι δύσκολο να πετύχει ως last sender τον server με τα κατάλληλα πακέτα του βίντεο που χρειάζεται για να αρχίσει το streaming. Επομένως, ο αρχικός διαμοιρασμός των πακέτων του βίντεο σε περισσότερους κόμβους δημιουργεί πρόβλημα στο να εντοπιστεί το σωστό πακέτο που χρειαζόμαστε για το streaming. Άρα και εδώ έχουμε πολλές αστοχίες.

Διαφορετικές Κατηγορίες Bandwidth



Στο παραπάνω γράφημα υπάρχει η παρακάτω αντιστοιχία των εφαρμογών:

- APP2(F)=APP2(F-2S)
- APP2(L)=APP2(N-2S)

Από το default στην 1η κατηγορία bandwidth και για τις 2 εφαρμογές μειώνεται το startup delay, λόγω του ότι έχουμε διαφορετικό bandwidth για upload και διαφορετικό για download (περισσότερο σε σχέση με το upload που μας συμφέρει). Από την 1η κατηγορία bandwidth όμως στη 2η , ενώ για την APP2(F-2S) αυξάνεται σημαντικά το startup delay όταν μειώνουμε το bandwidth (αυτή η καθυστέρηση οφείλεται κυρίως στη μεγαλύτερη συμφόρηση που προκαλείται εξαιτίας του γεγονότος ότι οι αρχικοί servers έχουν όλα τα πακέτα του βίντεο), για την APP2(N-2S) το startup delay μειώνεται ελάχιστα από 14.96seconds σε 14.5 seconds, ποσότητα που είναι σχεδόν αμελητέα. Συνολικά παρατηρούμε για την εφαρμογή APP2(N-2S) ότι το startup delay παραμένει σταθερό με τη μείωση του bandwidth, λόγω της καλύτερης scalability της εφαρμογής.

4.3) ΕΦΑΡΜΟΓΗ 3 (Συστήματα P2P- Αλγόριθμος Prime)

Οι διάφορες παραλλαγές της Εφαρμογής 3.

1) App3F(2S)

Σε αυτή την εφαρμογή υπάρχουν 2 αρχικοί αποστολείς (οι 50, 100), που έχουν και οι δύο ολόκληρο το βίντεο. Κατά τη φάση του diffusion αναλαμβάνουν αυτοί οι δύο peers αποκλειστικά να στείλουν τα κατάλληλα 30 πακέτα σε κάθε peer ανάλογα με τον κλάδο στον οποίο βρίσκονται. Αυτή η διαδικασία προκαλεί μεγάλη

κατανάλωση στο upload bandwidth αυτών των δύο peers και τεράστιο φόρτο στο δίκτυο, όπως θα διαπιστώσουμε και στην ανάλυση των αποτελεσμάτων. Επίσης, στην παρούσα εφαρμογή δε διαχωρίζεται η φάση diffusion με τη φάση του swarming. Ουσιαστικά το swarming αρχίζει από το πρώτο check time, άσχετα με το αν το πακέτο που ζητείται μπορεί να μην υπάρχει στο swarming parent, εφόσον η φάση του diffusion δεν έχει τελειώσει ακόμα και οι peers δεν έχουν τη δεδομένη χρονική στιγμή και τα 30 πακέτα που θα αποκτήσουν μετά το τέλος της φάσης του diffusion. Έτσι , σε αυτή την εφαρμογή έχουμε τη παραπάνω μορφή αστοχίας που δημιουργεί κάποια προβλήματα όπως θα δούμε και παρακάτω στην ανάλυση των αποτελεσμάτων.

2)App3F(4S)

Εδώ έχουμε την ίδια εφαρμογή με την παραπάνω. Η μόνη διαφορά είναι ότι τώρα έχουμε 4 peers (οι 25,50,75,100) που έχουν κάποια πακέτα του βίντεο ο καθένας κι από αυτούς αρχίζει η αποστολή του βίντεο στους υπόλοιπους peers. Συγκεκριμένα οι peers (50,100) έχουν τα πρώτα 150 πακέτα του βίντεο, ενώ οι peers (25,75) έχουν τα τελευταία 150 πακέτα του βίντεο.

3)App3(D-S)(2S)

Κι εδώ υπάρχουν 2 peers (50, 100) πού έχουν ολόκληρο το βίντεο αρχικά. Κατά τη φάση του diffusion και εδώ αναλαμβάνουν αυτοί οι δύο peers αποκλειστικά να στείλουν τα κατάλληλα 30 πακέτα σε κάθε peer ανάλογα με τον κλάδο στον οποίο βρίσκονται. Η διαφορά που έχει η συγκεκριμένη εφαρμογή με την εφαρμογή 1 είναι ότι οι φάσεις diffusion και swarming πλέον διαχωρίζονται. Δηλαδή η φάση swarming αρχίζει μόνο όταν τελειώσει η φάση diffusion. Έτσι στη φάση swarming όλοι οι peers έχουν τα 30 απαραίτητα πακέτα, με συνέπεια να μην υπάρχουν πλέον αστοχίες όταν ζητούνται πακέτα σε αυτή τη φάση. Ο διαχωρισμός αυτός των φάσεων γίνεται με κάποιο είδος επικοινωνίας μεταξύ των peers. Όταν, λοιπόν, κάθε peer αποκτήσει τα 30 πακέτα του diffusion στέλνει ένα μήνυμα τύπου (DIFFUSION_OVER) σε όλους τους υπόλοιπους peers. Όταν όλοι οι peers λάβουν το μήνυμα (DIFFUSION_OVER) από όλους τους υπόλοιπους peers αποθηκεύεται η συγκεκριμένη χρονική στιγμή. Έτσι στο επόμενο check time μετά τη χρονική στιγμή που αποθηκεύτηκε αρχίζει η διαδικασία του swarming. Με αυτόν τον τρόπο υπάρχει μια καθυστέρηση από τη στιγμή που όλοι οι peers έχουν τα 30 πακέτα του diffusion μέχρι τη στιγμή που αρχίζει το swarming (λόγω της επικοινωνίας που αναφέραμε παραπάνω), αυτή η καθυστέρηση ονομάζεται total delay και θα μετρηθεί παρακάτω στην ανάλυση των αποτελεσμάτων.

4) App3(D-S)(4S)

Αυτή η παραλλαγή είναι ίδια με την παραπάνω εφαρμογή. Η διαφορά είναι και πάλι ότι υπάρχουν 4 peers (25,50,75,100) που έχουν τα ίδια πακέτα του βίντεο όπως και στην παραπάνω παραλλαγή (2) .

5)App3(TREE)(2S)

Έχουμε αναφέρει ότι σε όλες τις παραπάνω εφαρμογές την αποστολή όλων των πακέτων στη φάση diffusion αναλαμβάνουν οι peers που έχουν όλο το βίντεο αρχικά, προκαλώντας μεγάλο φόρτο στο δίκτυο. Αυτό ακριβώς γίνεται προσπάθεια να αλλαχθεί στην παρούσα εφαρμογή. Εδώ οι 2 peers (50,100) που έχουν ολόκληρο το βίντεο στέλνουν τα κατάλληλα πακέτα μόνο στον πρώτο peer του κάθε κλάδου (αποκλειστικά στους peers του $1^{ου}$ επιπέδου του). Για παράδειγμα στέλνουν τα πακέτα 1,11,21,31,41,…,291 μόνο στον peer1, τα πακέτα

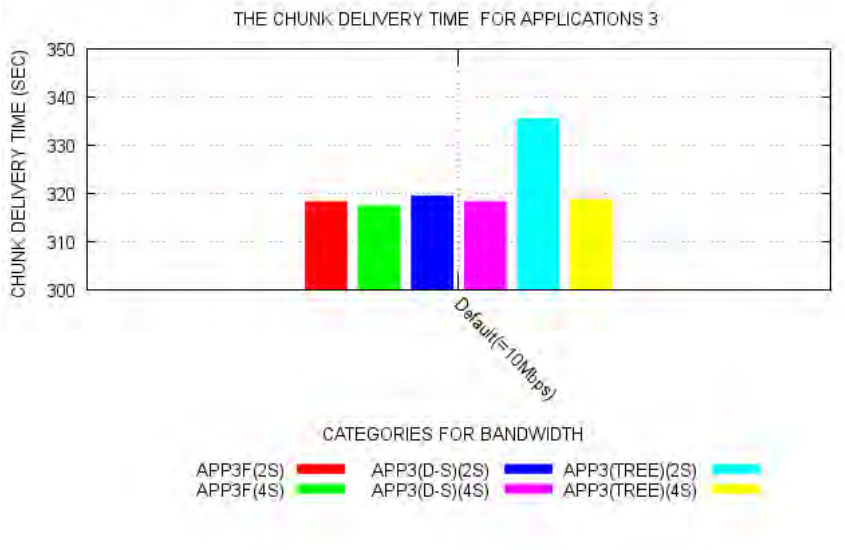
2,12,22,32,42,…,192 μόνο στον peer2 και συνεχίζεται έτσι αυτή η διαδικασία. Από τη στιγμή που οι peers του 1^{ου} επιπέδου του diffusion tree (1,2,3,4,5,6,7,8,9,10) λάβουν κάποιο από τα 30 πακέτα που πρέπει να λάβουν σε αυτή τη φάση το προωθούν στο επόμενο peer του κλάδου. Έτσι τα 30 πακέτα που πρέπει να λάβει ο κάθε peer φτάνουν με μια διαδικασία push κατά μήκος του κάθε κάθετου κλάδου του δέντρου. Γενικότερα ως διαδικασία push προσδιορίζεται η μετάδοση πακέτων από τα υψηλότερα επίπεδα στα χαμηλότερα του diffusion tree κατά τη διάρκεια της φάσης diffusion.

6)App3(TREE)(4S)

Κι εδώ έχουμε την ίδια εφαρμογή με την παραπάνω, μόνο πού έχουμε 4 peers(25,50,75,100) που έχουν τα ίδια πακέτα του βίντεο όπως οι peers στις εφαρμογές 2,4.

Ανάλυση αποτελεσμάτων της ΕΦΑΡΜΟΓΗΣ 3.

