τμήμα μηχανικών πληροφορικής τ.ε.
Τ.Ε.Ι. Δυτικής Ελλάδας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

“Τεχνικές ελέγχου συμφόρησης για το πρωτόκολλο TCP”

Καλπακιώρης Παύλος
ΑΜ : 1400

Γαλανοπούλου Άννα
ΑΜ : 1659

Επιβλέπων: Δρ Παξιμάδης Θ. Κωνσταντίνος

Ναύπακτος 2016

Περιεχόμενα

Περίληψη-Σκοπός

Το 1986 το διαδίκτυο αντιμετώπισε για πρώτη φορά σοβαρά προβλήματα συμφόρησης λόγω της εκρηκτικής ανάπτυξης της κυκλοφορίας, με κύριο αποτέλεσμα τη μεγάλη απώλεια πακέτων. Η βασική αιτία ήταν η εφαρμογή του πρωτοκόλλου μεταφοράς TCP. Σε αυτή την εργασία περιγράφονται αλγόριθμοι αποφυγής συμφόρησης που χρησιμοποιούνται για την αποφυγή φαινομένων συμφόρησης και κατ΄ επέκταση αθρόας απώλειας πακέτων στο δίκτυο. Οι αλγόριθμοι αυτοί υποχρεώνουν την κυκλοφορία του δικτύου να ακολουθεί μια πολιτική που ονομάζεται διατήρηση πακέτου.

Έχουν προταθεί διάφορες εκδόσεις του πρωτοκόλλου TCP για να αλλάξουν – συνεργαστούν με τους μηχανισμούς ελέγχου συμφόρησης και να βελτιώσουν την απόδοση. Βασικός μηχανισμός των αλγόριθμων αποφυγής συμφόρησης είναι ο μηχανισμός αργής εκκίνησης (slow start). Παρουσιάζουμε διάφορες εκδόσεις των αλγόριθμων αποφυγής συμφόρησης καθώς και τους μηχανισμούς που περιλαμβάνουν.

Ένα σημαντικό πρόβλημα στο πρωτόκολλο ελέγχου μεταφοράς TCP είναι ότι δεν είναι αρκετά γρήγορο στην αποφυγή συμφόρησης και ότι δεν έχει καλή απόδοση σε μεταδόσεις μεγάλων αποστάσεων με αποτέλεσμα να μην αξιοποιεί πλήρως τους πόρους σε διαδρομές με μεγάλο bandwidth delay product.
Είχαν προταθεί εκδόσεις του TCP που όντας «επιθετικές» δεν απέδωσαν τα επιθυμητά αποτελέσματα λόγω έλλειψης δικαιοσύνης. Εδώ παρουσιάζουμε έναν μηχανισμό αποφυγής συμφόρησης βασισμένο στην καθυστέρηση πακέτων (delay based), έχει επιθετική συμπεριφορά μόνο σε υπο-χρησιμοποιούμενες συνδέσεις. Χρησιμοποιεί τον πιο συντηρητικό αλγόριθμο TCP-Reno όπου εφαρμόζει το μηχανισμό AIMD (addictive increase/multiplicative decrease).

Επίσης παρουσιάζονται και άλλες παραλλαγές του TCP (TCP Tahoe, TCP Reno, TCP New Reno, Sack TCP, TCP Vegas, HighSpeed TCP, το Scalable TCP).

Τέλος παρουσιάζονται και αναλύονται τα αποτελέσματα της προσομοίωσης αυτών των μηχανισμών σε περιβάλλον NS2.

Abstract

In 1986 the Internet had to face, for the first time, serious congestion problems due to the explosive growth of traffic, main effect of high packet loss. The main reason was the implementation of the transport protocol TCP. In this paper described congestion avoidance algorithms used to avoid bottlenecks and therefore massive packet loss in the network achieving stability. These algorithms require network traffic to follow a policy called retention package.

Various TCP protocol versions have been proposed to change-cooperate with the congestion control mechanisms and improve performance. The basic mechanism of congestion avoidance algorithm is the slow start mechanism (slow start). We present several versions of congestion avoidance algorithms as well as the mechanisms were included.

A major problem in the transmission control protocol is that it is not fast enough to avoid congestion and does not perform well over long distance transmissions so it does not fully utilize the resources to paths with large bandwidth delay product.
Some being “aggressive” TCP versions were suggested but without having the desired results due to lack of justice. Here we present a congestion avoidance mechanism based on packet delay (delay based), has an aggressive behavior only in underutilized links.
Uses to the most conservative algorithm TCP-Reno which applies the AIMD (addictive increase /multiplicative decrease) mechanism.

Also introduce other TCP variations (TCP Tahoe, TCP Reno, TCP New Reno, Sack TCP, TCP Vegas, HighSpeed TCP, Scalable TCP).

Finally we present and analyze the results of the simulation of these mechanisms in an NS2 environment.

Κεφάλαιο 1 Εισαγωγή

Από τις αρχές του 1980 παρατηρείται μια εκρηκτική αύξηση στα δίκτυα υπολογιστών. Εκατοντάδες πανεπιστήμια και κρατικές εγκαταστάσεις συνδέονταν χρησιμοποιώντας μισθωμένες τηλεφωνικές γραμμές. Το TCP ή πρωτόκολλο ελέγχου μετάδοσης βρίσκεται στο επίπεδο μεταφοράς (transport layer), είναι ένα αξιόπιστο πρωτόκολλο το οποίο επιτρέπει σε μια ροή byte που προέρχεται από μια μηχανή να παραδίδεται χωρίς σφάλματα σε οποιαδήποτε μηχανή στο δίκτυο. Διαθέτει έναν μηχανισμό ελέγχου ροής, όπου με τη χρήση των επιβεβαιώσεων (acknowledgements) ενός μηχανισμού κυλιόμενου παραθύρου ο παραλήπτης είναι σε θέση να ελέγξει τη ροή δεδομένων. Ο παραλήπτης ορίζει το μέγεθος του παραθύρου (σε bytes) το οποίο χρησιμοποιείται από τον αποστολέα ως ανώτερο όριο για τα δεδομένα που μπορεί να στείλει χωρίς να έχει λάβει επιβεβαίωση. Κατά τον τρόπο αυτό εξασφαλίζεται πως ο αποστολέας δε θα στείλει ταχύτερα απ' ότι μπορεί να απορροφήσει ο παραλήπτης. Επιπρόσθετα υλοποιεί έναν ιδιαίτερα συντονισμένο μηχανισμό ελέγχου συμφόρησης.

Οι πιο σημαντικοί αλγόριθμοι ελέγχου συμφόρησης είναι οι : *Αργής εκκίνησης* (slow start), *αποφυγής συμφόρησης* (congestion avoidance), *γρήγορης επαναμετάδοσης* (fast retransmit), *γρήγορης ανάκαμψης* (fast recovery). Αλγόριθμος αργής εκκίνησης όπου ο ρυθμός με τον οποίο μεταδίδονται τα πακέτα από τον αποστολέα στο δίκτυο είναι ίσος με τον αριθμό των επιβεβαιώσεων που επιστρέφουν στο άλλο άκρο της σύνδεσης. Στον αλγόριθμο αποφυγής συμφόρησης η απώλεια ενός πακέτου είναι δείγμα συμφόρησης κάπου στο δίκτυο, υπάρχουν δύο ενδείξεις για την απώλεια ενός πακέτου. Η πρώτη είναι η εκδήλωση ενός timeout και η δεύτερη είναι τα διπλά ACK που λαμβάνει ο δέκτης. Στον αλγόριθμο γρήγορης επαναμετάδοσης σύμφωνα με το μηχανισμό αυτό το TCP προσπαθεί να αποφύγει την εκδήλωση του timeout προτού μεταδοθεί ένα χαμένο πακέτο. Τέλος, ο αλγόριθμος γρήγορης ανάκαμψης μετά από την επαναμετάδοση του χαμένου πακέτου εφαρμόζεται αντί του αλγορίθμου αργής εκκίνησης.

Η ιδέα του μηχανισμού ελέγχου συμφόρησης όπου περιορίζει την ταχύτητα αποστολής των δεδομένων ώστε αποστολέας να μην υπερφορτώνει το δίκτυο και να διατηρεί μία μεταβλητή κατάσταση για κάθε σύνδεση, ονομάζεται congestion window (παράθυρο συμφόρησης). Χρησιμοποιείται από την πηγή για να περιορίσει το πλήθος των δεδομένων που επιτρέπεται να βρίσκονται σε μετάβαση σε μία δεδομένη χρονική στιγμή. Υπάρχουν διάφορες κατηγορίες TCP αλγορίθμων ελέγχου συμφόρησης όπου θα αναλυθούν στο παρακάτω κείμενο.

Κεφάλαιο 2 Έλεγχος και αποφυγή συμφόρησης

2.1 Εισαγωγή

Τα δίκτυα υπολογιστών είχαν υποστεί μια ραγδαία ανάπτυξη κατά το τέλος της δεκαετίας του 1980, με αποτέλεσμα να υπόκεινται σε προβλήματα συμφόρησης. Παρατηρήθηκαν επίσης απώλειες 10% των εισερχόμενων πακέτων λόγω των τοπικών υπερχειλίσεων των buffer. Ένα μεγάλο μέρος της αιτίας ήταν στην εφαρμογή του πρωτοκόλλου μεταφοράς. Στην απόκριση συμφόρησης του δικτύου οι προφανείς τρόποι εφαρμογής σε ένα πρωτόκολλο μεταφοράς βασισμένο σε παράθυρα, μπορούσε να οδηγήσει σε λάθος συμπεριφορά. Οι αλγόριθμοι βασίζονταν στην ιδέα της επίτευξης της σταθερότητας του δικτύου, αυτό επιτεύχθηκε με μια αρχή που ονομάστηκε "packet conservation" (διατήρηση πακέτου). Τον Οκτώβριο του 86, το διαδίκτυο είχε για πρώτη φορά μια σειρά από «καταρρεύσεις συμφόρησης», το bandwidth δεδομένων υπέστη σοβαρή μείωση και ξαφνική πτώση χιλιάδων πακέτων. Το 4.3BSD (Berkeley UNIX) TCP είχε λανθασμένη συμπεριφορά και μπορούσε να υποστεί ρυθμίσεις ώστε να λειτουργεί καλύτερα κάτω από άθλιες συνθήκες δικτύου.

Από εκείνη τη στιγμή, είχαν μπει επτά νέοι αλγόριθμοι στο 4 BSD TCP (Berkeley Software Distribution)

(i) round-trip-time variance estimation
(ii) exponential retransmit timer backoff
(iii) slow-start
(iv) more aggressive receiver ack policy
(v) dynamic window sizing on congestion
(vi) Karn's clamped retransmit backoff
(vii) fast retransmit

Οι αλγόριθμοι (i) και (v) πηγάζουν από μια παρατήρηση: Μια ροή σε μια σύνδεση TCP πρέπει να υπακούει μια αρχή «διατήρησης πακέτου», δηλ., μια σύνδεση «στην ισορροπία» όπου έχει σταθερό τρέξιμο με ένα πλήρες παράθυρο δεδομένων κατά τη μεταφορά, η ροή πακέτων είναι "συντηρητική" δηλ. ένα νέο πακέτο δεν τίθεται στο δίκτυο έως ότου φύγει ένα παλαιό πακέτο.

Υπάρχουν μόνο τρεις τρόποι για να αποτύχει η διατήρηση πακέτων:
(1) Η σύνδεση δεν φτάνει στην ισορροπία, ή
(2) Ένας αποστολέας εγχέει ένα νέο πακέτο προτού να βγει ένα παλαιό πακέτο, ή
(3) Η ισορροπία δεν μπορεί να επιτευχθεί λόγω των ορίων των πόρων κατά μήκος της πορείας.

(1) Αποτυχία: πρέπει να είναι από μια σύνδεση που είτε αρχίζει είτε ξανά ξεκινά μετά την απώλεια ενός πακέτου. Ένας άλλος τρόπος να εξεταστεί είναι να πούμε ότι ο αποστολέας χρησιμοποιεί acks όπως ένα «ρολόι» για να ελέγξει τα νέα πακέτα στο δίκτυο. Δεδομένου ότι ο δέκτης μπορεί να δημιουργήσει acks όχι γρηγορότερα απ' ότι τα πακέτα δεδομένων μπορούν να πάρουν.