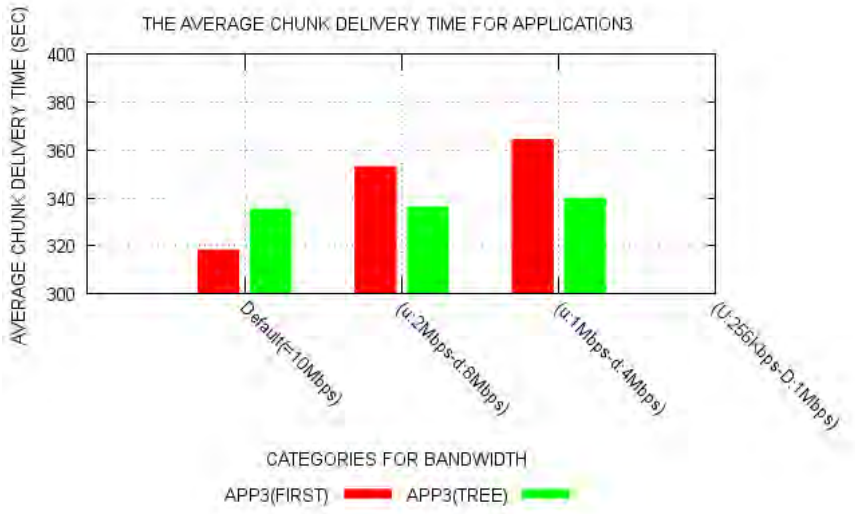
1) Chunk Delivery Time



Αρχικά η πρώτη παρατήρηση που μπορούμε να κάνουμε είναι ότι οι εφαρμογές στις οποίες υπάρχουν 4 peers που έχουν κάποια πακέτα του βίντεο έχουν καλύτερα αποτελέσματα σε σχέση με αυτές που έχουν 2 peers και έχουν όλα τα πακέτα του βίντεο. Ο λόγος που γίνεται αυτό είναι ότι με 4 peers ως αρχικούς αποστολείς έχουμε καλύτερη διαβάθμιση στο δίκτυο και εκμεταλλευόμαστε πιο αποτελεσματικά το bandwidth μεταξύ περισσότερων συνδέσεων στο δίκτυο. Όταν ο αριθμός των αρχικών αποστολέων είναι μεγαλύτερος το δίκτυό μας έχει περισσότερη ευελιξία(flexibility) και μειώνεται η πιθανότητα του content bottleneck. Το content bottleneck συμβαίνει όταν δε χρησιμοποιείται επαρκώς (utilization) το upload bandwidth των servers. Οι εφαρμογές APP3F(2S)(318.4 seconds) και APP3F(4S)(317.6seconds) έχουν τα καλύτερα αποτελέσματα σε σχέση

με τις υπόλοιπες, κυρίως γιατί αυτές οι εφαρμογές δεν επιβαρύνονται με την καθυστέρηση (total delay) για τον διαχωρισμό των φάσεων diffusion και swarming. Έπειτα η εφαρμογή APP3(D-S)(2S)(319.62seconds) έχει καλύτερα αποτελέσματα σε σχέση με την APP3(TREE)(2S)(335.62seconds). Αυτό γίνεται γιατί η APP3(TREE) χρειάζεται κάποιο παραπάνω χρόνο για να ολοκληρώσει τη διαδικασία του push στη φάση του diffusion, η οποία εξαρτάται από το βάθος των υποδέντρων του diffusion. Τι γίνεται όμως όταν έχουμε περισσότερους peers (4) που πρέπει να στείλουν σε όλους τους υπόλοιπους peers όλα τα πακέτα του diffusion; Φτάνει το διαθέσιμο bandwidth ή θα έχουμε προβλήματα; Από το γεγονός ότι η APP3(TREE)(4S)(318.82seconds) έχει περίπου ίσα αποτελέσματα από την APP3(D-S)(4S)(318.42seconds) φαίνεται ότι η APP3(D-S) προκαλεί αρκετό φόρτο στο δίκτυο, έτσι ώστε η APP3(TREE) να έχει περίπου ίσα αποτελέσματα παρόλο την καθυστέρηση λόγω του push στη φάση diffusion.

Κατηγορίες Bandwidth



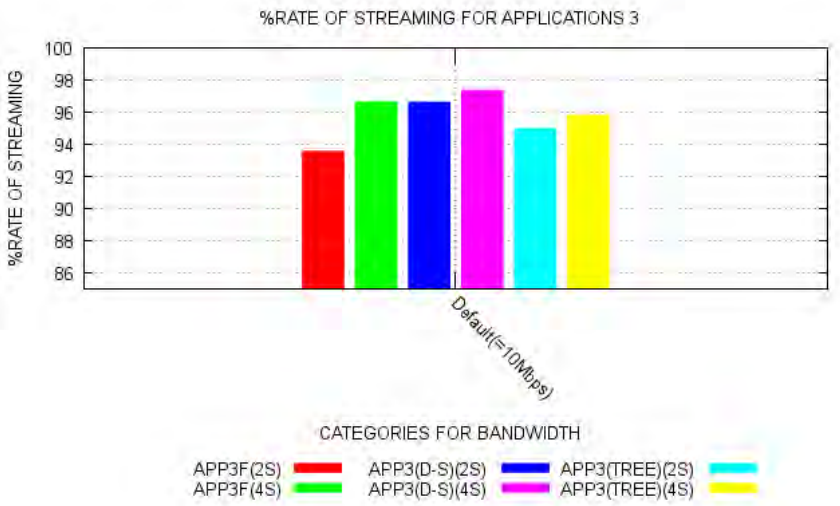
Στο παραπάνω γράφημα έχουμε τις παρακάτω αναλογίες για τις ονομασίες των εφαρμογών:

- APP3(FIRST)=APP3F(2S)
- APP3(TREE)=APP3(TREE)(2S)

Είναι ιδιαίτερα σημαντικό ότι παρατηρούμε αρκετές διαφοροποιήσεις όταν αλλάζουμε το bandwidth σε σχέση με το default bandwidth. Για παράδειγμα, στις κατηγορίες 1(download:8Mbps, upload:2Mbps) και 2 (download:4Mbps, upload:1Mbps) φαίνεται ότι η APP3(TREE) έχει καλύτερα αποτελέσματα από την APP3(FIRST),ενώ για τη default περίπτωση δεν ισχύει το ίδιο. Αυτό γίνεται γιατί στην APP3(TREE) εκμεταλλευόμαστε το bandwidth όλων των συνδέσεων μεταξύ των peers μέσω της διαδικασίας του push (μετάδοση των πακέτων από τα υψηλότερα επίπεδα στα χαμηλότερα του diffusion tree), ενώ στην APP3(FIRST) δεν υπάρχει πλέον ούτε αρκετό upload ούτε αρκετό download bandwidth για να στείλουν αποκλειστικά οι αρχικοί αποστολείς τα κατάλληλα πακέτα σε όλους τους peers στη φάση του diffusion. Έτσι δημιουργείται αρκετή συμφόρηση στο δίκτυο με συνέπεια οι peers της APP3(FIRST) να καθυστερούν περισσότερο να λάβουν όλα τα πακέτα του βίντεο. Επίσης μία επιπλέον καθυστέρηση προκαλείται στην εφαρμογή APP3(FIRST), εξαιτίας του μη διαχωρισμού των φάσεων diffusion και swarming. Όταν το swarming ξεκινάει πριν τελειώσει το diffusion,

οι swarming parents δεν έχουν και τα 30 πακέτα που θα αποκτήσουν μετά το τέλος της φάσης diffusion. Όταν όμως οι swarming parents δεν έχουν αρκετά πακέτα να μεταδώσουν στα παιδιά τους προκαλείται μια καθυστέρηση λόγω content bottleneck. Τέλος παρατηρούμε κι εδώ ότι ακόμα και με τις μειώσεις του bandwidth για την εφαρμογή APP3(TREE)(2S) το chunk delivery time παραμένει σχεδόν σταθερό.

2)% Average Rate Of Streaming



Και πάλι οι εφαρμογές με 4 αρχικούς αποστολείς του βίντεο (χωρίς αυτοί να έχουν ολόκληρο το βίντεο) έχουν καλύτερα αποτελέσματα από αυτούς με 2 αποστολείς (έχουν ολόκληρο το βίντεο) λόγω καλύτερης διαβάθμισης όπως αναφέραμε και παραπάνω. Από την άλλη η APP3(D-S)(2S)(96.64%) έχει καλύτερη επίδοση από την APP3(FIRST)(2S)(93.59%), εξαιτίας του γεγονότος ότι σε αυτή διαχωρίζονται οι φάσεις diffusion και swarming. Έτσι όταν αρχίζει το swarming, οι peers παίρνουν άμεσα όποιο πακέτο ζητήσουν , αφού σε όλους τους peers έχουν φτάσει ήδη τα 30 πακέτα του diffusion. Άρα το streaming μπορεί να συνεχιστεί πιο αποτελεσματικά, ενώ στην APP3(FIRST) μπορεί να υπάρχουν αστοχίες που καθυστερούν το streaming . Τέλος η APP3(TREE)(2S)(95.01%) έχει χειρότερα αποτελέσματα από την APP3(D-S)(2S), εξαιτίας της καθυστέρησης λόγω της διαδικασίας του push στο τελείωμα του diffusion και κατά συνέπεια στην έναρξη του streaming.

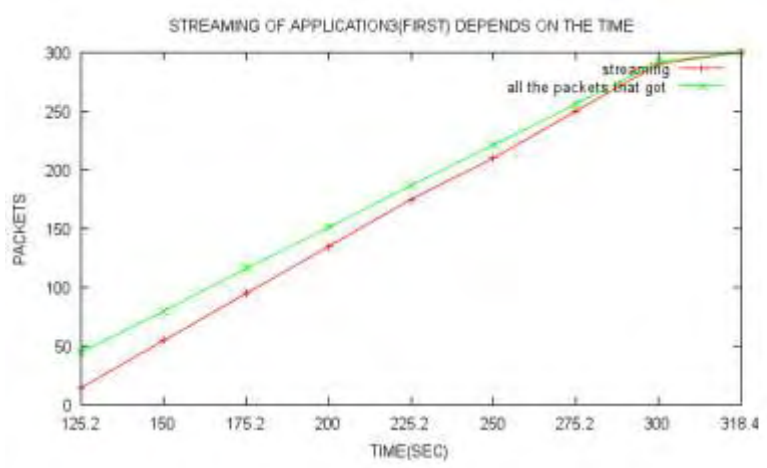
Γραφήματα του Streaming στην πορεία του χρόνου

Είδαμε παραπάνω το % Average rate of streaming για όλες τις εφαρμογές. Παρακάτω για κάθε εφαρμογή θα δούμε ένα γράφημα που θα μας δείχνει πόσα πακέτα έχει λάβει ο κόμβος 21 και πόσα έχουν παίξει σε δεδομένες χρονικές στιγμές. Με αυτόν τον τρόπο θα εξεταστεί η εξέλιξη του streaming στην πορεία του χρόνου.