2.2 Έλεγχος Ροής Παραθύρου 'Self-clocking'

Το "self clocking" σύστημα, [1] προσαρμόζεται αυτόματα στο εύρος ζώνης και στις παραλλαγές καθυστέρησης και έχει μια ευρεία δυναμική περιοχή. Τα χαρακτηριστικά που κάνουν το "self clocking" σταθερό σύστημα όταν είναι σε λειτουργία το καθιστά και δύσκολο να ξεκινήσει. Για να ξεκινήσει το «ρολόι», έχει αναπτυχθεί ο αλγόριθμος Slow-start για την σταδιακή αύξηση ποσότητας των δεδομένων κατά την μεταφορά.

2.3 Slow-start

Η μέθοδος αύξησης του παραθύρου slow-start δεν είναι ότι είναι αργή: παίρνει το χρόνο $R\log_2 W$ όπου R είναι ο round-trip-time και W είναι το μέγεθος. Αυτό σημαίνει ότι το παράθυρο ανοίγει αρκετά γρήγορα και έχει μια αμελητέα επίδραση στην απόδοση, ακόμη και στις συνδέσεις με ένα μεγάλο bandwidth–delay product. Ο αλγόριθμος εγγυάται ότι το εύρος του παραθύρου συμφόρησης δεν μπορεί να το ξεπεράσει το παράθυρο αποστολής που διαφημίζει ο δέκτης στον αποστολέα.

- Προσθέτει ένα παράθυρο συμφόρησης, cwnd (Congestion Window), στην κατάσταση ανά σύνδεση.
- Κατά την εκκίνηση ή επανεκκίνηση μετά από μια απώλεια, θέτει το cwnd σε ένα πακέτο.
- Σε κάθε ack για νέα δεδομένα, αυξάνει το cwnd εκθετικά.
- Κατά την αποστολή, στέλνει το μήκος παραθύρου που ανακοίνωσε ο παραλήπτης στον αποστολέα και το cwnd.

2.3.1 Διατήρηση σε ισορροπία: Round Trip Timing

Όταν τα δεδομένα ρέουν αξιόπιστα τα προβλήματα (2) και (3) θα πρέπει να αντιμετωπιστούν. Υποθέτοντας ότι η εφαρμογή του πρωτοκόλλου είναι σωστή, το (2) πρέπει να αντιπροσωπεύει μια αποτυχία στο χρονόμετρο αναμετάδοσης του αποστολέα. Ένας καλός εκτιμητής RTT, όπου είναι ο πυρήνας του χρονομέτρου αναμετάδοσης, είναι το πιο σημαντικό χαρακτηριστικό της κάθε εφαρμογής του πρωτοκόλλου που αναμένει να επιβιώσει ένα βαρύ φορτίο.

Οι προδιαγραφές του πρωτοκόλλου TCP υποδηλώνουν την εκτίμηση του round trip time μέσω του low-pass φίλτρου.

$$R \leftarrow aR + (1-a)M$$

Όπου R είναι η μέση εκτίμηση RTT, το M είναι μια μέτρηση του RTT από το πιο πρόσφατο acked πακέτο, και το a είναι ένα φίλτρο σταθερού gain (κέρδους) με προτεινόμενη τιμή του 0,9. Μόλις η εκτίμηση R είναι ενημερωμένη, το διάστημα αναμετάδοσης timeout, RTO, για το επόμενο πακέτο που αποστέλλεται έχει οριστεί σε βR. Το προτεινόμενο β είναι ίσο με 2. Πάνω από αυτό το σημείο, η σύνδεση θα ανταποκριθεί για να αυξήσει το φορτίο αναμεταδίδοντας μόνο στα πακέτα που έχουν καθυστερήσει κατά τη μεταφορά. Αυτό αναγκάζει το δίκτυο για να κάνει άχρηστη εργασία και σπαταλά το εύρος ζώνης στα αντίγραφα των πακέτων που τελικά παραδίδεται.

2.4 Προσαρμογή στη διαδρομή: αποφυγή συμφόρησης

Αν τα χρονόμετρα είναι σε καλή κατάσταση, είναι δυνατόν να δηλώνουν με βεβαιότητα ότι ένα timeout υποδεικνύει ένα χαμένο πακέτο και όχι ένα προβληματικό χρονόμετρο.

Τα πακέτα χάνονται για δύο λόγους:

- Έχουν υποστεί ζημιά κατά τη μεταφορά, ή
- Το δίκτυο είναι συμφορημένο και κάπου στην πορεία υπήρξε ανεπάρκεια στη χωρητικότητα του buffer (διαμεσολαβητής) .

Στις περισσότερες διαδρομές του δικτύου, η απώλεια που οφείλεται σε βλάβη είναι σπάνια (<<1%), έτσι είναι πιθανότερο πως μια απώλεια πακέτων οφείλεται σε συμφόρηση του δικτύου.
Μια στρατηγική « αποφυγής συμφόρησης », θα έχει δύο συστατικά :

- Το δίκτυο θα πρέπει να είναι σε θέση να επισημάνει τα σημεία τέλους μεταφοράς όταν η συμφόρηση συμβαίνει (ή πρόκειται να συμβεί).
- Τα σημεία τέλους πρέπει να έχουν μια πολιτική όπου θα μειώνει τη χρησιμοποίηση εάν αυτό το σήμα λαμβάνεται και θα αυξάνει τη χρησιμοποίηση εάν το σήμα δεν έχει ληφθεί.

Εάν η απώλεια πακέτων είναι (σχεδόν) πάντα εξαιτίας της συμφόρησης και αν ένα timeout (σχεδόν) πάντα οφείλεται σε ένα πακέτο που χάνεται, έχουμε έναν καλό σήμα

που προειδοποιεί για το «συμφορημένο δίκτυο», ιδιαίτερα δεδομένου ότι το σήμα αυτό παραδίδεται αυτόματα από όλα τα υφιστάμενα δίκτυα, χωρίς ειδική τροποποίηση.
Το άλλο μέρος της στρατηγικής αποφυγής συμφόρησης, είναι η δράση end node, που είναι σχεδόν ταυτόσημη στο σύστημα DEC / ISO και TCP και προκύπτει άμεσα από ένα μοντέλο πρώτης τάξης χρονολογικών σειρών του δικτύου.

Η καλύτερη πολιτική αύξησης είναι να κάνει μικρές, σταθερές αλλαγές στο μέγεθος του παραθύρου.

- Σε κάθε timeout, ορίζει το cwnd στο μισό του τρέχοντος παραθύρου (αυτή είναι πολλαπλασιαστική μείωση).

- Σε κάθε ACK για νέα δεδομένα, αυξάνει το cwnd κατά 1 / cwnd (αυτή είναι η προσθετική αύξηση).

- Κατά την αποστολή, στέλνεται το μήκος παραθύρου που ανακοίνωσε ο παραλήπτης στον αποστολέα και το cwnd.

Ο αλγόριθμος αυτός είναι μόνο αποφυγής συμφόρησης (congestion avoidance), δεν περιλαμβάνει την “slow-start”. Δεδομένου ότι η απώλεια πακέτων που σηματοδοτεί η συμφόρηση θα οδηγήσει σε μια νέα αρχή, είναι σχεδόν βέβαιο ότι θα είναι αναγκαία η slow start εκτός από τα παραπάνω. Επειδή και οι δυο αλγόριθμοι “congestion avoidance” και “slow-start” προκαλούνται από ένα διάλειμμα, χειρίζονται το παράθυρο συμφόρησης. Είναι πραγματικά ανεξάρτητοι αλγόριθμοι με απολύτως διαφορετικούς στόχους. Οι δύο αλγόριθμοι πρέπει να παρουσιαστούν χωριστά ακόμα κι αν στην πράξη πρέπει να εφαρμοστούν από κοινού.

2.5 Έλεγχος συμφόρησης από πλευράς gateway

Ενώ οι αλγόριθμοι στα άκρα των μεταφορών μπορούν να εξασφαλίσουν πως δεν γίνεται υπέρβαση της χωρητικότητας του δικτύου, δεν μπορούν να εξασφαλίσουν τη δίκαιη κατανομή αυτής. Μόνο σε πύλες, κατά τη σύγκλιση των ροών, υπάρχουν αρκετές πληροφορίες για τον έλεγχο της κοινής χρήσης και δίκαιης κατανομής εκεί. Έτσι, φαίνεται ότι ο αλγόριθμος « ανίχνευσης συμφόρησης» είναι το επόμενο μεγάλο βήμα.

Ο στόχος αυτού του αλγορίθμου είναι να στείλει ένα μήνυμα προς τους end nodes όσο το δυνατόν νωρίτερα. Από τη στιγμή που συνεχίζεται η χρησιμοποίηση πτώσεων πακέτων ως σήμα συμφόρησης, η πύλη « αυτοπροστασίας » από μια λανθασμένη συμπεριφορά υποδοχής θα πρέπει να πέσει “έξω”: Η υποδοχή θα έχει απλά τα περισσότερα από τα πακέτα που έπεσαν καθώς η πύλη προσπαθεί να πει ότι χρησιμοποιεί περισσότερο από το μερίδιο που του αναλογεί. Κατά συνέπεια, όπως ο αλγόριθμος end node, ο αλγόριθμος πυλών πρέπει να μειώσει τη συμφόρηση ακόμα κι αν κανένα end node δεν τροποποιείται για να κάνει τη αποφυγή συμφόρησης. Οι κόμβοι που εφαρμόζουν την αποφυγή συμφόρησης θα πάρουν το δίκαιο μερίδιο του εύρους ζώνης

και ένα ελάχιστο αριθμό packet drops.
Δεδομένου ότι η συμφόρηση αυξάνεται εκθετικά, το να ανιχνευθεί νωρίς είναι σημαντικό. Εάν ανιχνευθεί νωρίς, μικρές αναπροσαρμογές στα παράθυρα των αποστολέων θα το θεραπεύσουν. Διαφορετικά, μαζικές προσαρμογές είναι αναγκαίες προκειμένου να δώσουν στο δίκτυο αρκετή χωρητικότητα για να αντλήσουν από το απόθεμα. Η αξιόπιστη ανίχνευση της πληροφορίας είναι ένα μη τετριμμένο πρόβλημα.

2.5.1 Γρήγορος αλγόριθμος για RTT σημαίνει και μεταβολή

Ο αλγόριθμος RFC793 για τον υπολογισμό του RTT ανήκει σε μια κατηγορία εκτιμητών που ονομάζονται αναδρομικά σφάλματα πρόβλεψης ή στοχαστική κλίση αλγορίθμων. Λαμβάνοντας υπόψη μια νέα μέτρηση m από το RTT, το TCP ενημερώνει μια εκτίμηση του μέσου όρου RTT **α** από όπου g είναι το "κέρδος" (0 < g < 1) που πρέπει να σχετίζεται με το m (λόγο σήματος προς θόρυβο).

$$\alpha \leftarrow (1\text{-}g)\alpha + gm$$

Αυτό έχει μια λογική και υπολογίζει γρηγορότερα, εάν γίνει αναδιάταξη και συλλογή όρων όπου είναι πολλαπλασιασμένοι με το g

$$a \leftarrow \alpha + g(m - a)$$

Το α είναι πρόβλεψη της επόμενης μέτρησης m - α το οποίο είναι το σφάλμα στην εν λόγω πρόβλεψη και η παραπάνω έκφραση λέει ότι κάνει μια νέα πρόβλεψη με βάση την παλαιά.

2.5.2 Η συνδυασμένη αργή εκκίνηση με τον αλγόριθμο αποφυγής συμφόρησης

Ο αποστολέας διατηρεί δύο μεταβλητές κατάστασης για τον έλεγχο συμφόρησης :

(1) ένα παράθυρο slow-start/congestion, cwnd

(2) ένα όριο μεγέθους, ssthresh (slow-start threshold), για εναλλαγή μεταξύ των δύο αλγορίθμων.

Ο αποστολέας στέλνει πάντα το μήκος παραθύρου που ανακοίνωσε ο παραλήπτης στον αποστολέα και το cwnd.
Σε ένα διάλειμμα, το μισό μέγεθος του τρέχοντος παραθύρου καταγράφεται στο ssthresh (αυτό είναι το πολλαπλασιαστικό μέρος μείωσης του αλγορίθμου αποφυγής συμφόρησης), το cwnd έπειτα τίθεται σε 1 πακέτο (αυτό ξεκινάει την slow-start). Όταν τα νέα δεδομένα είναι acked, ο αποστολέας :

```
if (cwnd < ssthresh)        / Αν βρίσκεται ακόμα σε slow−start ανοίγει το παράθυρο εκθετικά /
    cwnd += 1;
else                        / Αλλιώς κάνει αποφυγή συμφόρησης, αυξάνει κατά 1 /
    cwnd += 1/cwnd;
```

Έτσι η αργή εκκίνηση ανοίγει το παράθυρο γρήγορα σε όποια αποφυγή συμφόρησης νομίζει ότι είναι ένα ασφαλές σημείο λειτουργίας, τότε η αποφυγή συμφόρησης αναλαμβάνει και σιγά-σιγά αυξάνει το μέγεθος του παραθύρου για να εξετάσει το μεγαλύτερο διαθέσιμο εύρος ζώνης στην πορεία.

2.5.3 Αλληλεπίδραση με τη ρύθμιση παραθύρου RTT

Ορισμένες συνδέσεις TCP, ιδιαίτερα πάνω από μια σύνδεση με πολύ χαμηλή ταχύτητα, όπως σε μια dial-up line, αντιμετώπιζαν μια ατυχή αλληλεπίδραση ανάμεσα στη ρύθμιση συμφόρησης παραθύρου και στο συγχρονισμό αναμετάδοσης.

Οι διαδρομές δικτύων διαιρούνται σε δύο κατηγορίες:

- Delay-dominated, όπου οι αποθηκεύσεις και μεταβιβάσεις ή/και καθυστερήσεις διέλευσης καθορίζουν το RTT
- Bandwidth-dominated, όπου η (συμφόρηση) σύνδεση bandwidth και το μέσο πακέτο καθορίζουν το RTT

Σε μια bandwidth-dominated διαδρομή του εύρους ζώνης **b**, μια αποφυγής συμφόρησης παραθύρου του Δw θα αυξήσει το RTT των πακέτων μετά την αύξηση κατά

$$\Delta R \approx \Delta w/b$$

Η πλαστή αναμετάδοση λόγω μιας αύξησης παραθύρου μπορεί να εμφανιστεί κατά τη διάρκεια της αποφυγής συμφόρησης. Το παράθυρο μπορεί να αυξηθεί κατά 1 πακέτο.

Τα timeouts χρησιμοποιούνται για να μειώσουν το παράθυρο σε κάτι πιο κατάλληλο για τη διαδρομή. Οι slow start και congestion avoidance αλγόριθμοι έχουν σκοπό να μην προκαλέσουν πλαστή αναμετάδοση.

Τα πρωτόκολλα εφαρμογής SMTP και NNTP έχουν τάση "διαπραγμάτευσης" όπου ανταλλάσσονται μερικά πακέτα stop and wait και στη συνέχεια είναι η φάση της μεταφοράς δεδομένων. Δυστυχώς, οι ανταλλαγές «διαπραγμάτευσης» ανοίγουν το παράθυρο συμφόρησης, έτσι η έναρξη της φάσης μεταφοράς δεδομένων θα πετάξει διάφορα πακέτα στο δίκτυο χωρίς slow start και σε μια bandwidth-dominated διαδρομή, γρηγορότερα απ' ότι το RTO μπορεί να ακολουθήσει την αύξηση του RTT που έχει προέλθει από αυτά τα πακέτα.

2.6 Πολιτική Προσαρμογής Παραθύρου

Η χρησιμοποίηση του ½^{ου} ως όρο μείωσης, βασίστηκε στην ακόλουθη παρατήρηση. Όταν ένα πακέτο απορρίπτεται, είτε στην εκκίνηση (ή επανεκκίνηση μετά από πτώση) είτε σε σταθερή κατάσταση αποστολής αν ξεκινήσει, ξέρει ότι το μισό του τρέχοντος μεγέθους του παραθύρου « δούλεψε ». Συνεπώς, η αξία των πακέτων ενός παραθύρου ανταλλάχθηκε χωρίς πτώσεις (η αργή εκκίνηση το εγγυάται αυτό). Εάν η σύνδεση είναι σε κατάσταση που τρέχει και ένα πακέτο χαθεί, είναι πιθανώς επειδή μια νέα σύνδεση ξεκίνησε και πήρε κάποιο από το εύρος ζώνης.

Παρά το γεγονός ότι ένας παράγοντας δυο αλλαγών στο μέγεθος του παραθύρου μοιάζει μια μεγάλη ποινή απόδοσης στους όρους συστημάτων, το κόστος είναι αμελητέο, τα πακέτα πέφτουν μόνο όταν διαμορφώνεται μεγάλη ουρά. Για να αναγκαστούν οι αποστολείς να μειώσουν τα παράθυρά τους, η συμφόρηση βρίσκεται σε χρησιμοποίηση 100% χωρίς υπερβολικό εύρος ζώνης διαθέσιμο για να διαλύσει την ουρά. Εάν ένα πακέτο είναι dropped, κάποιος αποστολέας κλείνει για δύο RTT, είναι ακριβώς ο αναγκαίος χρόνος για να αδειάσει η ουρά αναμονής. Κατά συνέπεια η καθυστέρηση έχει μειωθεί στο ελάχιστο χωρίς το σύστημα να χάνει οποιαδήποτε συμφόρηση εύρους ζώνης.

Κεφάλαιο 3 Έλεγχος συμφόρησης TCP Window Based

3.1 Εισαγωγή

Διάφορες προτάσεις έχουν υποβληθεί για να αλλάξουν τους μηχανισμούς ελέγχου συμφόρησης TCP για να βελτιώσουν την απόδοση του. Μια νέα έρευνα για την αποφυγή συμφόρησης τείνει να μειώνει την ταχύτητα, σε αυτή την ομάδα είναι οι window based αλγόριθμοι ελέγχου συμφόρησης που χρησιμοποιούν το μέγεθος του παραθύρου συμφόρησης για να καθορίσουν την ταχύτητα μετάδοσης. Οι δύο κύριοι αλγόριθμοι window based ελέγχου συμφόρησης όπως είδαμε και στο προηγούμενο κεφάλαιο είναι congestion avoidance και slow start.

Μεγάλη προσοχή τοποθετήθηκε στην slow start όπου υπάρχουν 3 γραμμές έρευνας για τη βελτίωση της: Μια ομάδα χρησιμοποίησε την εκτίμηση ορισμένων παραμέτρων για να καθορίσει την αρχική ταχύτητα, η δεύτερη ομάδα χρησιμοποίησε την εκτίμηση bandwidth ενώ η τρίτη ομάδα χρησιμοποίησε το ρητό αίτημα για τη βοήθεια δικτύων να καθορίσει την αρχική ταχύτητα ξεκινήματος. Αναλύθηκαν τα προβλήματα της κάθε πρότασης και προτάθηκε ένα πολλαπλάσιο ξεκίνημα για το TCP, όπου διευκρινίζουν ότι η ταχύτητα ξεκινήματος είναι επιλέξιμη από ένα n-arry σύνολο αλγορίθμων.

Υπάρχουν δυο τεχνικές ελέγχου συμφόρησης η window-based και η rate-based. Στη window-based προσέγγιση, ο ρυθμός μετάδοσης δεδομένων ρυθμίζεται με βάση τον καθορισμό ενός μεγέθους παραθύρου συμφόρησης κάνοντας χρήση του μηχανισμού AIMD (Additive Increase / Multiplicative Decrease). Ενώ στη rate-based τεχνική ο ένα σύνολο εξισώσεων χρησιμοποιείται για να ελέγχει την ταχύτητα μετάδοσης. Το TCP χρησιμοποιεί τη window-based προσέγγιση ως τεχνική ελέγχου συμφόρησης.

3.2 Παραλλαγές του TCP

Οι παραλλαγές του TCP, που είναι ενδιαφέρουσες, περιλαμβάνουν εκείνες που εφαρμόστηκαν ήδη, όμως για να εφαρμοστούν υιοθέτησαν το συμβατικό αλγόριθμο slow start.

3.2.1 TCP Tahoe

Κυκλοφόρησε το 1988 από τον V. Jacobson και περιέχει τον AIMD ο οποίος είναι ο μηχανισμός ελέγχου του. Το TCP Tahoe ελέγχει τη συμφόρηση ως εξής: Όταν το παράθυρο συμφόρησης είναι κάτω από το κατώτατο όριο, το παράθυρο συμφόρησης αυξάνεται εκθετικά (επιβραδύνετε το επίπεδο έναρξης). Όταν το παράθυρο συμφόρησης είναι επάνω από το κατώτατο όριο το παράθυρο συμφόρησης αυξάνεται γραμμικά (προσθετική αύξηση) δηλ. αποφυγή συμφόρησης. Όποτε υπάρχει ένα διάλειμμα, το κατώτατο όριο τίθεται στο ήμισυ του τρέχοντος παραθύρου συμφόρησης και το παράθυρο συμφόρησης τίθεται σε ένα, ενώ το πακέτο αναμεταδίδεται (πολλαπλασιαστική μείωση).

Οι αλγόριθμοι που εφαρμόζονται είναι slow start και congestion avoidance .

3.2.2 TCP Reno

Το Tahoe τροποποιήθηκε από το Reno όπου θέτει το παράθυρο συμφόρησης σε ένα πακέτο όταν έχει timeout (RTO). Το Reno επέκτεινε τον αλγόριθμο για να περιλάβει το μηχανισμό γρήγορης αναμετάδοσης. Οι αλγόριθμοι που εφαρμόζονται στο Reno είναι οι slow start, congestion avoidance, fast retransmit και fast recovery.

Το TCP Reno εφαρμόζει το μηχανισμό AIMD (Additive increase/multiplicative decrease)του TCP, αυξάνει το παράθυρο συμφόρησης W από ένα segment ανά RTT για κάθε λαμβανόμενο ack και μειώνει κατά το ήμισυ το παράθυρο συμφόρησης για κάθε γεγονός απώλειας ανά RTT. Το TCP Reno ελέγχει το παράθυρο συμφόρησης όπως ακολουθεί:

$$\text{Increase: } W=W + (1 / W)$$

$$\text{Decrease: } W=W- ((1 / 2)W)$$

Επαναλαμβάνει περιοδικά την αύξηση και μείωση παραθύρου και απαιτεί εξαιρετικά μικρό packet loss για να διατηρήσει ένα μεγάλο παράθυρο που δεν είναι δυνατό στην πραγματικότητα.

3.2.3 TCP New Reno

Σημειώνει το μέγιστο σημαντικό πακέτο **s** καθώς εισάγεται στη fast recovery. Όταν ένα νέο ack παραλαμβάνεται και αναγνωρίζει όλα τα σημαντικά πακέτα, η fast recovery εξέρχεται και το cwnd τίθεται στη μισή αξία του ssthres και έπειτα διέρχεται στο στάδιο αποφυγής συμφόρησης. Αλλά, εάν παραλαμβάνεται ένα μερικό ack, υποθέτει ότι το επόμενο πακέτο στη σύνδεση χάνεται και προσπαθεί να αναμεταδώσει.
Βγαίνει σε fast recovery όταν αναγνωρίζονται όλα τα στοιχεία στο παράθυρο

3.2.4 SACK (TCP με επιλεκτική αναγνώριση)

Το new Reno πάσχει από το γεγονός ότι η ανίχνευση κάθε απώλειας πακέτων διαρκεί ένα RTT. Το SACK είναι μια επέκταση του πρωτοκόλλου TCP Reno και του new Reno, αναγνωρίζει επιλεκτικά την επιβεβαίωση των πακέτων. Αυτό απαιτεί κάθε ack να περιέχει μια καταχώρηση για όποιο πακέτο έχει αναγνωριστεί. Αυτό επιτρέπει στον αποστολέα να υπολογίσει ποια πακέτα έχουν υπολογιστεί και ποια εκκρεμούν. Ο SACK διατηρεί τη slow start και τη γρήγορη αναμετάδοση του Reno.