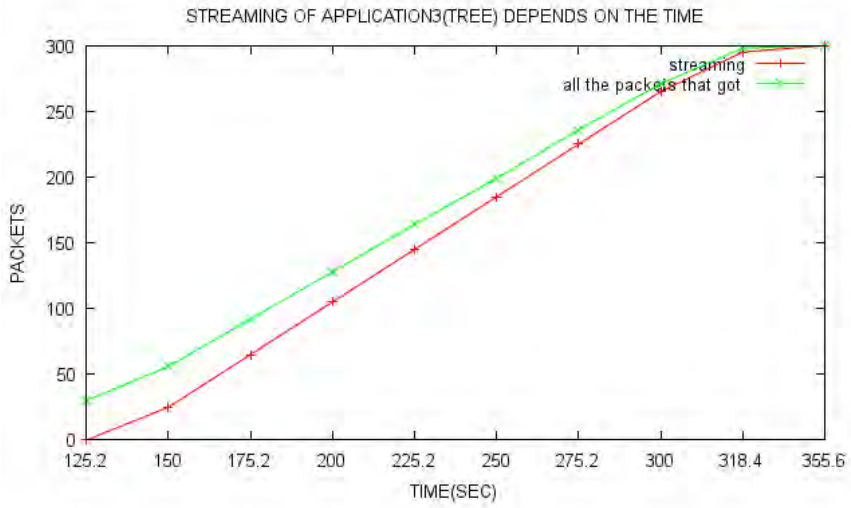
Συγκρίνοντας παρακάτω τις εφαρμογές APP3F(2S) και APP3(TREE)(2S), φαίνεται ότι τα παρακάτω γραφήματα μοιάζουν. Αρχικά, μέχρι την ολοκλήρωση της φάσης diffusion, δεν υπάρχει εξέλιξη στο streaming (η απόκλιση μεταξύ των 2 γραμμών στο γράφημα είναι αρκετά μεγάλη), ενώ όσο η φάση του swarming προχωράει και το

streaming εξελίσσεται με πιο γρήγορους ρυθμούς η οι 2 γραμμές του γραφήματος αρχίζουν να συγκλίνουν, μόνο που η APP3F(2S) αρχίζει να συγκλίνει πιο γρήγορα (στα 300 seconds περίπου) γιατί δεν επιβαρύνεται με την επικοινωνία που απαιτείται για τον διαχωρισμό των φάσεων diffusion-swarming στην APP3(TREE)(2S) (συγκλίνει στα 318,4 sec περίπου).

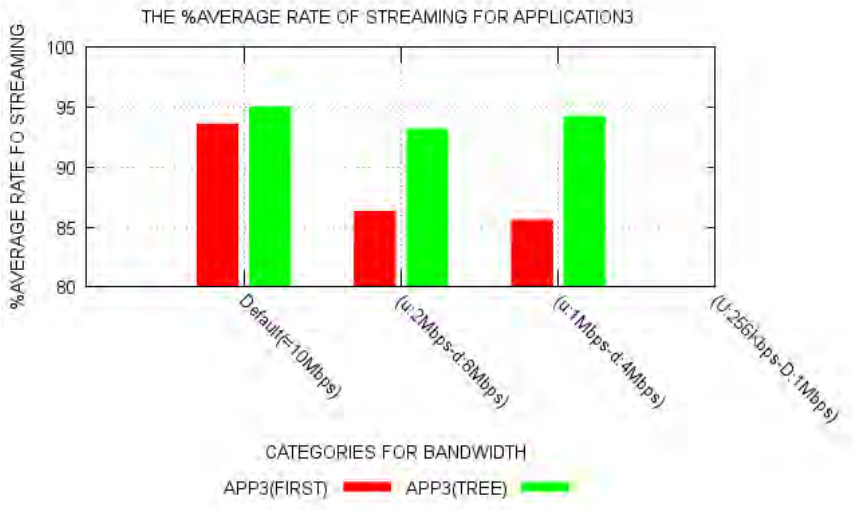
1)App3F(2S)



2)APP3(TREE)(2S)



Κατηγορίες Bandwidth



Στο παραπάνω γράφημα έχουμε τις παρακάτω αναλογίες για τις ονομασίες των εφαρμογών:

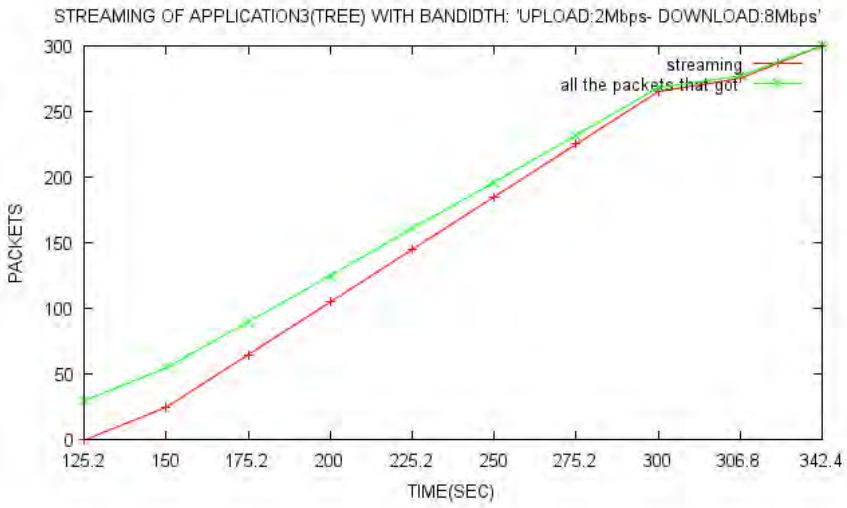
- APP3(FIRST)=APP3F(2S)
- APP3(TREE)=APP3(TREE)(2S)

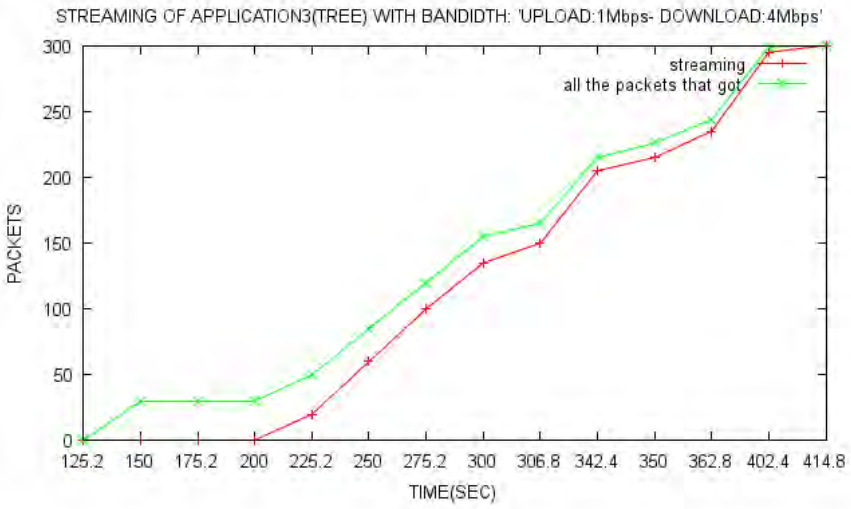
Εδώ μπορούμε να παρατηρήσουμε ότι για την APP3(FIRST) η επίδοση χειροτερεύει ενώ μειώνουμε το bandwidth, γιατί όπως είπαμε και παραπάνω το bandwidth πλέον δεν επαρκεί και δημιουργείται περισσότερη συμφόρηση στο δίκτυο. Από την άλλη η APP3(TREE) βελτιώνεται από την κατηγορία 1 bandwidth(93.18%) στην κατηγορία 2 bandwidth(94.25%). Επομένως μπορούμε να πούμε ότι λόγω της καλύτερης διαβάθμισης που έχει η APP3(TREE) στο diffusion η μείωση του bandwidth δεν την επηρεάζει και τόσο. Κι εδώ στην APP3(TREE)(2S) η ποιότητα του streaming παραμένει σταθερή σε όλες τις κατηγορίες bandwidth λόγω της καλύτερης scalability που έχει και της καλύτερης χρήσης του upload bandwidth που γίνεται σε όλους τους κόμβους .

Γραφήματα του Streaming στην πορεία του χρόνου

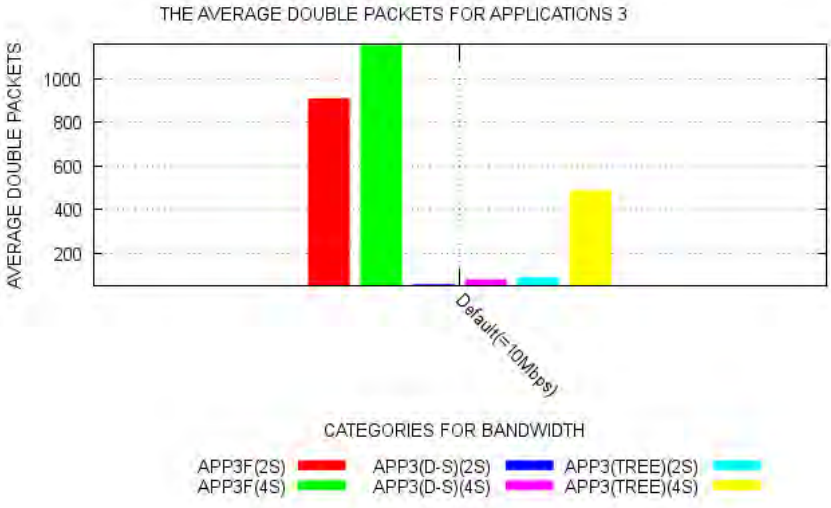
APP3(TREE)(2S)

Βλέπουμε παρακάτω, συγκρίνοντας την εφαρμογή APP3(TREE)(2S) στην $1^{η}$ κατηγορία (upload:2Mbps-download:8Mbps) και στη $2^{η}$ κατηγορία bandwidth (upload:1Mbps- download:4Mbps), την αστάθεια που υπάρχει στο σύστημα (περισσότερες καμπυλώσεις) λόγω της μείωσης του bandwidth.