Ορίζει ότι κάθε φορά που ο αποστολέας θα τεθεί σε γρήγορη κατάσταση ανάκτησης, μια μεταβλητή "σωλήνα" αρχίζει να χρησιμοποιείται για την εκτίμηση του αριθμού των πακέτων που λείπουν κατά μήκος της διαδρομής. Θέτει έπειτα το μέγεθος του cwnd στο μισό του τρέχοντος μεγέθους του ως συνήθως. Κάθε φορά που παραλαμβάνεται ένα ack, το μέγεθος του σωλήνα μειώνεται κατά ένα και όταν διαβιβάζεται ένα πακέτο ή αναμεταδίδεται, ο σωλήνας αυξάνεται κατά ένα. Κάθε φορά που το μέγεθος του σωλήνα γίνεται μικρότερο τότε το cwnd, ελέγχει ποια πακέτα είναι ακόμη να αναγνωριστούν για να αναμεταδοθούν αμέσως. Εάν δεν υπάρχει κανένα σημαντικό ack, στέλνει ένα νέο πακέτο. Κατά συνέπεια ο αποστολέας στέλνει μόνο το νέο ή το αναμεταδιδόμενο πακέτο εάν ο σωλήνας είναι μικρότερος από το cwnd.

Με αυτόν τον τρόπο το SACK μπορεί να στείλει περισσότερα από ένα χαμένα πακέτα σε ένα ενιαίο RTT. Ένα σημαντικό μειονέκτημα του αλγορίθμου SACK είναι η σχετική δυσκολία στην εφαρμογή της επιλεκτικής αναγνώρισης.

3.2.5 TCP Vegas

Υπάρχει μια κατηγορία αλγορίθμων ελέγχου συμφόρησης που προσαρμόζουν το μέγεθος παραθύρου συμφόρησης με βάση την καθυστέρηση end to end. Αυτή η προσέγγιση παρουσιάζεται ως TCP Vegas.

- Στη slow start κατάσταση, ο έλεγχος συμφόρησης ενσωματώθηκε από μια σκόπιμη καθυστέρηση στην αύξηση παραθύρου συμφόρησης.

- Όταν ένα packet loss εμφανίζεται, το TCP Vegas μεταχειρίζεται την παραλαβή ορισμένων ACKs, ως ώθηση που ελέγχει εάν εμφανιστεί ένα timeout.

- Ενημερώνει το παράθυρο συμφόρησης βασισμένο στην καθυστέρηση end to end αντί της χρησιμοποίησης του packet loss ως παράμετρο αναπροσαρμογών.

Το Vegas επεκτείνει μια στρατηγική αναμετάδοσης του Reno. Καταγράφει πότε κάθε πακέτο στέλνεται και υπολογίζει μια εκτίμηση του RTT για κάθε μετάδοση. Αυτό επιτυγχάνεται με την παρακολούθηση του χρόνου κάθε επιβεβαίωσης ACK προς τον αποστολέα.

Κάθε φορά που ένα διπλότυπο ACK παραληφθεί, εκτελεί τον ακόλουθο έλεγχο:

if (current RTT > RTT estimate)

Αν αυτό είναι αλήθεια, αναμεταδίδει το πακέτο χωρίς να περιμένει για 3 διπλά ACK ή ένα timeout, όπως στο Reno. Ως εκ τούτου, το Vegas λύνει το πρόβλημα των χαμένων πακέτων όταν το παράθυρο είναι πολύ μικρό δηλ. λιγότερο από τρία και δεν μπορεί να λάβει αρκετά διπλά ACKs. Η συμπεριφορά αποφυγής συμφόρησης TCP Vegas είναι διαφορετική από άλλες εφαρμογές TCP. Καθορίζει τα στάδια συμφόρησης χρησιμοποιώντας το ποσοστό αποστολής. Εάν υπάρχει μια μείωση στο υπολογισμένο ποσοστό μετάδοσης ως αποτέλεσμα της μεγάλης σειράς αναμονής στη σύνδεση, μειώνεται το μέγεθος του παραθύρου. Όταν το ποσοστό αποστολής αυξάνεται, αυξάνεται επίσης και το μέγεθος του παραθύρου.

3.2.6 Delay Based αλγόριθμοι αποφυγής συμφόρησης

Χρησιμοποιούν την καθυστέρηση αναμονής για να επισημάνουν την ανάγκη για τη ρύθμιση παραθύρου, όπου μπορεί να λύσει το πρόβλημα της δικαιοσύνης. Για να εφαρμοστεί ο delay based έλεγχος είναι απαραίτητο να μετρηθεί η καθυστέρηση μετάδοσης. Η καθυστέρηση διάδοσης είναι συνήθως το μικρότερο παρατηρηθέν RTT. Πειραματικά πρωτόκολλα που προσπαθούν να βελτιώσουν την απόδοση του TCP κάτω από μεγάλο BDP, καταρτίζουν τον κανόνα αυξήσεων TCP να γίνει επιθετικότερος. Το Loss- delay based χρησιμοποιήθηκε από το TCP Africa, TCP Illinois, TCP Compound. Αυτά τα πρωτόκολλα προσπαθούν να αυξήσουν το μέγεθος του παραθύρου επιθετικότερα από το TCP new Reno εφόσον δε χρησιμοποιείται πλήρως από το δίκτυο και αυτό μεταβαίνει στη συμπεριφορά AIMD Reno όταν είναι κοντά στη συμφόρηση.

3.2.7 Σύνοψη της προτεινόμενης εφαρμογής ελέγχου συμφόρησης TCP

Το TCP Tahoe, Reno και New Reno αναφέρονται ως New-Reno και υλοποιείται σε πάνω από το 90 % της κυκλοφορίας διαδικτύου, Έγινε επίσημα αναγνωρισμένο το 2004, το compound TCP αναμενόταν να αυξηθεί κατά τη χρήση του σε ολόκληρο το διαδίκτυο λόγω της εφαρμογής του στα MS-Windows 7.

3.3 Rate-Based σύστημα ελέγχου συμφόρησης

Μεγάλο ενδιαφέρον επικεντρώνεται στη μέγιστη χρήση του δικτύου [2]. Το optimization-based συσχετίζει μια συνάρτηση χρησιμότητας με κάθε ροή και μεγιστοποιεί τη συνολική χρησιμότητα του συστήματος λειτουργίας υποκείμενη σε περιορισμούς χωρητικότητας συνδέσμου. Αυτό είναι ένα πρόβλημα βελτιστοποίησης (Kelly), βασισμένο στο ποσοστό ελέγχου συμφόρησης, τα σχέδια ελέγχου συμφόρησης μπορούν να αντιμετωπισθούν ως αλγόριθμοι που υπολογίζουν τις βέλτιστες ή υπό-βέλτιστες λύσεις στο πρόβλημα βελτιστοποίησης συστημάτων του Kelly. Τα σχέδια ελέγχου συμφόρησης μπορούν να ταξινομηθούν σε τρεις αλγορίθμους: Primal, Dual και Primal-Dual.

3.4 Μαθηματική μοντελοποίηση ελέγχου συμφόρησης του διαδικτύου

Μια σημαντική κατεύθυνση είναι η αναζήτηση για νέα μοντέλα για την αντικατάσταση του αλγορίθμου AIMD του Jacobson όπου έχει λειτουργήσει πολύ καλά στο διαδίκτυο. Ταξινομήθηκαν τα πρότυπα για τον έλεγχο συμφόρησης και τα σχετικά εργαλεία όπως τα παρακάτω:

- Packet level models: Ένα πρότυπο επιπέδων πακέτων αποτελεί τη θέση κάθε μεμονωμένου πακέτου όπως τα πακέτα περιμένουν στη σειρά και διαβιβάζονται από το δίκτυο. Η κατάσταση συστημάτων εξελίσσεται ως σειρά ιδιαίτερων γεγονότων. Τα γεγονότα στο διαδίκτυο είναι η άφιξη, η αναχώρηση των πακέτων και τα διαλείμματα.

- Fluid flow models: Ένα ρευστό πρότυπο ροής βλέπει τα στοιχεία, τα μεταφέρει ως συνεχές ρευστό, χωρίς τα όρια πακέτων. Οι μεταβλητές καταστάσεις ποικίλλουν συνεχώς, και περιγράφονται χρησιμοποιώντας διαφορικές εξισώσεις. Οι μεταβλητές καταστάσεις αντιπροσωπεύουν το μέσο όρο των αληθινών συστημάτων.

- Hybrid Models: Εδώ, η εξέλιξη της κατάστασης είναι ένα αποτέλεσμα των ιδιαίτερων γεγονότων μαζί με τις συνεχείς αλλαγές μεταξύ των γεγονότων. Ένα συνεχές πρότυπο χρησιμοποιείται για τις ενέργειες οικοδεσποτών δυναμικής και τελών αναμονής, ενώ η δράση όπως στην πολλαπλασιαστική μείωση του TCP διαμορφώνεται ως ιδιαίτερο γεγονός.

Ο αλγόριθμος ελέγχου συμφόρησης του Jacobson λειτουργεί σε δύο φάσεις: slow start και congestion avoidance.

3.5 Φάση Slow Start

Αρχικά (a) το μέγεθος του παραθύρου είναι ίσο με 1, και (b) αυξάνεται κατά 1 για κάθε λαμβανόμενο ack. Επαναλαμβάνεται (c) μέχρι το ssthresh να επιτευχθεί ή να ανιχνευτεί απώλεια πακέτων, εάν το ssthresh επιτευχθεί πηγαίνει στη φάση αποφυγής συμφόρησης. Τέλος (d) εάν ανιχνευθεί απώλεια πακέτων πηγαίνει στο βήμα 2.

Ssthresh= Slow Start Threshold

a) cwnd = 1;

b) while (ssthresh ≠ cwnd OR not 3 DUPACK) {

if ACK then

cwnd=cwnd + 1;
}

c) if ssthresh == cwnd

d) if 3DUPACK then

ssthresh = 0.5 * cwnd;

cwnd = 1;

Θέτει την αρχική τιμή του ssthresh στο μισό του μέγιστου μεγέθους παραθύρου. Αυτό καθορίζεται στην αρχή της μετάδοσης.

3.5.1 Φάση Congestion Avoidance

Αυξάνει το μέγεθος του παραθύρου κατά 1 για κάθε ACK που λαμβάνει. Αυτό σημαίνει ότι το μέγεθος του παραθύρου αυξάνεται κατά 1 μετά από όλα τα ACKs που έχουν ληφθεί για αυτό το παράθυρο.
Όταν ανιχνεύεται απώλεια πακέτων, μειώνει το μέγεθος παραθύρου, κατόπιν μεταβαίνει σε φάση slow start. Η απώλεια πακέτων στο TCP New Reno ανιχνεύεται μέσω :

- Απώλειας 3 διαδοχικών πακέτων, ή
- Από ένα RTO

3.5.2 Τροποποιήσεις Slow-Start

Υπήρξαν αρκετές τροποποιήσεις στο στάδιο Slow-Start για να ξεπεραστεί το πρόβλημα της απόδοσης. Μια ομάδα ερευνητών χρησιμοποίησε τεχνικές εκτίμησης για την εκτίμηση χωρητικότητας διαθέσιμου bandwidth και ρύθμισε το μέγεθος του παραθύρου συμφόρησης χρησιμοποιώντας τον εκτιμώμενο bandwidth.

- Swift Start
- Restricted slow-start
- Shared Passive Network Performance Discovery (SPAND)
- Adaptive Start προσαρμοστική έναρξη (Ren Wang)
- Capstart (Canvendish)

Η δεύτερη ομάδα χρησιμοποίησε νέους parameter-config βασισμένους στο slow start για να καθορίσει την ταχύτητα μετάδοσης.

- TCP Fast Start
- TCP Vegas
- New Parameter-Config Based Slow Start Mechanism (P-Start)

Η τρίτη ομάδα λάμβανε πληροφορίες ή ζητούσε βοήθεια από το δίκτυο. Σύλλεγε πληροφορίες για την κατάσταση της συμφόρησης, ανατροφοδοτούνταν από τους δέκτες και το μοίραζε με τελικά σημεία με συνδέσεις στο δίκτυο. Αυτό επέτρεψε σε συνδέσεις να καθορίζουν την κατάσταση συμφόρησης του δικτύου και έτσι να καθορίζει και ένα αρχικό ρυθμό αποστολής.

- Quick Start

Το E-speed start συνδυάζει το χαρακτηριστικό των δύο ομάδων (παράμετρο με βάση τις πράξεις παραποίησης της χωρητικότητας / εκτίμηση του bandwidth).