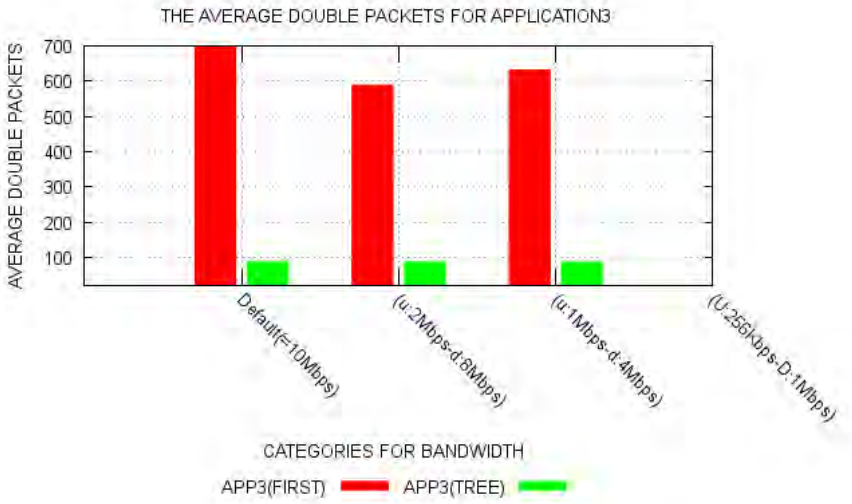


3)Average Number Of Double Packets



Αρχικά οι εφαρμογές με τους 4 αρχικούς αποστολείς έχουν περισσότερα διπλά πακέτα από τις εφαρμογές με 2 αρχικούς αποστολείς. Αυτό γίνεται γιατί όταν υπάρχουν 4 αρχικοί αποστολείς, υπάρχουν περισσότεροι αποστολείς που στέλνουν πακέτα στη φάση diffusion σε όλους τους peers, με συνέπεια να φτάνουν και περισσότερα διπλά πακέτα. Τα περισσότερα διπλά πακέτα τα έχουν οι εφαρμογές APP3(FIRST), συγκεκριμένα η APP3(FIRST)(2S)(910 διπλά πακέτα) έχει περισσότερα διπλά πακέτα επειδή οι φάσεις diffusion και swarming δεν διαχωρίζονται με αποτέλεσμα εφόσον δεν γνωρίζουμε πότε τελειώνει το diffusion, αυτό να συνεχίζεται και εφόσον έχουν φτάσει σε όλους τους peers τα 30 πακέτα (της συγκεκριμένης φάσης). Αυτό δείχνει μια αρνητική πτυχή που έχει η συγκεκριμένη υλοποίηση με συνέπεια να τροφοδοτούνται με πολλά διπλά πακέτα οι peers. Η APP3(TREE)(2S)(90 διπλά πακέτα) έχει περισσότερα διπλά πακέτα από την APP3(D-S)(2S)(59 διπλά πακέτα), επειδή η φάση diffusion διαρκεί περισσότερο, έχοντας ως επακόλουθο οι αρχικοί αποστολείς να στέλνουν περισσότερα διπλά πακέτα στους πρώτους peers των κλάδων του δέντρου.

Κατηγορίες Bandwidth



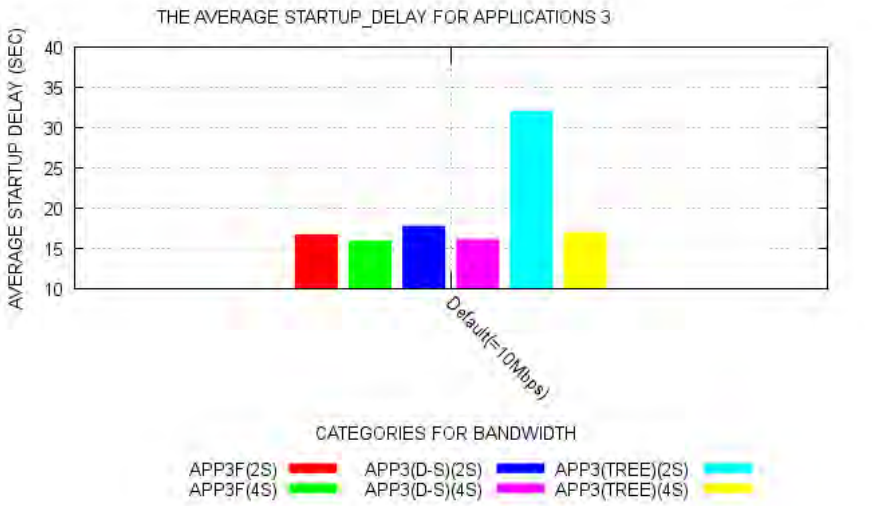
Στο παραπάνω γράφημα έχουμε τις παρακάτω αναλογίες για τις ονομασίες των εφαρμογών:

- APP3(FIRST)=APP3F(2S)
- APP3(TREE)=APP3(TREE)(2S)

Αναφέραμε και παραπάνω το λόγο για τον οποίο η APP3(FIRST) έχει περισσότερα διπλά πακέτα από την APP3(TREE). Ωστόσο η APP3(FIRST) αυξάνει τα διπλά πακέτα της όσο μειώνουμε το bandwidth από την κατηγορία 1 bandwidth(589 διπλά πακέτα) στην κατηγορία 2 (632 διπλά πακέτα). Αυτό σημαίνει ότι με τη μείωση του bandwidth αυξάνεται το chunk delivery time, με συνέπεια οι αρχικοί αποστολείς να δυσκολεύονται να διοχετεύσουν με τα πακέτα του diffusion τους peers που βρίσκονται σε μακρινή απόσταση (λόγω έλλειψης

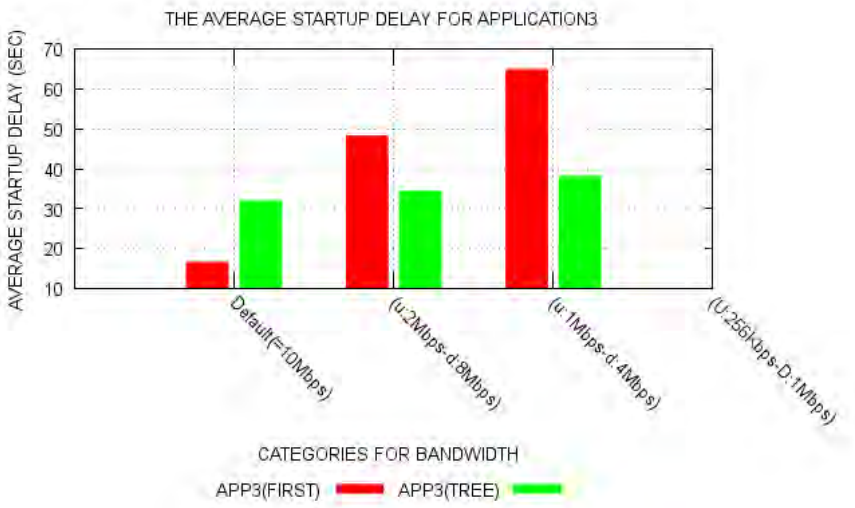
bandwidth). Άρα απαιτείται περισσότερος χρόνος για να φτάσουν τα πακέτα από τους αρχικούς αποστολείς στους μακρινούς peers στη διάρκεια του diffusion με αποτέλεσμα σε όλο αυτό το χρονικό διάστημα να διοχετεύονται με διπλά πακέτα οι peers που βρίσκονται σε κοντινές αποστάσεις από τους αρχικούς αποστολείς. Από την άλλη για την εφαρμογή APP3(TREE) που λόγω του push δεν στέλνονται πακέτα ανάμεσα σε μακρινούς peers στο diffusion, δεν υπάρχουν μεγάλες διαφορές στα διπλά πακέτα ανάμεσα στην κατηγορία 1 bandwidth(90 διπλά πακέτα) και στην κατηγορία 2 bandwidth(89 διπλά πακέτα). Κι εδώ στην APP3(TREE)(2S) τα διπλά πακέτα παραμένουν σταθερά σε όλες τις κατηγορίες bandwidth λόγω της καλύτερης scalability που έχει.

4)Average Startup Delay



Και πάλι οι εφαρμογές με 4 αρχικούς αποστολείς έχουν μικρότερα startup delay από αυτές με τις 2 εφαρμογές. Το μικρότερο startup delay το έχουν οι εφαρμογές APP3(FIRST). Συγκεκριμένα, η APP3(FIRST)(2S) έχει startup delay (16.8 seconds), γιατί δεν έχουμε την καθυστέρηση για την επικοινωνία του διαχωρισμού των φάσεων που έχουμε στις άλλες δύο εφαρμογές. Βέβαια, δεν ξέρουμε τι θα γινόταν αν μειώναμε το bandwidth πιο κάτω από τη default τιμή. Στη συνέχεια η APP3(D-S)(2S)(17.82 seconds) έχει μικρότερο startup delay από την APP3(TREE)(2S)(32.05 seconds), εξαιτίας του push που καθυστερεί τη διαδικασία του diffusion στην APP3(TREE)(2S)(σε Default bandwidth). Προφανώς όσο καθυστερεί το diffusion τόσο αυξάνεται και το startup delay, αφού τα 30 πακέτα που λαμβάνουν οι peers δεν είναι στη σωστή σειρά για να παιχτούν.

Κατηγορίες Bandwidth



Στο παραπάνω γράφημα έχουμε τις παρακάτω αναλογίες για τις ονομασίες των εφαρμογών:

- APP3(FIRST)=APP3F(2S)
- APP3(TREE)=APP3(TREE)(2S)

Εκείνο που παρατηρούμε αρχικά είναι ότι όσο αυξάνουμε το bandwidth τόσο αυξάνεται το startup delay και για τις 2 εφαρμογές. Εκείνο που έχει σημασία όμως είναι, όταν έχουμε αρκετό bandwidth (default περίπτωση) η APP3(FIRST) έχει μικρότερο startup delay από την APP3(TREE), γιατί φαίνονται οι καθυστερήσεις που υπάρχουν στο diffusion στην APP3(TREE) (λόγω επικοινωνίας για διαχωρισμό των φάσεων και λόγω του push). Όταν πάμε στην 1η κατηγορία bandwidth, δεν είναι εμφανείς πλέον οι παραπάνω καθυστερήσεις στην APP3(TREE)και έχει μικρότερη startup delay(34.62 seconds) από την APP3(FIRST)(48.3 seconds). Όπως έχουμε αναφέρει και παραπάνω, μειώνοντας το bandwidth στην APP3(FIRST) οι αρχικοί αποστολείς δυσκολεύονται να επικοινωνήσουν με τους μακρινούς peers αυξάνοντας το startup delay. Έτσι, στους μακρινούς peers δεν φτάνουν γρήγορα αρκετά πακέτα, με συνέπεια να μην έχει αρκετά πακέτα ο συγκεκριμένος peer για να μεταδώσει στα παιδιά του. Γι' αυτόν τον λόγο δημιουργείται μια καθυστέρηση λόγω content bottleneck. Μειώνοντας ακόμα παραπάνω το bandwidth και πηγαίνοντας στη $2^{η}$ κατηγορία bandwidth η διαφορά στο startup delay ανάμεσα στις 2 εφαρμογές μεγαλώνει. Η APP3(TREE) έχει startup delay (38.22 seconds), ενώ η APP3(FIRST) έχει (64.89 seconds). Κι εδώ στην APP3(TREE)(2S) το startup delay παραμένει σταθερό σε όλες τις

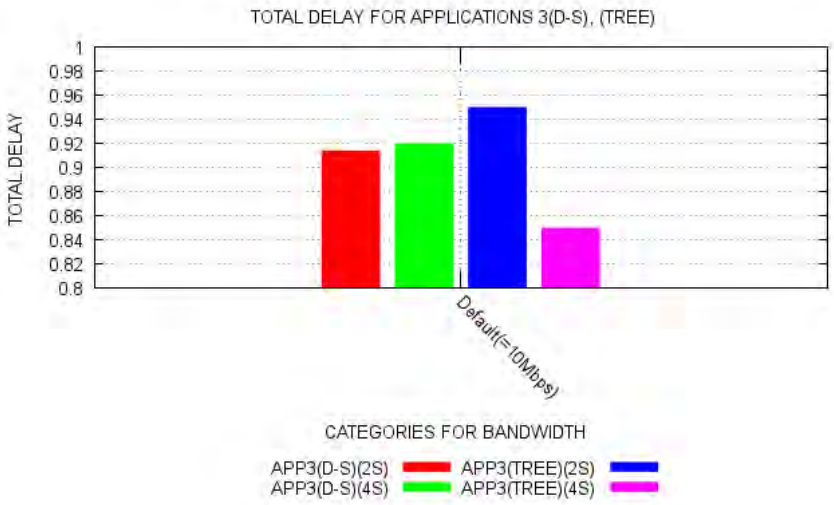
κατηγορίες bandwidth λόγω της καλύτερης scalability που έχει και της καλύτερης χρήσης του upload bandwidth που γίνεται σε όλους τους κόμβους.