3.6 Προβλήματα που σχετίζονται με τη μεμονωμένη TCP παραλλαγή

Ο μηχανισμός ελέγχου του TCP Tahoe έχει ένα πρόβλημα, εάν ένα πακέτο χαθεί, ο αποστολέας μπορεί να χρειαστεί να περιμένει μεγάλο διάστημα για να συμβεί ένα timeout για την απώλεια αυτή να ανιχνευθεί και να αναμεταδοθεί. Το TCP Reno σχεδιάστηκε για να λύσει αυτό το πρόβλημα.

Ένα άλλο πρόβλημα είναι ότι στο Reno αν το παράθυρο είναι πολύ μικρό, όταν συμβαίνει η απώλεια, ο αποστολέας δεν μπορεί να λάβει αρκετά διπλά ACKs για μια γρήγορη αναμετάδοση. Ένα πρόβλημα με τον αλγόριθμο New-Reno είναι η ανικανότητά του να ανιχνεύει οποιοδήποτε άλλο packet-loss στο παράθυρο. Αυτό το πρόβλημα προτείνεται να λύσει το SACK. Οι Delay-based αλγόριθμοι ελέγχου συμφόρησης προτάθηκαν για την επίλυση των προβλημάτων που σχετίζονται με τα πρότυπα ελέγχου συμφόρησης loss based που έχουν προβλήματα στη δικαιοσύνη, όταν μοιράζονται τη σύνδεση με το TCP New Reno. Το BIC TCP και το CUBIC προσπαθούν να λύσουν αυτά τα προβλήματα.

Κεφάλαιο 4 TCP-Africa: Ένας προσαρμοστικός και δίκαιος γρήγορος κανόνας αύξησης για το Scalable TCP

4.1 Εισαγωγή

Το TCP είναι το καλύτερο δυνατό για τον έλεγχο μεταφοράς δεδομένων, όμως από 100 Mbps και πάνω παρουσιάζει πρόβλημα σε μεγάλες αποστάσεις και δεν είναι αρκετά γρήγορο στην αποφυγή συμφόρησης. Οι πρώτες προσπάθειες για την αντιμετώπιση αυτού του προβλήματος εξοπλίζοντας το TCP με αυξημένη επιθετικότητα, έχουν δείξει το μειονέκτημα της φτωχής δικαιοσύνης με το πρότυπο TCP Reno. Έτσι προτάθηκε το TCP Africa το οποίο ήταν μια νέα λειτουργία ευαίσθητης καθυστέρησης, χαρακτηρίστηκε ασφαλές, αποτελεσματικό και "αμερόληπτο".

Το TCP Reno προτάθηκε το 1988 και είχε μεγάλη επιτυχία στο διαδίκτυο σε γενικές γραμμές, μέχρι που άρχισε να φτάνει στα όρια του στο σύγχρονο internet, με το αυξανόμενο bandwidth. Όταν προτάθηκε, οι συνδέσεις ήταν της τάξεως των 56 Kbps. Έχει περιορισμούς σε δίκτυα με μεγάλο bandwidth-delay-product, χρησιμοποιεί το επιθυμητό παράθυρο συμφόρησης που είναι περίπου ίσο με το bandwidth-delay-product της σύνδεσης και έχει ικανοποιητική ανάκτηση παραθύρου μετά από μια περίπτωση συμφόρησης. Σύμφωνα με το μοντέλο τυχαίας απώλειας πακέτων, το TCP-Reno μπορεί να απαιτεί παράλογα low packet drop για αυτούς τους συνδέσμους υψηλού bandwidth-delay-product. Διορθώνοντας τις ελλείψεις του TCP με καθαρό loss based πρωτόκολλο οδηγεί σε πολύ επιθετικά πρωτόκολλα τα οποία προκαλούν τις ανησυχίες δικαιοσύνης και ασφάλειας. Αντιθέτως τα delay based πρωτόκολλα κάνουν καλή χρήση πληροφοριών σχετικά με την κατάσταση του δικτύου που παρέχεται από καθυστερήσεις πακέτων, επιτυγχάνοντας σταθερή κατάσταση απόδοσης και ελάχιστο packet loss.

Τα delay sensitive πρωτόκολλα σε σχέση με τα non-responsive είναι σε μειονεκτική θέση στο χρόνο αντίδρασης. Τα non-responsive δεν προκαλούν αρκετά packet drops στα non-delay-responsive πρωτόκολλα για να τα αναγκάσουν να διατηρήσουν μόνο το δικό τους μερίδιο. Το TCP-Africa προτάθηκε ως μια νέα delay sensitive διπλής λειτουργίας αποφυγής συμφόρησης για το TCP που υπόσχεται άριστη αξιοποίηση, αποτελεσματικότητα, και απόκτηση του διαθέσιμου εύρους ζώνης, βελτίωση της ασφάλειας, της δικαιοσύνης και RTT ιδιότητες πόλωσης. Χρησιμοποιεί έναν επιθετικό, εξελικτικό κανόνα αύξησης παραθύρου που επιτρέπει τη γρήγορη αξιοποίηση του διαθέσιμου bandwidth, επίσης προβλέπει επιφανείς γεγονότα συμφόρησης.

4.2 Τα υπάρχοντα πρωτόκολλα/Εκτιμήσεις για ένα πρωτόκολλο υψηλής ταχύτητας

Έχουν προταθεί νέα υψηλής ταχύτητας πρωτόκολλα end-to-end και χωρίζονται σε δύο ομάδες:

- Purely loss based αλγόριθμοι, όπως STCP, HSTCP, και BIC-TCP
- Delay based αλγόριθμοι, όπως το FAST TCP.

Κατά το σχεδιασμό ενός high speed πρωτοκόλλου τροποποιήθηκε η αύξηση και η μείωση των παραμέτρων του TCP, αλλά και άλλα ζητήματα τα οποία είναι τα ακόλουθα :

- Throughput
- Peer fairness
- TCP fairness
- Congestion collapse

4.2.1 Επιθετικά loss based πρωτόκολλα

Το κύριο μειονέκτημα του TCP-Reno σε μεγάλο DBP εντοπίζεται στο γραμμικό τρόπο αύξησης αποφυγής συμφόρησης. Αντιδρά στα packet drops ως εξής:

- Μειώνει το παραθύρου συμφόρησης κατά το ήμισυ
- Αυξάνει το παράθυρο συμφόρησης κατά ένα πακέτο ανά RTT

Ένα πρωτόκολλο MIMD (Multiplicative Increase Multiplicative Decrease), όπως το STCP το οποίο είναι το επιθετικότερο από τις ήδη υπάρχουσες προτάσεις του TCP χρησιμοποιεί κανόνες προσαρμογής πολλαπλασιαστικής αύξησης και πολλαπλασιαστικής μείωσης παραθύρου. Για κάθε πακέτο που ελήφθη αυξάνει το παράθυρο συμφόρησης κατά 0,01 πακέτα, και σε ένα packet drop, το παράθυρο μειώνεται 0,875 φορές στο τρέχον παράθυρο. Το MIMD είναι ισοδύναμο με ένα πρόσθετο πρόγραμμα αύξησης όπου η αύξηση μεγέθους του βήματος αυξάνεται αναλογικά με το μέγεθος του παραθύρου συμφόρησης.

Στην παραλλαγή HSTCP (High Speed TCP) ο ρυθμός αύξησης μεγαλώνει λίγο πιο αργά από αυτό του STCP (Scalable TCP), αλλά και πάλι είναι πολύ γρήγορα σε σύγκριση με το TCP Reno. Ένα επιθετικό TCP οδηγεί πάντα σε μείωση της δικαιοσύνης.
Το HSTCP και το STCP έχουν ανεπιθύμητες ιδιότητες της δικαιοσύνης όταν οι ροές με διαφορετικά round trip times ανταγωνίζονται για ένα κοινό σύνδεσμο.
Το BIC-TCP έχει ένα επιθυμητό RTT, προσεγγίζει γρήγορα ένα αναμενόμενο ασφαλή παράθυρο και αυξάνει σταδιακά πάνω από αυτό το παράθυρο.

Για την αντιμετώπιση του κινδύνου κατάρρευσης συμφόρησης, τα loss based πρωτόκολλα αυξάνουν το παράθυρο συμφόρησης περισσότερο από ένα πακέτο ανά RTT. Ο επιθετικός ρυθμός αύξησης μπορεί να προκαλέσει αύξηση της επιβάρυνσης στους ενδιάμεσους απομονωτές δρομολογητών. Ο αριθμός των πακέτων που χάνονται ανά συμβάν συμφόρησης τείνει να σχετίζεται με τον αριθμό των επιπλέον πακέτων που εισήλθαν στο δίκτυο στο τελευταίο RTT. Παρόλα αυτά παραμένει ένας επιθυμητός στόχος για την αποφυγή τέτοιων προβλημάτων. Λόγω της επιθετικότερης συμπεριφοράς τους, τα προαναφερθέντα high-speed loss-based πρωτόκολλα προκάλεσαν γεγονότα συμφόρησης σε μια πολύ υψηλότερη συχνότητα από ότι προκαλείται από το TCP-Reno.

4.3 Προκληθείσα συχνότητα γεγονότος συμφόρησης

Εξετάστηκε η συχνότητα εκδήλωσης συμφόρησης που προκλήθηκε μόνη της για ένα πρωτόκολλο AIMD (Additive Increase Multiplicative Decrease), όπως το TCP-Reno. Για ένα γενικό πρωτόκολλο AIMD, η επαγόμενη περίοδο εκδήλωσης συμφόρησης είναι :

$$\mathbf{T} = \frac{(1-b)\,Wmax}{\alpha}$$

a (αύξηση μεγέθους βημάτων)
b (ο πολλαπλασιαστικός συντελεστής μείωσης)

Για το Reno, a = 1 και b = 0,5. Έτσι, βρίσκουμε, για το TCP Reno, ότι η επαγόμενη περίοδο εκδήλωσης συμφόρησης είναι :

$$\mathrm{T} = \frac{Wmax}{2}$$

Τα MIMD πρωτόκολλα, τα οποία χρησιμοποιούν μια πολλαπλασιαστική παράμετρο αύξησης **a** και μια πολλαπλασιαστική παράμετρο μείωσης **b**, έχουν μια σταθερή κατάσταση περιόδου συμφόρησης.
Το HSTCP και STCP έχουν ταχεία επιστροφή στο μέγιστο παράθυρο. Ενώ είναι καλό για τη συμφόρηση, στην πραγματικότητα φτάνοντας σε πλήρη ικανότητα γρήγορα θα προκληθεί η επόμενη εκδήλωση συμφόρησης, [3].

4.3.1 TCP-Vegas-like πρωτόκολλα

Μια άλλη σημαντική ομάδα πρωτοκόλλων υψηλής ταχύτητας είναι τα πρωτόκολλα Vegas-like delay based πρωτόκολλα, όπως το FAST TCP. Τα delay based πρωτόκολλα έχουν πολλές επιθυμητές ιδιότητες:

- Είναι απίθανο να προκαλέσουν σημαντική αναμονή καθυστέρησης
- Μπορεί να συγκλίνουν γρήγορα σε ισορροπία
- Μπορούν να τρέχουν σε σταθερή κατάσταση, χωρίς να προκαλούνται packet drops

Το μειονέκτημά τους είναι πως δεν είναι σε θέση να ανταγωνιστούν πρωτόκολλα με απληστία όπως το Reno σε ζεύξεις με συμφόρηση.
Τα delay based πρωτόκολλα προσπαθούν να μετρήσουν το διαθέσιμο bandwidth ενός συνδέσμου και αφήνουν ένα Ν τοις εκατό του εύρους ζώνης ελεύθερο. Τα Vegas-like πρωτόκολλα τείνουν να αποφεύγουν την πρόκληση packet drops, και ως εκ τούτου, δεν είναι σε θέση να προκαλούν άπληστες ροές για να κάνουν πίσω. Οποιοδήποτε πρωτόκολλο που μειώνει το παράθυρο συμφόρησης και βασίζεται στις πληροφορίες καθυστέρησης είναι θεωρητικά ευάλωτο σε αυτό το πρόβλημα, όπως και το FAST TCP.