5)Total Delay

Εδώ προσθέτουμε μία καινούργια παράμετρο (total delay), η οποία αφορά μόνο τις εφαρμογές στις οποίες διαχωρίζονται οι φάσεις diffusion-swarming (APP3(D-S), APP3(TREE)). Η παράμετρος total delay είναι το χρονικό διάστημα από τη στιγμή πού ο τελευταίος peer αποκτά τα 30 πακέτα του diffusion μέχρι τη χρονική στιγμή που αρχίζει το swarming για τον πρώτο peer. Αυτό το χρονικό διάστημα περιλαμβάνει το άθροισμα:

1) της καθυστέρησης από τη στιγμή που ο τελευταίος peer τελειώνει το diffusion μέχρι τη στιγμή που και ο τελευταίος peer λαμβάνει όλα τα μηνύματα τύπου (DIFFUSION_OVER) από όλους τους υπόλοιπους peers, χρονική στιγμή την οποία αποθηκεύουμε ως last period_swarm .

2) της καθυστέρησης από τη χρονική στιγμή last period_swarm μέχρι τη χρονική στιγμή που κάποιος peer λαμβάνει το επόμενο swarming_timerMsg event (μετά τη χρονική στιγμή last period_swarm) και αρχίζει για πρώτη φορά να υλοποιεί το swarming. Στα προηγούμενα swarming_timerMsg events ο παραπάνω peer δεν έκανε απολύτως καμία ενέργεια, αφού δεν είχε φτάσει σε αυτόν το σήμα της ολοκλήρωσης της φάσης diffusion.

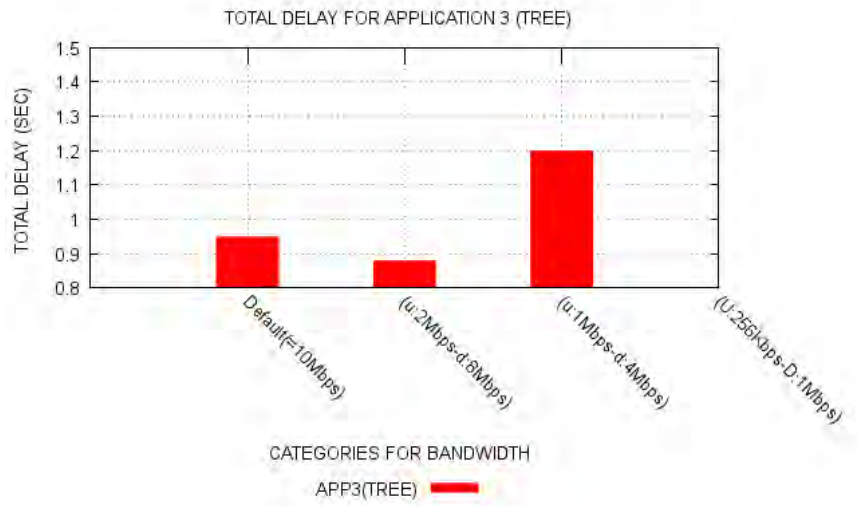


Όσον αφορά την εφαρμογή APP3(D-S) βλέπουμε ότι το total delay παραμένει σταθερό όταν έχουμε 2 αρχικούς αποστολείς (0.914 seconds) σε σχέση με 4 αποστολείς (0.92 seconds). Από την άλλη συγκρίνοντας το total delay της APP3(D-S)(2S)(0.914 seconds) με την APP3(TREE)(2S)(0.95 seconds), βλέπουμε ότι το total delay της APP3(TREE)(2S) είναι αμελητέα μεγαλύτερο γιατί το diffusion καθυστερεί να τελειώσει(λόγω του push), (γύρω

στα 131.07 seconds) , ενώ για την APP3(D-S)(2S) το diffusion τελειώνει γύρω στα 115.06 seconds. Ο κυριότερος λόγος για την καθυστέρηση του diffusion (131.07 seconds) είναι ότι κάποιος πατέρας μπορεί να μην έχει αρκετά πακέτα σε κάποια χρονική στιγμή για να τα μεταδώσει στο παιδί του προκαλώντας μια καθυστέρηση λόγω content bottleneck.

Αν τώρα συγκριθεί το total delay της APP3(TREE)(4S)(0.85 seconds) με την APP3(D-S)(4S)(0.92 seconds), βλέπουμε ότι τώρα τα πράγματα αλλάζουν και η APP3(TREE) έχει μικρότερη total delay από την APP3(D-S). Η APP3(TREE) τελειώνει το diffusion στα 114.37 seconds, ενώ η APP3(D-S) τελειώνει το diffusion στα 113.9 seconds. Όμως παρόλο που η APP3(TREE) καθυστερεί ελάχιστα να τελειώσει το diffusion σε σχέση με την APP3(D-S), στη συνέχεια γίνεται γρηγορότερα η επικοινωνία μεταξύ των peers με συνέπεια να έχει ελάχιστα μικρότερο total delay από την APP3(D-S). Όπως είπαμε παραπάνω στην εφαρμογή APP3(TREE)(2S) το diffusion τελειώνει στα 131.07 seconds ενώ στην APP3(TREE)(4S) στα 114.37 seconds. Αυτό συμβαίνει γιατί αποκλειστικά με 2 αποστολείς στην APP3(TREE)(2S) πρέπει να φτάσουν τα 30 πακέτα του diffusion στους peers του 1^{ου} επιπέδου, προκαλώντας συμφόρηση στις συνδέσεις αυτών των 2 αρχικών αποστολέων με αποτέλεσμα να καθυστερούν να φτάσουν τα πακέτα στους peers του 1^{ου} επιπέδου και έτσι να καθυστερεί να τελειώσει και το diffusion. Όταν τώρα έχουμε 4 αρχικούς αποστολείς υπάρχει καλύτερη διαβάθμιση με συνέπεια να μειώνεται και η συμφόρηση στις συνδέσεις αυτών των αρχικών αποστολέων με τους peers του 1^{ου} επιπέδου. Γι' αυτόν τον λόγο τελειώνει και πιο γρήγορα το diffusion.

Κατηγορίες Bandwidth



Συγκρίνοντας το total delay της εφαρμογής APP3(TREE), από την κατηγορία 1 bandwidth(0.88 seconds) στην κατηγορία 2 (1.2 seconds), βλέπουμε ότι με τη μείωση του bandwidth έχουμε μία καθυστέρηση στην ολοκλήρωση της φάσης diffusion, από τα 131.94 seconds που ήταν στην κατηγορία 1, φτάνει στα 135.22 seconds στην κατηγορία 2 του bandwidth.

4.4) Συμπεράσματα

Γενικά μπορούμε να πούμε ότι οι εφαρμογές με τους 4 αρχικούς αποστολείς έχουν καλύτερες επιδόσεις για όλες τις παραμέτρους, λόγω καλύτερης διαβάθμισης, παρόλο που δεν έχουν όλα τα πακέτα του βίντεο αρχικά, σε σχέση με τις εφαρμογές που έχουν 2 αρχικούς αποστολείς που έχουν όλα τα πακέτα του βίντεο αρχικά. Επίσης παρατηρήθηκε ότι μεγαλύτερη σημασία έχει το upload bandwidth μεταξύ των αποστολέων και των peers του $1^{ου}$ επιπέδου του diffusion tree. Εκεί χρειάζεται οι αρχικοί αποστολείς να μεταδώσουν τα κατάλληλα πακέτα σε όλους τους peers του $1^{ου}$ επιπέδου και γι' αυτόν τον λόγο προκαλείται μια καθυστέρηση λόγω της συμφόρησης που προκαλείται. Μια καλή τακτική θα ήταν να περιορίσουμε τον αριθμό των peers στους οποίους μεταδίδουν πακέτα οι αρχικοί αποστολείς. Για παράδειγμα, θα μπορούσε ο αριθμός των peers στους οποίους μεταδίδουν πακέτα οι αρχικοί αποστολείς να ταυτίζεται με το upload bandwidth του κάθε αρχικού αποστολέα. Έτσι θα είχαμε καλύτερο utilization του upload bandwidth των αρχικών αποστολέων και θα μειώναμε τη συμφόρηση σε αυτό το σημείο. Αυτός είναι και ο βασικός λόγος που έχουμε καλύτερα αποτελέσματα όταν χρησιμοποιούμε 4 αρχικούς αποστολείς και έτσι ο καθένας από αυτούς πρέπει να μεταδώσει λιγότερα πακέτα στους peers του $1^{ου}$ επιπέδου, σε σχέση με την περίπτωση των 2 αρχικών αποστολέων. Η μόνη παράμετρος που υστερούν οι εφαρμογές με τους 4 αρχικούς αποστολείς είναι τα διπλά πακέτα.

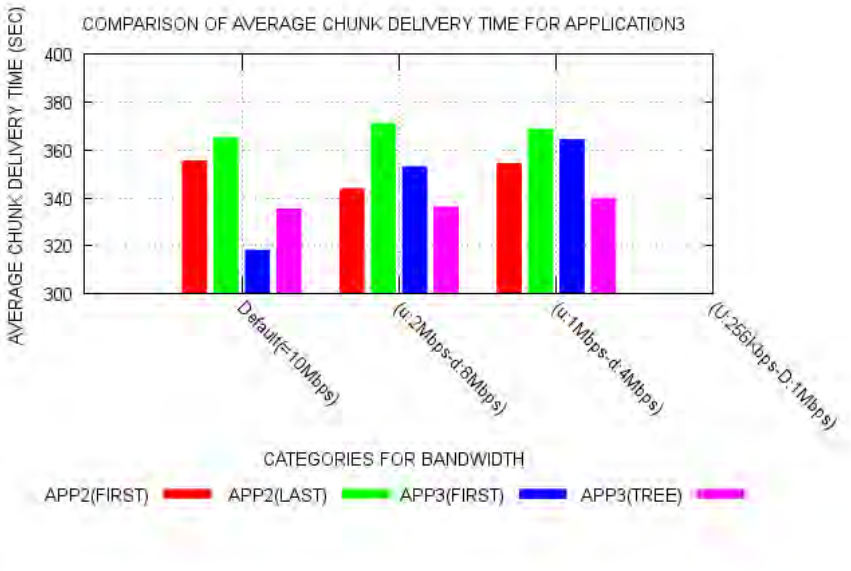
Μία άλλη παρατήρηση που μπορούμε να κάνουμε είναι ότι συχνά όταν μειώνεται το bandwidth, η εφαρμογή APP3(TREE) ξεπερνάει σε διάφορες παραμέτρους τις άλλες εφαρμογές, ενώ στη default περίπτωση του bandwidth δεν συνέβαινε το ίδιο. Αυτό εξηγείται από το γεγονός ότι λόγω του push που κάνει δεν χρειάζεται να υπάρχει επικοινωνία μεταξύ peers που βρίσκονται σε μακρινές αποστάσεις, έτσι ώστε να μην φτάνει το διαθέσιμο bandwidth, καθώς η μετάδοση των πακέτων στη φάση diffusion γίνεται ανάμεσα σε γειτονικά επίπεδα του diffusion tree.

Επιπλέον, ένα ιδιαίτερα σημαντικό στοιχείο που πρέπει να αναφέρουμε είναι ότι όταν συγκρίνονται οι 2 εφαρμογές APP3F(2S) και APP3(TREE)(2S) στις διαφορετικές κατηγορίες bandwidth είναι ότι παρά τις μειώσεις του bandwidth οι επιδόσεις της τελευταίας εφαρμογής σε όλες τις παραμέτρους είναι σχεδόν σταθερές. Από αυτό γίνεται κατανοητό ότι η καλύτερη scalability της APP3(TREE)(2S) σε σχέση με την APP3F(2S) προσφέρει πιο σταθερές επιδόσεις σε προσομοιώσεις σε συνθήκες πραγματικού συστήματος και οδηγεί σε καλύτερη χρήση του upload bandwidth των κόμβων του overlay.

Τέλος, σχετικά με την τελευταία εφαρμογή που υλοποιήθηκε την APP3(TREE), η καθυστέρηση της φάσης του diffusion εξαρτάται από το βάθος (depth) των κλάδων του δέντρου του diffusion. Πράγμα που σημαίνει ότι κάποια στιγμή κάποιος πατέρας ίσως έχει πολύ λίγα πακέτα να μεταδώσει στο παιδί του προκαλώντας μια σημαντική καθυστέρηση λόγω content bottleneck. Από την άλλη, ο λόγος που μπορεί να καθυστερήσει η φάση swarming είναι οι μεγάλες αποστάσεις των peers που ζητούν κάποια πακέτα από αυτούς τους peers από τους οποίους τα ζητούν (δηλαδή τους swarming parents).

Γενικότερα το θετικό της εφαρμογής APP3 στο diffusion είναι η οργάνωση των πακέτων σε διαφορετικά γκρουπ και η μετάδοση διαφορετικού γκρουπ σε διαφορετικά υποδέντρα. Ένα αρνητικό αυτής της εφαρμογής, το οποίο θα μπορούσε να βελτιστοποιηθεί, είναι ότι η μετάδοση των πακέτων στα διάφορα υποδέντρα στη φάση του diffusion γίνεται παράλληλα δημιουργώντας μεγαλύτερη συμφόρηση στο δίκτυο. Αν σειριοποιούσαμε τη μετάδοση στη φάση diffusion ώστε η μετάδοση των πακέτων σε κάθε υποδέντρο να γίνεται σε διαφορετική χρονική στιγμή τότε θα είχαμε προφανώς καλύτερα αποτελέσματα. Τέλος, στην περίπτωση που οι αρχικοί αποστολείς λάμβαναν τα πακέτα του βίντεο σταδιακά και δεν τα είχαν όλα τα πακέτα εξαρχής, η καθυστέρηση που προκαλεί το push στο diffusion θα μπορούσε να είναι ευεργετική, γιατί μέσα στο χρονικό διάστημα το οποίο ένα πακέτο προωθείται κατά μήκος ενός υποδέντρου, ο αρχικός αποστολέας αποκτάει κάποια από τα επόμενα πακέτα και είναι έτοιμος να τα μεταδώσει μέσω του diffusion.

5) Σύγκριση ΕΦΑΡΜΟΓΗΣ 2 – ΕΦΑΡΜΟΓΗΣ 3



Για τις παραπάνω παραλλαγές ισχύουν οι παρακάτω αντιστοιχίες:

- APP2(FIRST)=APP2(F-2S)
- APP2(LAST)=APP2(N-2S)
- APP3(FIRST)=APP3F(2S)
- APP3(TREE)=APP3(TREE)(2S)

Συγκρίνοντας το chunk delivery time μεταξύ κάποιων παραλλαγών της Εφαρμογής 2 και της Εφαρμογής 3 παρατηρούμε τα παρακάτω.

Πρώτα από όλα βλέπουμε ότι η APP3(TREE) έχει το μικρότερο chunk delivery time σε σχέση με οποιαδήποτε παραλλαγή της Εφαρμογής 2, πράγμα που δείχνει ότι λόγω της πιο ομοιόμορφης και πιο ισορροπημένης χρησιμοποίησης του bandwidth μεταξύ όλων των peers που γίνεται στην εφαρμογή APP3(TREE) , το βίντεο φτάνει πιο γρήγορα σε όλους τους peers.

6) Ανάλυση των αποτελεσμάτων για τον Pull- Based Αλγόριθμο για P2P-TV Streaming Systems

Ο αλγόριθμος Prime που είδαμε παραπάνω ανήκει στην κατηγορία των push αλγορίθμων που οργανώνουν τους peers σε δέντρα διανομής, στα οποία προωθούνται τα πακέτα του βίντεο. Γενικότερα στους push αλγορίθμους οι peers οργανώνονται σε δέντρα διανομής, τα οποία είναι συνήθως στατικά και μέσω αυτών μεταφέρονται συνεχόμενα chunks του βίντεο. Εκτός από την παραπάνω κατηγορία αλγορίθμων υπάρχει και η

κατηγορία των pull αλγορίθμων, στους οποίους προηγείται μια φάση trading. Σε αυτή τη φάση ο κάθε peer ανακοινώνει σε κάποιους γείτονες ποια πακέτα έχει, από τα οποία οι γείτονες επιλέγουν κάποια και του τα ζητούν. Οι pull αλγόριθμοι μειονεκτούν σε σχέση με τους push αλγορίθμους στο chunk delivery time, ενώ οι push αλγόριθμοι μειονεκτούν λόγω της πολυπλοκότητας τους για την κατασκευή των δέντρων και της μικρότερης ανθεκτικότητας στην παρουσία churn. Ο κάθε peer έχει μια transmission queue, στην οποία φτάνουν οι θετικές ανταποκρίσεις των γειτόνων για κάποιο πακέτο.

Τρείς είναι οι καθοριστικοί παράμετροι σε αυτούς τους αλγορίθμους:

1) Ο αριθμός των γειτόνων στους οποίους ο κάθε peer ανακοινώνει τα πακέτα που έχει.

2) Η συχνότητα με την οποία οι peers ανακοινώνουν τα πακέτα που έχουν στους γείτονες.

3) Ο αριθμός των πακέτων που ζητούν οι γείτονες από αυτόν που τους τα έχει ανακοινώσει.

Υλοποίηση

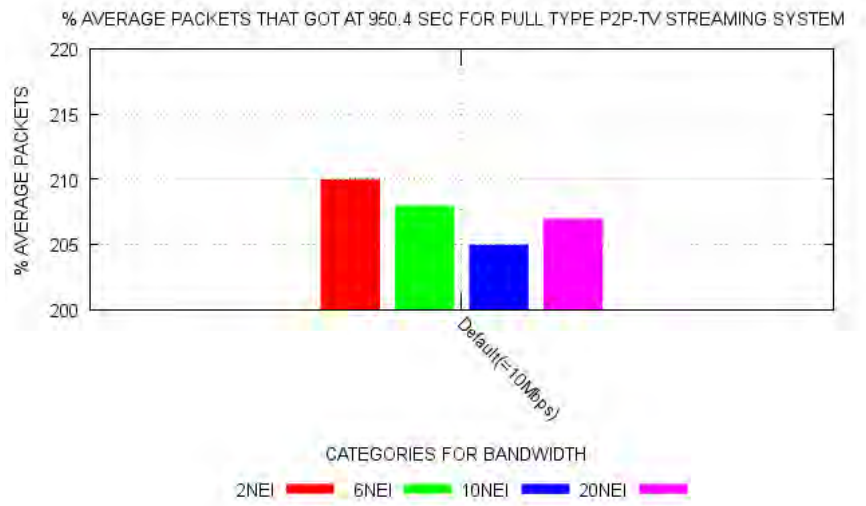
Στη δική μας υλοποίηση η παράμετρος 2, η συχνότητα ανακοίνωσης στους γείτονες, είναι 0,3 seconds.

Η παράμετρος 3, ο αριθμός των πακέτων που ζητούνται από τους γείτονες, είναι 1, το πιο σημαντικό για την εξέλιξη του streaming στον έλεγχο που κάνουν όλοι οι peers κατά τη διάρκεια του check time.

Στις παρακάτω μετρήσεις θα αποδειχθεί πώς επηρεάζει τα αποτελέσματα η αυξομείωση της παραμέτρου 1, δηλαδή ο αριθμός των γειτόνων στους οποίους ο κάθε peer ανακοινώνει τα πακέτα που έχει. Συγκεκριμένα έχουμε 4 παραλλαγές , σε κάθε μία από τις οποίες η παράμετρος 1 είναι:

1. 2 γείτονες
2. 6 γείτονες
3. 10 γείτονες
4. 20 γείτονες

1) Συνολικά πακέτα που έχουν φτάσει σε μια δεδομένη χρονική στιγμή.

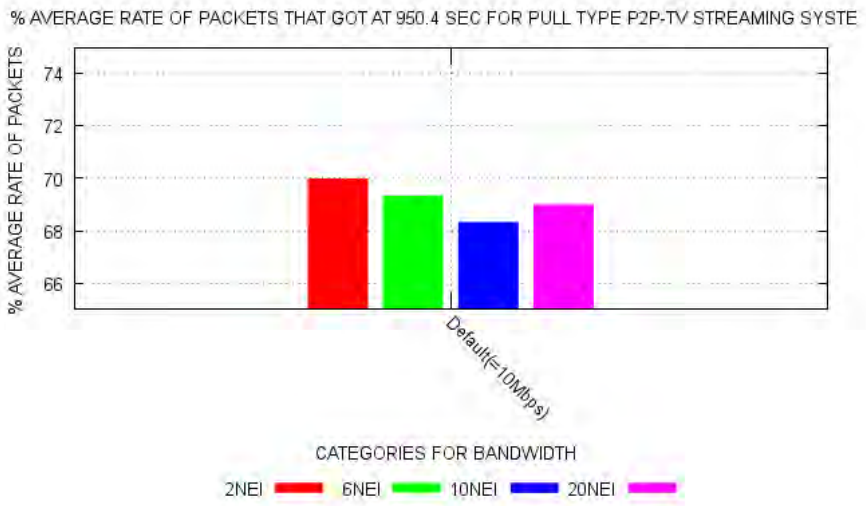


Εκείνο που μπορούμε να πούμε είναι ότι η σημαντικότερη διαφορά που παρατηρήθηκε συγκρίνοντας το simulation των pull αλγορίθμων (P2P-TV Streaming Systems) και το simulation των push αλγορίθμων (αλγόριθμος Prime) είναι ότι το chunk delivery time των pull αλγορίθμων είναι πολύ μεγαλύτερο σε σχέση με αυτό των push αλγορίθμων.