4.3.2 Ζητήματα σχεδιασμού

Το HSTCP και STCP είναι πιο επιθετικά μόνο κατά τη στιγμή που στέλνουν τη μέγιστη χωρητικότητα, σε αντίθεση με τα delay based πρωτόκολλα που είναι λιγότερο επιθετικά κατά την αποστολή. Η σταθερή κατάσταση οφείλεται στο γεγονός ότι από τη στιγμή που η ουρά αρχίζει να γεμίζει, η καθυστέρηση θα αυξηθεί παρέχοντας στο πρωτόκολλο τις πρώτες πληροφορίες.

Το TCP Africa είναι ένας προσαρμοστικός και δίκαιος ταχείας αύξησης μηχανισμός αποφυγής συμφόρησης για το TCP. Σε περιπτώσεις απουσίας της συμφόρησης, η γρήγορη λειτουργία χρησιμοποιεί ένα επιθετικά επεκτάσιμο κανόνα αποφυγής συμφόρησης και σε περιπτώσεις παρουσίας της συμφόρησης, η αργή λειτουργία, μεταβαίνει στο πιο συντηρητικό κανόνα αποφυγής συμφόρησης (Reno). Υπό ευνοϊκές συνθήκες καθυστέρησης με την προϋπόθεση πως υπάρχει ελεύθερο εύρος ζώνης το πρωτόκολλο αυξάνει το παράθυρο με ένα επιθετικό επεκτάσιμο τρόπο. Το TCP-Africa, σε "fast" λειτουργία (τεχνική πολλαπλασιαστικής αύξησης), κατοχυρώνει γρήγορα διαθέσιμο εύρος ζώνης. Μόλις αρχίσει η καθυστέρηση να αυξάνεται και το bandwidth μειωθεί, τότε μεταβαίνει σε ένα συντηρητικό γραμμικό τρόπο αύξησης, με ρυθμό ενός πακέτου ανά RTT.

4.4 Περιγραφή του TCP-Africa

Μια εκθετικά εξομαλυμένη υψηλής ακρίβειας εκτίμηση round trip time, **α** Rtt διατηρείται από τη ροή. Αυτή η καθυστέρηση πληροφορίας, μαζί με την ελάχιστη καθυστέρηση παρατηρήθηκε στη διαδρομή **min**RTT, που χρησιμοποιείται για να εκτιμηθεί η καθυστέρηση αναμονής στο σύνδεσμο. Το W δηλώνει το τρέχον παράθυρο συμφόρησης της ροής, η παράμετρος **α** είναι μια σταθερά η οποία ορίζεται ως πραγματικός αριθμός μεγαλύτερος από ένα και καθορίζει το πόσο ευαίσθητο είναι το πρωτόκολλο στη καθυστέρηση. Το TCP Africa χρησιμοποιεί το ακόλουθο για την ανίχνευση συμφόρησης.

$$\frac{W(aRTT - minRTT)}{aRTT} \gtrless \alpha.$$

Η ποσότητα (aRTT − minRTT) μας δίνει μια εκτίμηση της καθυστέρησης αναμονής του δικτύου. Δεδομένου ότι το συνολικό round trip time είναι minRTT + (aRTT − minRTT), η ποσότητα (aRTT − minRTT) / aRTT είναι η αναλογία του round trip time που οφείλεται σε καθυστέρηση αναμονής, μάλλον από καθυστέρηση διάδοσης. Δεδομένου ότι το πρωτόκολλο TCP διατηρεί ένα μέσο ρυθμό αποστολής W/aRTT πακέτων ανά δευτερόλεπτο, το W(aRTT − minRTT)/aRTT είναι μια εκτίμηση του αριθμού των πακέτων που το πρωτόκολλο έχει επί του παρόντος στην ουρά.

Τα βήματα αποφυγή συμφόρησης που ακολουθείται από το πρωτόκολλο είναι τα εξής:

if (W (aRTT − minRTT) < × aRTT) {

W = W + fast increase(W)/W

} else {

W = W + 1/W

}

Η γρήγορη αύξηση λειτουργίας (W) ορίζεται από ένα σύνολο τροποποιημένων κανόνων αύξησης για το TCP [3]. Αυτή τη στιγμή, χρησιμοποιεί τους κανόνες αποφυγής συμφόρησης που καθορίζονται από HSTCP για να καθορίζει την καμπύλη αύξησης, όταν πληρεί την προϋπόθεση καθυστέρησης.

4.4.1 Safety (Ασφάλεια) και Fairness (δικαιοσύνη)

Από πειράματα που έγιναν για την παρατήρηση της ασφάλειας και της δικαιοσύνης προέκυψαν τα ακόλουθα αποτελέσματα.

Ασφάλεια:

- Το HTCP παρουσίαζε συχνά γεγονότα συμφόρησης που προκλήθηκαν από τη ροή του HTCP παρεμποδίζοντας την απόδοση της ροής του TCP- Reno.

- Στο TCP Africa το TCP-Reno ανακουφίζει αποτελεσματικά τη ροή στο σύνδεσμο πρόσβασης. Η αλλαγή του TCP Africa για να επιβραδύνει τη λειτουργία της ανάπτυξης είναι σαφώς ορατή.

Δικαιοσύνη:

- Η δικαιοσύνη του Reno στο πείραμα για το TCP- Africa. Το TCP -Africa και TCP - Reno ανακουφίζουν τον κοινόχρηστο σύνδεσμο. Η αλλαγή του TCP -Africa για να επιβραδύνει τη λειτουργία της ανάπτυξης είναι εμφανείς σε κάθε κύκλο περίπτωσης συμφόρησης.

- Η δικαιοσύνη του Reno στο πείραμα για το HTCP. Το TCP - Reno είναι ανίκανο να επιτύχει το σημαντικό μέρος του bandwidth της σύνδεσης.

4.4.2 Προσαρμογή στις συνθήκες του δικτύου

Το TCP- Africa μειώνει γρήγορα το bandwidth σε απάντηση προς τη ροή του UDP. Εισάγεται ο αργός τρόπος αύξησης του πρωτοκόλλου καθώς η ροή πλησιάζει το μέγιστο διαθέσιμο bandwidth.

***Ανθεκτικότητα*:** Ένα άλλο πλεονέκτημα αυτής της τεχνικής αύξησης παραθύρου είναι πως είναι απολύτως ανθεκτικό σε δυσλειτουργία του delay metric. Σε ένα δίκτυο όπου η καθυστέρηση δεν αυξάνει, το πρωτόκολλο θα παραμείνει στη Fast λειτουργία, ενεργώντας σαν ένα κανονικό STCP ή HSTCP αποστολέα.

***Ασφάλεια*:** Ένας τέτοιος προσαρμοστικός αλγόριθμος αύξησης είναι μια συντηρητική επιλογή για τον κανόνα αύξησης και επιτρέπει στο πρωτόκολλο να τρέξει σε υψηλές ταχύτητες ενώ εξακολουθεί να είναι ασφαλής για το διαδίκτυο συνολικά.

Κεφάλαιο 5 Ανάλυση απόδοσης αποφυγής συμφόρησης του TCP

5.1 Εισαγωγή

Η απαίτηση για γρήγορη μεταφορά μεγάλου όγκου δεδομένων και η επέκταση των υποδομών του δικτύου είναι πάντα αυξανόμενη, ωστόσο το TCP δεν ικανοποιεί αυτή την απαίτηση.

Η αργή απόκριση του TCP σε μεγάλης απόστασης δίκτυα αφήνει αχρησιμοποίητο ένα μεγάλο εύρος ζώνης. Παραλλαγές του ελέγχου συμφόρησης του TCP είναι:

- Loss based, high-speed TCP αλγόριθμοι συμφόρησης που χρησιμοποιούν απώλειες πακέτων ως ένδειξη της συμφόρησης.
- Delay-based TCP που δίνει έμφαση στην καθυστέρηση των πακέτων ως ένα σήμα για να καθορίσει το ρυθμό με τον οποίο θα στείλει τα πακέτα.

Έχουν γίνει προσπάθειες για το συνδυασμό και των δύο αλγορίθμων για την επίτευξη δίκαιης κατανομής bandwidth και της δικαιοσύνης μεταξύ των ροών. Το TCP είναι ένα καλά ανεπτυγμένο πρωτόκολλο μεταφοράς, είναι γρήγορο, αποτελεσματικό και ευαίσθητο στις συμφορήσεις του δικτύου. Ένα αρνητικό του ελέγχου συμφόρησης του TCP είναι το back-off AIMD όπου είναι πολύ απότομο στη μείωση μεγέθους του παραθύρου και βλάπτει το ρυθμό μετάδοσης δεδομένων.
Ο Tomoya Hatano πρότεινε ένα TCP friendly για τον έλεγχο συμφόρησης που είναι αποδοτικό στη μετάδοση δεδομένων στα high speed δίκτυα, δίκαιο με το TCP Reno και κάνει δίκαιη κατανομή bandwidth μεταξύ των ροών με διαφορετικά RTT.

5.2 Μεγάλης ταχύτητας TCP Παραλλαγές

Το TCP αποδίδει πολύ καλά σε χαμηλή έως μέση ταχύτητα στα δίκτυα (Kbps έως αρκετά Mbps), αντιθέτως σε δίκτυα υψηλής ταχύτητας έχει πολύ κακή απόδοση.

5.2.1 HighSpeed TCP

Το HSTCP είναι ένας τροποποιημένος μηχανισμός ελέγχου συμφόρησης του TCP για την χρήση TCP συνδέσεων με μεγάλα παράθυρα συμφόρησης. Είναι περισσότερο επιθετικό από το TCP Reno και λαμβάνει 10 φορές το bandwidth από το standard TCP σε ένα περιβάλλον με packet drop rate 10^{-6}, το οποίο είναι άδικο.

5.2.2 Scalable TCP

Το Scalable TCP έχει σχεδιαστεί για να είναι σταδιακά αναπτυσσόμενο και να συμπεριφέρεται με τον ίδιο τρόπο με τις παραδοσιακές TCP στοίβες όταν τα μικρά παράθυρα είναι επαρκείς. Αρχικά είχε σχεδιαστεί για backbone συνδέσεις υψηλής ταχύτητας, και φαινόταν να είναι ο σημαντικότερος υποψήφιος για την αντικατάσταση στην επόμενη γενιά διαδικτύου.

Το STCP είναι μια τροποποίηση από τη πλευρά του αποστολέα για τον έλεγχο συμφόρησης στο TCP, και υιοθετεί την τεχνική MIMD. Κάνει καλύτερη αξιοποίηση μιας σύνδεσης δικτύου με μεγάλο bandwidth delay product, επίσης όταν αναμιγνύεται με το πρότυπο TCP, εξουσιάζει το bandwidth του δίκτυο για αρκετά μεγάλο bandwidth delay product. Στην περίπτωση που αναμιγνύεται με το standard TCP, αυτό παρουσιάζει εχθρότητα προς το standard TCP.

Μια σύνδεση STCP μπορεί να ανακτήσει ακόμα και ένα μεγάλο παράθυρο συμφόρησης σε σύντομο χρονικό διάστημα και έτσι να κάνει αποτελεσματική χρήση του bandwidth σε υψηλής ταχύτητας δίκτυα.

5.2.3 CUBIC TCP

Είναι μια βελτιωμένη έκδοση του δυαδικού BIC για τον έλεγχο αυξημένης συμφόρησης. Αυτό απλοποιεί τη λειτουργία του BIC για τον έλεγχο του παραθύρου και βελτιώνει τη φιλικότητα του TCP και την αμεροληψία του RTT όπως το BIC, η λειτουργία αύξησης είναι πάρα πολύ επιθετική για το TCP ειδικά κάτω από σύντομο RTT ή σε αργόστροφα δίκτυα.

Το πρωτόκολλο αντιπροσωπεύει, τη λειτουργία ανάπτυξης παραθύρου του CUBIC, όσον αφορά το χρόνο που έχει παρέλθει από την τελευταία περίπτωση απώλειας, είναι πολύ παρόμοιο με τη λειτουργία αύξησης του BIC. Η CUBIC λειτουργία παρέχει καλή επεκτασιμότητα και σταθερότητα. Το πρωτόκολλο κρατά τον ρυθμό ανάπτυξης παραθύρου ανεξάρτητο από το RTT, το οποίο διατηρεί το TCP πρωτόκολλο φιλικό κάτω από βραχυπρόθεσμα και μακροπρόθεσμα RTTs.