Στην παραπάνω εικόνα παίρνουμε μια τυχαία χρονική στιγμή του simulation (950.4 second) και βλέπουμε πόσα πακέτα έχουν φτάσει (βγάζουμε τον μέσο όρο από τα πακέτα που φτάνουν σε όλους τους peers) σε κάθε παραλλαγή. Παρατηρούμε , λοιπόν, ότι η εφαρμογή που χρησιμοποιεί μόνο 2 γείτονες έχει καλύτερα αποτελέσματα από τις υπόλοιπες και στη συνέχεια η εφαρμογή με τους 6 γείτονες έχει καλύτερα αποτελέσματα από αυτές με 10 και 20 γείτονες. Αυτό εξηγείται από το γεγονός ότι όταν υπάρχουν περισσότεροι γείτονες γεμίζουν οι transmission queues των peers που κάνουν τις ανακοινώσεις στους γείτονές τους, με συνέπεια να

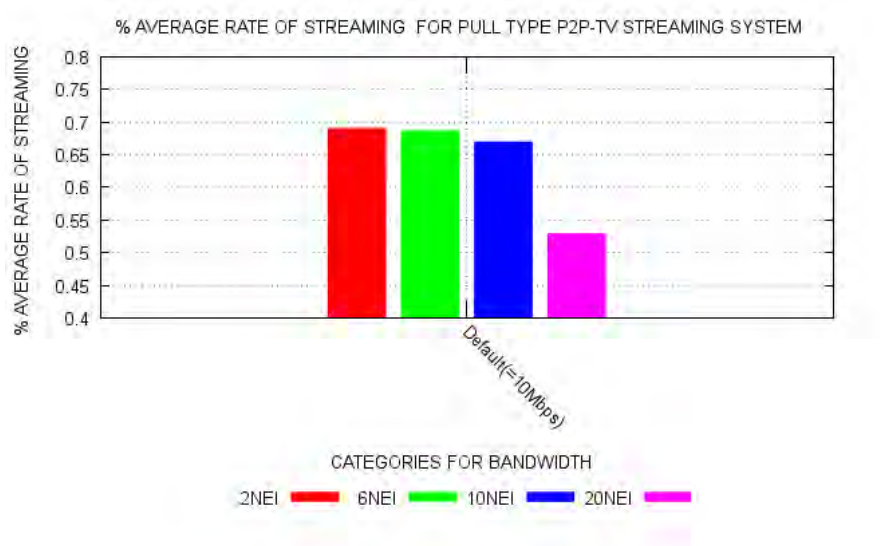
δημιουργείται μια συμφόρηση στο δίκτυο, εξαιτίας της οποίας φτάνουν στους peers λιγότερα πακέτα του βίντεο. Όταν όμως συγκρίνεται η παραλλαγή με τους 10 γείτονες με αυτήν με τους 20 γείτονες, παρατηρούμε ότι αυτή με τους 20 γείτονες έχει καλύτερα αποτελέσματα. Αυτό γίνεται προφανώς γιατί η διαφορά ανάμεσα σε 10 και 20 γείτονες είναι αρκετά μεγάλη και η ανακοίνωση των διαθέσιμων πακέτων των peers σε 20 γείτονες προκαλεί γενικότερα μεγαλύτερη επικοινωνία μεταξύ των peers του overlay (ο κάθε peer έχει τη δυνατότητα να επιλέξει κάποιο πακέτο από τις ανακοινώσεις πολύ περισσότερων γειτόνων (20)) με αποτέλεσμα να φτάνουν πιο πολλά πακέτα τη δεδομένη χρονική στιγμή σε κάθε peer. Γενικότερα, σε αυτού του είδους τις εφαρμογές όταν η παράμετρος 1 έχει μικρό αριθμό γειτόνων, οι peers που ανακοινώνουν τα πακέτα τους στους γείτονες δεν εκμεταλλεύονται πλήρως το upload bandwidth τους, με συνέπεια οι transmission queues να αδειάζουν γρήγορα προκαλώντας μεγάλα διαστήματα μη δραστηριότητας στο δίκτυο (idle times). Από την άλλη, όταν η παράμετρος 1 έχει πολλούς γείτονες οι transmission queues γεμίζουν γρήγορα, προκαλώντας μία επιπρόσθετη καθυστέρηση στην αποστολή των πακέτων του βίντεο στους peers (chunk delivery delay). Έτσι πρέπει να βρεθεί η χρυσή τομή για τη παράμετρο 1 και πρέπει να προσαρμοστεί ανάλογα με την upload capacity του κάθε peer, το μέσο RTT και το demand rate του συστήματος. Επιπρόσθετα, υπάρχει μία αναλογία μεταξύ της παραμέτρου 1, δηλαδή του αριθμού των γειτόνων με το upload bandwidth που έχουν οι peers. Όσο περισσότερο upload bandwidth έχουν οι peers τόσο το καλύτερο είναι να αυξηθεί ο αριθμός των γειτόνων με τους οποίους επικοινωνούν, έτσι ώστε να εκμεταλλεύονται όλο το uplink bandwidth τους και να παρουσιάζουν καλύτερα αποτελέσματα σε όλες τις παραμέτρους. Στη δική μας υλοποίηση χρησιμοποιούμε (download bandwidth= upload bandwidth= default bandwidth= 10Mbps), μια ποσότητα που είναι αρκετά μεγάλη για upload bandwidth και έτσι εξηγείται γιατί με 20 γείτονες έχουμε καλύτερα αποτελέσματα στο chunk delivery time, από την περίπτωση που έχουμε 10 γείτονες.

2)% Rate Των Πακέτων Που Φτάνουν Στο Σύστημα Σε Μια Δεδομένη Χρονική Στιγμή



Στην παραπάνω εικόνα βλέπουμε τα ίδια αποτελέσματα με την εικόνα που αναλύσαμε παραπάνω. Μόνο που εδώ βλέπουμε τα αποτελέσματα εκφρασμένα σε ποσοστά επί της %.

3)% Average Rate Of Streaming



Στο τελευταίο γράφημα βλέπουμε την ποιότητα του streaming στη δεδομένη χρονική στιγμή των

(950.4 second) εκφρασμένη σε ποσοστά επί της %. Πιο συγκεκριμένα, βλέπουμε για κάθε peer πόσα πακέτα του βίντεο είναι στη σωστή σειρά και μπορούν να παιχτούν επί του συνόλου των πακέτων που έχουν λάβει και στο τέλος βγάζουμε το μέσο όρο από όλους τους peers. Παρατηρούμε, λοιπόν, στο παραπάνω γράφημα ότι όσο λιγότερους γείτονες έχουμε στην παράμετρο 1 τόσο καλύτερα είναι τα ποσοστά του streaming. Αυτό συμβαίνει γιατί, όπως έχουμε αναφέρει και πιο πάνω, όταν οι peers ανακοινώνουν τα πακέτα που έχουν σε περισσότερους γείτονες γεμίζουν οι transmission queues τους, προκαλώντας συμφόρηση στο δίκτυο και επιπλέον chunk delivery delay μέχρι να φτάσουν τα πακέτα που έχουν ζητήσει οι peers για να συνεχίσουν αποτελεσματικά το streaming.

Συμπεράσματα

1)Όπως αναφέραμε και παραπάνω οι pull αλγόριθμοι έχουν χειρότερο chunk delivery time από τους push αλγόριθμους.

2)Στο τελευταίο αλγόριθμο (P2P-TV Streaming System) οι peers ζητούν πακέτα μόνο από γειτονικούς peers στο overlay, ενώ στον αλγόριθμο Prime κατά τη διάρκεια του swarming οι peers μπορεί να ζητήσουν και πακέτα από peers που βρίσκονται σε μακρινή απόσταση στο overlay. Εξαιτίας αυτού το διαθέσιμο bandwidth για συνδέσεις απομακρυσμένων peers δεν επαρκεί, με αποτέλεσμα να υπάρχει κάποια καθυστέρηση στην αποστολή των πακέτων του βίντεο.

3) Υπάρχει μία αναλογία μεταξύ του αριθμού των γειτόνων με τους οποίους επικοινωνούν οι peers και του upload bandwidth που διαθέτουν. Peers με υψηλό upload bandwidth είναι προτιμότερο να επικοινωνούν με περισσότερους γείτονες έτσι ώστε να αξιοποιούν όλο το διαθέσιμο upload bandwidth τους και κατά συνέπεια να παρουσιάζουν καλύτερα αποτελέσματα σε όλες τις παραμέτρους.

7) Γενικά Συμπεράσματα Και Μελλοντικές Επεκτάσεις

Γενικά Συμπεράσματα

Αφού εξετάστηκαν αναλυτικά κατά περίπτωση τα αποτελέσματα που παρατηρήθηκαν για κάθε εφαρμογή που υλοποιήθηκε μπορούμε να φτάσουμε σε κάποια γενικά συμπεράσματα για εφαρμογές που στόχος τους είναι το streaming live content (π.χ. όπως βίντεο).

Το πρώτο σημαντικό συμπέρασμα που μπορεί κάποιος να εξάγει από τις παραλλαγές που υλοποιήθηκαν για κάθε εφαρμογή, είναι το πόσο σημαντικό είναι και πόσο βελτιώνει την ποιότητα του streaming μια καλύτερα διαβαθμισμένη (scalable) οργάνωση του overlay. Στην πλειονότητα των εφαρμογών που υλοποιήθηκαν και πιο συγκεκριμένα στην υλοποίηση του αλγορίθμου Prime σημαντικό ρόλο παίζει η αρχική επικοινωνία μεταξύ των πηγών και των κόμβων του $1^{ου}$ επιπέδου του diffusion tree, στα οποία φτάνει το overlay. Σε αυτό ακριβώς το σημείο είναι αναγκαίο να μειωθεί η συμφόρηση, έτσι ώστε να μην προστεθεί επιπλέον καθυστέρηση στη φάση του diffusion. Προσπαθώντας να ικανοποιηθεί η παραπάνω απαίτηση αντιμετωπίστηκε το εξής δίλλημα: 1) να μειωθεί ο αριθμός των peers του $1^{ου}$ επιπέδου, με συνέπεια να μειωθεί ο αριθμός των υποδέντρων του diffusion, γεγονός που με τη σειρά του θα προκαλέσει την αύξηση του βάθους (depth) των diffusion subtrees καθώς και την αύξηση της καθυστέρησης για να ολοκληρωθεί η φάση του diffusion, αφού θα χρειαστούν περισσότερα χρονικά διαστήματα για να καλυφθεί το μεγαλύτερο depth των diffusion subtrees ή 2) να χρησιμοποιηθούν παραπάνω peers στο $1^{ο}$ επίπεδο, έτσι ώστε να αυξηθούν τα diffusion subtrees επιφέροντας μεγαλύτερη scalability στο diffusion tree, προκαλώντας όμως συμφόρηση στην επικοινωνία πηγών- peers $1^{ου}$ επιπέδου. Στις δικές μας προσομοιώσεις έγινε προσπάθεια να λυθεί το παραπάνω αδιέξοδο με έναν λίγο διαφορετικό τρόπο, αυξάνοντας τον αριθμό των αρχικών πηγών που αρχίζουν να αποστέλλουν τα πακέτα του βίντεο στο δίκτυο. Παρατηρήθηκε λοιπόν ότι αυξάνοντας τον αριθμό των αρχικών πηγών βελτιώνονται όλες οι παράμετροι που καθορίζουν την ποιότητα του streaming σε όλους τους peers.