5.3 Delay-Based έλεγχος συμφόρησης

Αλγόριθμοι ελέγχου συμφόρησης Delay-based TCP όπως το TCP Vegas προσπαθούν να χρησιμοποιήσουν τις πληροφορίες συμφόρησης που περιέχονται στα πακέτα RTT.

5.3.1 TCP Vegas (Delay-based TCP)

TCP Vegas [4] είναι ένας αλγόριθμος ελέγχου συμφόρησης TCP που τονίζει το packet delay, παρά το packet loss, σαν σήμα για να καθορίσει το ρυθμό με τον οποίο στέλνει τα πακέτα. Το TCP Vegas ανιχνεύει την συμφόρηση βασιζόμενο στην αύξηση του RTT, οι τιμές των πακέτων στη σύνδεση σε αντίθεση με το TCP Reno ανιχνεύουν τη συμφόρηση μόνον αφού έχει πραγματοποιηθεί από packet drops. Ο αλγόριθμος εξαρτάται σε μεγάλο βαθμό από τον ακριβή υπολογισμό της τιμής της Base RTT. Η Base RTT έχει οριστεί να είναι η ελάχιστη όλων των RTTs. Είναι το πρώτο σήμα που στέλνεται από τη σύνδεση.

Εάν η σύνδεση δεν υπερχειλίσει, η αναμενόμενη ταχύτητα μετάδοσης δίνεται από :

$$\text{Expected Throughput} = \text{WindowSize} / \text{BaseRTT}$$

όπου το μέγεθος του παραθύρου είναι το μέγεθος από το τρέχον παράθυρο συμφόρησης.

Στη συνέχεια η τρέχουσα πραγματική ταχύτητα αποστολής υπολογίζεται μία φορά ανά round trip time ως:

$$\text{Actual throughput} = W / RTT$$

Το παράθυρο συμφόρησης ρυθμίζεται ανάλογα με την διαφορά μεταξύ των αναμενόμενων και των πραγματικών ποσοστών αποστολής.

$$\text{Διαφορά} = (\text{expected} - \text{actual}) \text{ baseRTT}$$

Επίσης δύο όρια a και b ορίζονται έτσι ώστε, a < b και a > b να αντιστοιχούν για να έχουν πάρα πολύ λίγο και πάρα πολύ επιπλέον κίνηση στο δίκτυο αντίστοιχα. Όταν η διαφορά < a, το TCP Vegas αυξάνει την συμφόρηση του παραθύρου γραμμικά κατά τη διάρκεια του επόμενου RTT, και όταν η διαφορά > b, το TCP Vegas μειώνει την συμφόρηση παραθύρου γραμμικά κατά τη διάρκεια του επόμενου RTT. Το παράθυρο συμφόρησης παραμένει αμετάβλητο όταν a < Διαφορά < b .

5.4 Έλεγχος συμφόρησης mixed loss-delay based TCP

Οι Loss-based υψηλής ταχύτητας αλγόριθμοι είναι επιθετικοί για να ικανοποιούν τις απαιτήσεις του bandwidth, αλλά αυτή η επιθετικότητα προκαλεί αδικίες στο RTT και TCP, οι Delay-based προσεγγίσεις παρέχουν στο RTT δικαιοσύνη αλλά όχι όπως το TCP. Έτσι υπάρχει το Compound TCP όπου είναι μια συνεργασία των delay-based και loss-based .

5.4.1 Compound TCP

Το Compound TCP ενσωματώνει ένα επεκτάσιμο delay-based συστατικό στο standard TCP αλγόριθμο αποφυγής συμφόρησης, όπου έχει λειτουργία γρήγορης αύξησης παραθύρου, μια περίπτωση συμφόρησης γίνεται αισθητή όταν το δίκτυο χρησιμοποιείτε και μειώνει τον ρυθμό αποστολής. Μεταβλητές κατάστασης Compound TCP :

- cwnd (παράθυρο συμφόρησης)
- dwnd (παράθυρο καθυστέρησης)
- awnd (διαφημιζόμενο παράθυρο δέκτη)

Το TCP παράθυρο αποστολής υπολογίζεται :

$$W = \min (cwnd+dwnd, awnd)$$

Το cwnd ενημερώνεται όπως το standard TCP. Κατά την άφιξη ενός ACK, το cwnd τροποποιείται ως εξής:

$$cwnd = cwnd + (1 / (cwnd + dwnd))$$

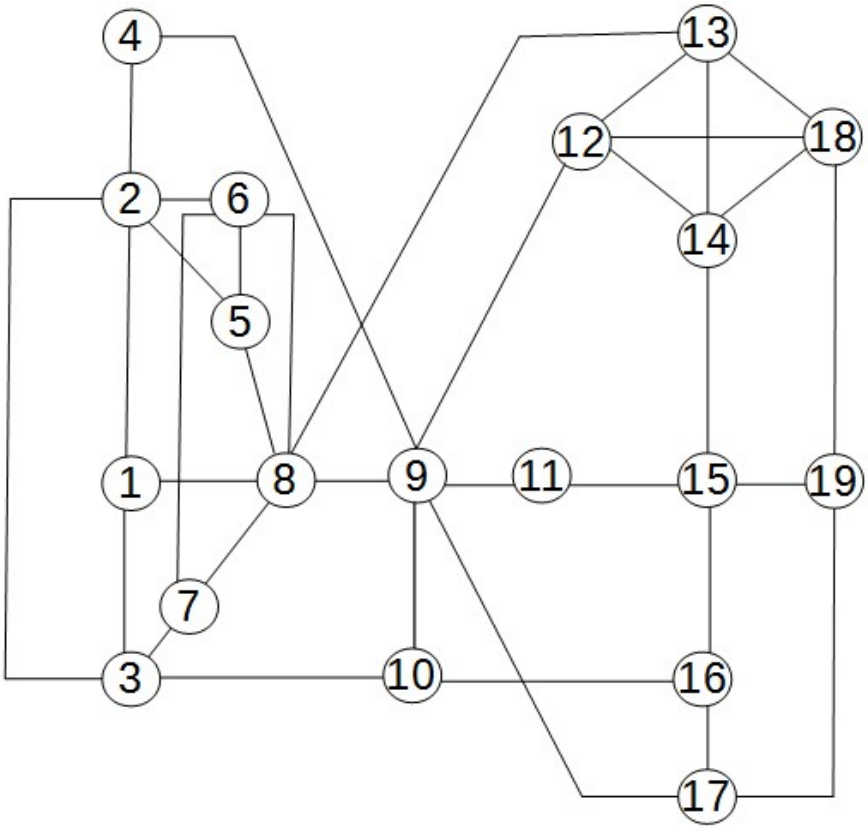
Κεφάλαιο 6 Προσομοίωση

6.1 Περιγραφή προσομοίωσης

Η παρακάτω προσομοίωση έγινε με σκοπό να παρατηρηθεί η απόδοση και η συμπεριφορά του παραθύρου συμφόρησης στο χρόνο. Χρησιμοποιήθηκε η έκδοση προσομοίωσης δικτύων 2 (NS-2), [6] για την προσομοίωση του TCP με τις προεπιλεγμένες παραμέτρους, αλλάζοντας μόνο το μέγεθος του παραθύρου. Ο τύπος κυκλοφορίας δεδομένων είναι FTP.

Στο σενάριο η τοπολογία είναι αυτή που χρησιμοποιείται στο [5], όλοι οι κόμβοι συνδέονται με 20Mbps και 10ms και τα ζεύγη πηγής/προορισμού περιγράφονται στο παρακάτω πίνακα.
Στα γραφήματα ο άξονας Y περιγράφει το παράθυρο συμφόρησης(*segments*) και ο άξονας X το χρόνο.

6.1.1 Η τοπολογία :



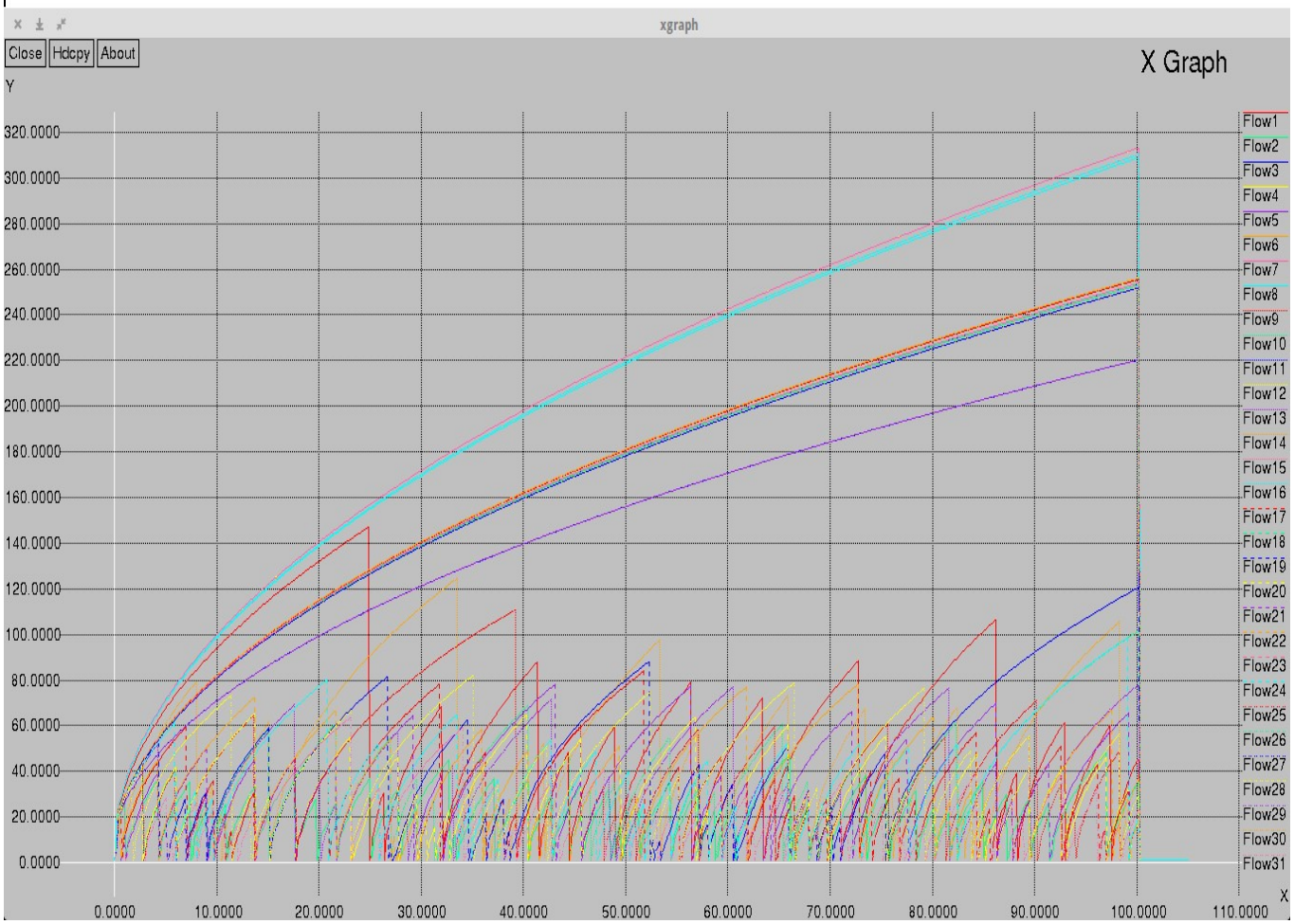
Σχ.1 Τοπολογία προσομοίωσης

Traffic Matrix 32 Ζεύγη Πηγή/Προορισμός	
From node	**To nodes**
n1	n9, n19
n2	n18
n3	n4, n11, n17
n4	n3, n10, n13
n5	n12
n6	n14
n7	n16
n8	n15
n9	n1, n19
n10	n4, n13
n11	n3, n10, n13
n12	n5
n13	n4, n10, n17
n14	n6
n15	n8
n16	n7, n18
n17	n3, n13
n18	n2, n16
n19	n1, n9