Δεύτερον, ιδιαίτερα σημαντικά είναι τα συμπεράσματα που βγήκαν συγκρίνοντας τις διάφορες παραλλαγές του αλγορίθμου Prime (Εφαρμογή 3) που υλοποιήθηκαν. Προστέθηκε, λοιπόν, η εξής νέα λειτουργία στην υλοποίηση του αλγορίθμου Prime, η επικοινωνία των peers, έτσι ώστε να πιστοποιούν ότι τελείωσε σε όλους η φάση diffusion και μπορεί να αρχίσει η φάση swarming, δηλαδή διαχωρίστηκαν οι φάσεις diffusion- swarming, έτσι ώστε η δεύτερη να αρχίζει μόνο όταν τελειώνει η πρώτη. Από την άλλη, υλοποιήθηκε και μια παραλλαγή του ίδιου αλγορίθμου, στον οποίο οι φάσεις diffusion- swarming δεν διαχωρίζονται. Παρατηρήθηκε, λοιπόν, ότι με διαχωρισμένες τις 2 φάσεις η παράμετρος του rate του streaming βελτιώνεται σημαντικά, αφού έτσι καταφέραμε να μειώσουμε το content bottleneck στη φάση του swarming, βρίσκοντας έναν τρόπο να διασφαλιστεί ότι όταν βρίσκεται σε εξέλιξη η φάση swarming έχουν φτάσει όλα τα πακέτα της φάσης diffusion στους peers. Μία επιπλέον σημαντική παρατήρηση που έγινε συγκρίνοντας τις 2 παραπάνω παραλλαγές του αλγορίθμου Prime αφορά τις προσομοιώσεις που έγιναν μειώνοντας σταδιακά το bandwidth στους peers. Αρχικά με το default bandwidth (10Mbps) στους peers, παράμετροι όπως το chunk delivery time και το startup

delay έχουν καλύτερες τιμές για την παραλλαγή χωρίς τον διαχωρισμό των 2 φάσεων. Όταν όμως μειώθηκε το bandwidth στους peers παρατηρήθηκε ότι το content bottleneck επηρέαζε δραστικά την παραλλαγή χωρίς τον διαχωρισμό των φάσεων με συνέπεια να πέφτουν κατακόρυφα οι επιδόσεις στο chunk delivery time και στο startup delay, ενώ στην παραλλαγή με τον διαχωρισμό των 2 φάσεων οι επιδόσεις στις ίδιες παραμέτρους παρέμεναν σχεδόν σταθερές παρόλο την μείωση του bandwidth στους peers. Έτσι εφόσον όσο μειώνεται το bandwidth, στην εφαρμογή με το μη διαχωρισμό των φάσεων, πέφτουν οι αποδόσεις στις παραπάνω παραμέτρους, ενώ στην εφαρμογή με διαχωρισμένες τις φάσεις οι αποδόσεις στις ίδιες παραμέτρους παραμένουν σταθερές. Κάποια στιγμή το chunk delivery time και το startup delay γίνεται καλύτερο στην εφαρμογή με διαχωρισμένες τις φάσεις, ενώ αρχικά αρχίσαμε με τα ακριβώς αντίστροφα αποτελέσματα για τις δύο παραλλαγές. Επίσης η παραλλαγή της Εφαρμογής 3 στην οποία η μετάδοση των πακέτων στη διάρκεια της φάσης του diffusion γίνεται με μία push διαδικασία από τα υψηλότερα επίπεδα στα χαμηλότερα επίπεδα του diffusion tree παρατηρήθηκε ότι οδηγεί σε καλύτερη χρήση του upload bandwidth όλων των κόμβων του overlay.

Τρίτον, συγκρίνοντας την Εφαρμογή 2, που είναι μια εφαρμογή τύπου servers- clients, αφού οι κόμβοι ζητούν και παίρνουν τα πακέτα του βίντεο μόνο από τις αρχικές πηγές, με την Εφαρμογή 3, που είναι οι προσομοιώσεις του αλγορίθμου Prime, στον οποίο οι peers ανταλλάσουν πακέτα προκειμένου να βελτιώσουν την ποιότητα του streaming, φαίνεται ότι το chunk delivery time είναι καλύτερο στις προσομοιώσεις του αλγορίθμου Prime. Αυτό σημαίνει ότι όλα τα πακέτα του βίντεο φτάνουν σε όλους τους peers του overlay πιο γρήγορα στις προσομοιώσεις του αλγορίθμου Prime.

Τέλος, προσομοιώνοντας έναν αλγόριθμου τύπου pull, παρατηρήθηκε ότι συγκρίνοντάς τον με τον αλγόριθμο Prime που είναι τύπου push, έχει χειρότερο chunk delivery time. Από την άλλη η απαίτηση της trading φάσης του pull αλγορίθμου οι peers να επικοινωνούν μόνο με τους γείτονές του, έχει ως συνέπεια να καταναλώνεται λιγότερο bandwidth από όσο χρειάζεται για την επικοινωνία στη φάση swarming του αλγορίθμου Prime, όπου οι peers των χαμηλότερων επιπέδων στο diffusion tree, με τον σχηματισμό του οποίου δημιουργείται το overlay, επικοινωνούν με τους peers του $1^{ου}$ επιπέδου του diffusion tree. Εκείνο που έγινε προσπάθεια να γίνει κατανοητό με τις προσομοιώσεις του pull αλγορίθμου είναι πόσο επηρεάζει την ποιότητα του streaming ο αριθμός των γειτόνων, στους οποίους οι peers ανακοινώνουν τα πακέτα που έχουν. Αυτό που διαπιστώθηκε γενικά είναι ότι όσο ο αριθμός των γειτόνων μειώνεται τόσο βελτιώνεται η ποιότητα του streaming, αφού μειώνεται η συμφόρηση στις transmission queues των peers που ανακοινώνουν τα πακέτα τους, αφού υπάρχουν αιτήματα για κάποιο πακέτο από λιγότερους γείτονες. Επιπλέον, καλό θα ήταν να έχουμε στο μυαλό μας όταν σχεδιάζονται τέτοιες pull τύπου εφαρμογές ότι υπάρχει μια άμεση σύνδεση του upload bandwidth με τον αριθμό των γειτόνων, στους οποίους ανακοινώνουν τα πακέτα τους. Θα ήταν αποδοτικό, δηλαδή, να γίνει κάποιος διαχωρισμός, έτσι ώστε οι peers όσο περισσότερο upload bandwidth έχουν με τόσο περισσότερους γείτονες να επικοινωνούν, προκειμένου να αξιοποιήσουν το upload bandwidth τους στο μέγιστο, μειώνοντας ταυτόχρονα και την πιθανότητα content bottleneck.

Μελλοντικές Επεκτάσεις

Μελετώντας αντίστοιχες εργασίες αναφορικά με τον αλγόριθμο Prime παρατηρήσαμε κάποιες βελτιστοποιήσεις που θα μπορούσαν να γίνουν στις υλοποιήσεις του αλγορίθμου Prime. Αυτό που θα μπορούσε να βελτιωθεί είναι οι καθυστερήσεις που παρατηρούνται κατά τη διάρκεια της φάσης του diffusion. Στις δικές μας προσομοιώσεις οι μεταδόσεις των διαφορετικών γκρουπ πακέτων στα διαφορετικά diffusion subtrees γίνονται παράλληλα, δηλαδή την ίδια χρονική στιγμή προκαλώντας μια σημαντική συμφόρηση στο overlay. Αυτό που θα μπορούσε να γίνει για να μειωθεί αυτή η συμφόρηση είναι να σειριοποιηθεί η μετάδοση των πακέτων σε peers του ίδιου επιπέδου του diffusion tree. Δηλαδή η μετάδοση 1 πακέτου κάθε διαφορετικού γκρουπ πακέτων στους peers του $1^{ου}$ επιπέδου από τις αρχικές πηγές να γίνεται σε διαφορετικές χρονικές στιγμές. Με αυτόν τον τρόπο εξασφαλίζεται επίσης ότι όσο ένας πατέρας μεταδίδει ένα πακέτο στον γιο του που βρίσκεται στο αμέσως πιο κάτω επίπεδο του ίδιου υποδέντρου, έχει φτάσει σε αυτόν τον πατέρα το επόμενο πακέτο του γκρουπ πακέτων, που αντιστοιχεί στο δικό του υποδέντρο, για να το μεταδώσει. Οι παραπάνω βελτιστοποιήσεις θα μπορούσαν να συμβάλλουν στην ελαχιστοποίηση του χρόνου που χρειάζεται για να ολοκληρωθεί η φάση diffusion στους peers, γεγονός που συμβάλει στη βελτίωση όλων των παραμέτρων που εξετάσαμε παραπάνω (π.χ. chunk delivery time, rate of streaming, startup delay), αφού όπως έχουμε αναφέρει ξανά το streaming υλοποιείται κατά τη διάρκεια της φάσης του swarming, με συνέπεια να μην μπορεί να αρχίσει η ζωντανή μετάδοση του βίντεο (streaming) πριν να ολοκληρωθεί η φάση του diffusion.

8) Αναφορές

Nazanin Magharei and Reza Rejaie, “PRIME: Peer- to- Peer Receiver-Driven Mesh-Based Streaming”, IEEE INFOCOM

Giuseppe Bianchi, Nicola Blefari Melazzi, Lorenzo Bracciale, Francesca Lo Piccolo, Stefano Salsano, “Fundamental delay bounds in peer-to-peer chunk-based real-time streaming systems”

Nazanin Magharei and Reza Rejaie, “Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches”, Yang Guo, Thomson Lab

Alessandra Carta, Marco Mellia, Michela Meo, Stefano Traverso, “Efficient Uplink Bandwidth Utilization in P2P- TV Streaming Systems”, Politecnico di Torino, Italy