Πίνακας: ζεύγη πηγής/προορισμού

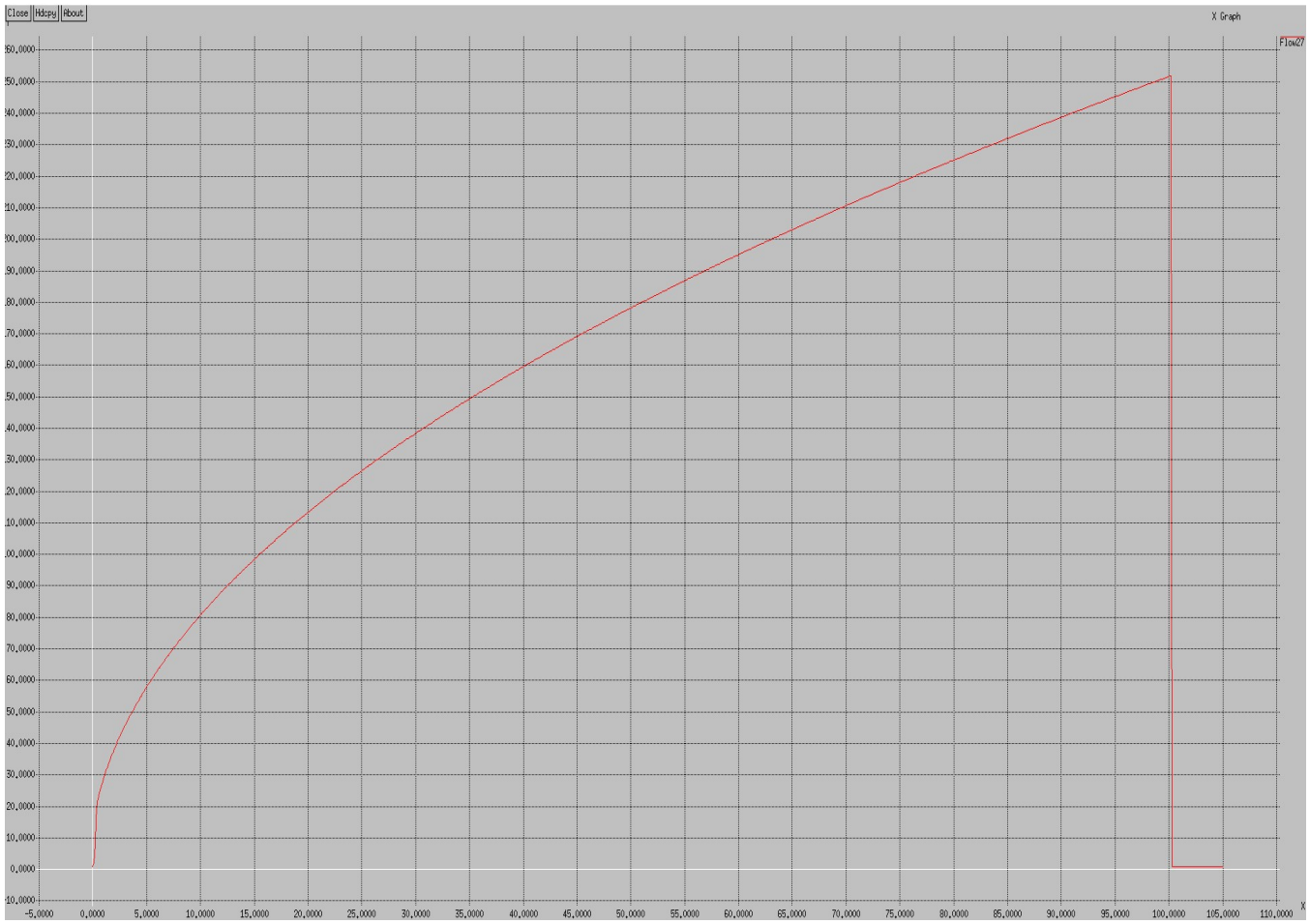
6.1.2 Υπόθεση α:

- Το μέγιστο όριο για το μέγεθος του παραθύρου έχει την προεπιλεγμένη τιμή 20.

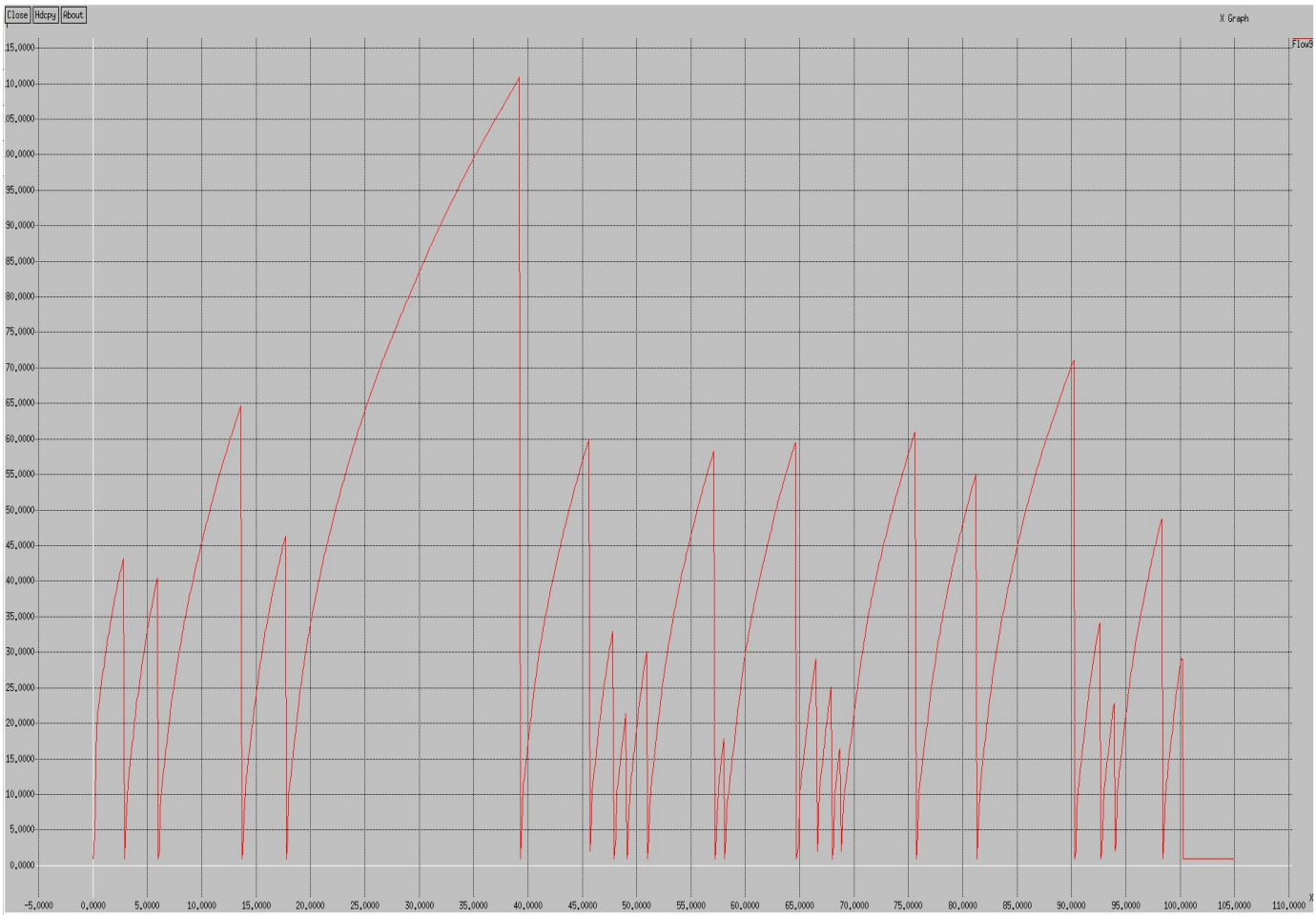


Σχ.2: Το παράθυρο συμφόρησης στο χρόνο, 32 flows

Παρατηρήθηκε το Flow27 στον κόμβο n17 και το Flow14 στον κόμβο n9.



Σχ.3:Το παράθυρο συμφόρησης στο χρόνο, Flow27 στον κόμβο n17



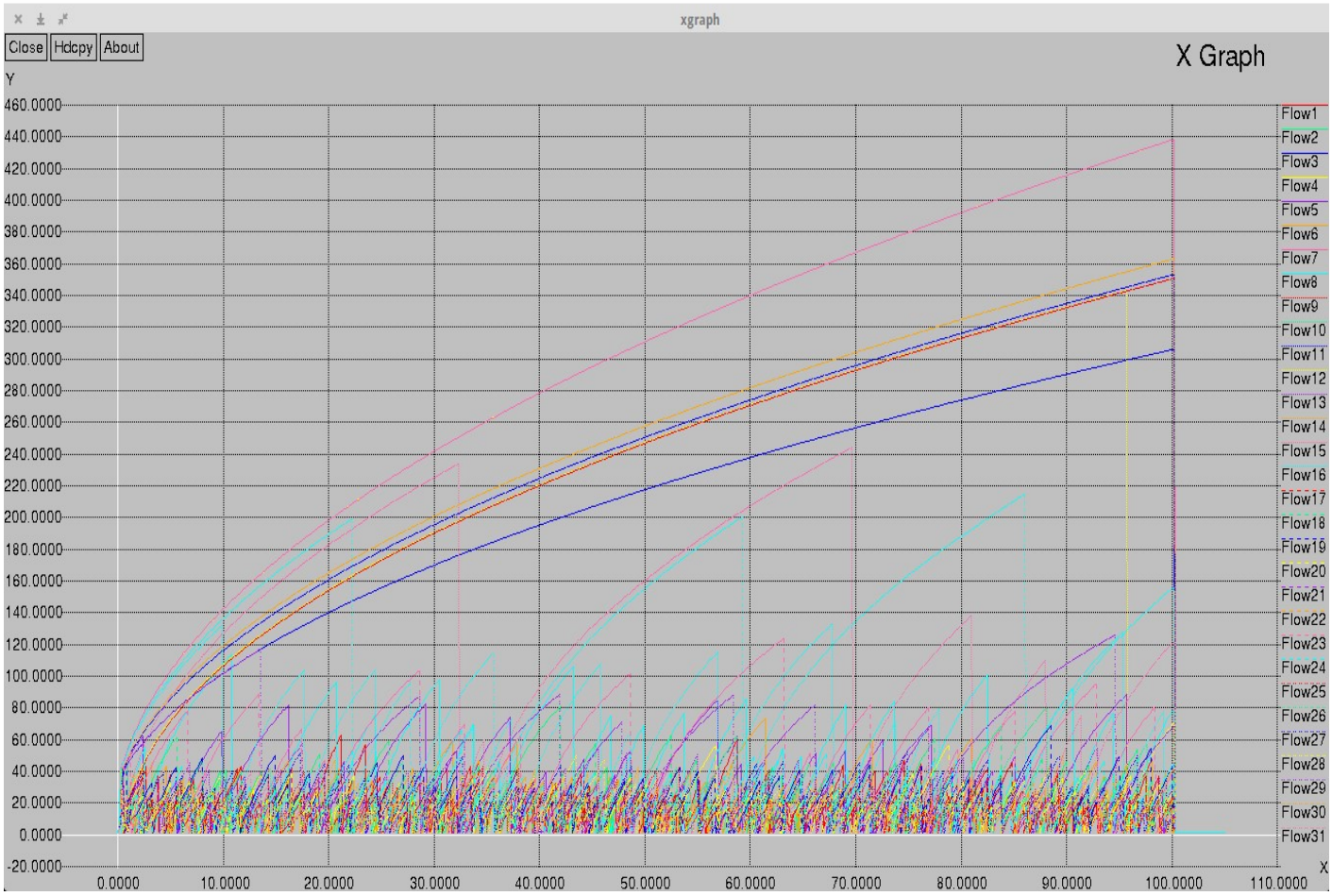
Σχ.4: Το παράθυρο συμφόρησης στο χρόνο, Flow14 στον κόμβο n9

Όπως φαίνεται στο Σχ.2 το παράθυρο σε αρκετές ροές φτάνει σε μεγάλες τιμές μέχρι το χρόνο που έτρεξε η προσομοίωση σε αντίθεση με τις προσομοιώσεις παρακάτω όπου αυξάνεται το όριο του παραθύρου. Στο Σχ.3 παρατηρείται ότι στο Flow27 δεν υπάρχει απώλεια πακέτων με αποτέλεσμα το cwnd να αυξάνεται (διπλασιάζεται) σε κάθε RTT ενώ όπως φαίνεται στο Σχ.4 το Flow14 έχει μικρότερα παράθυρα.

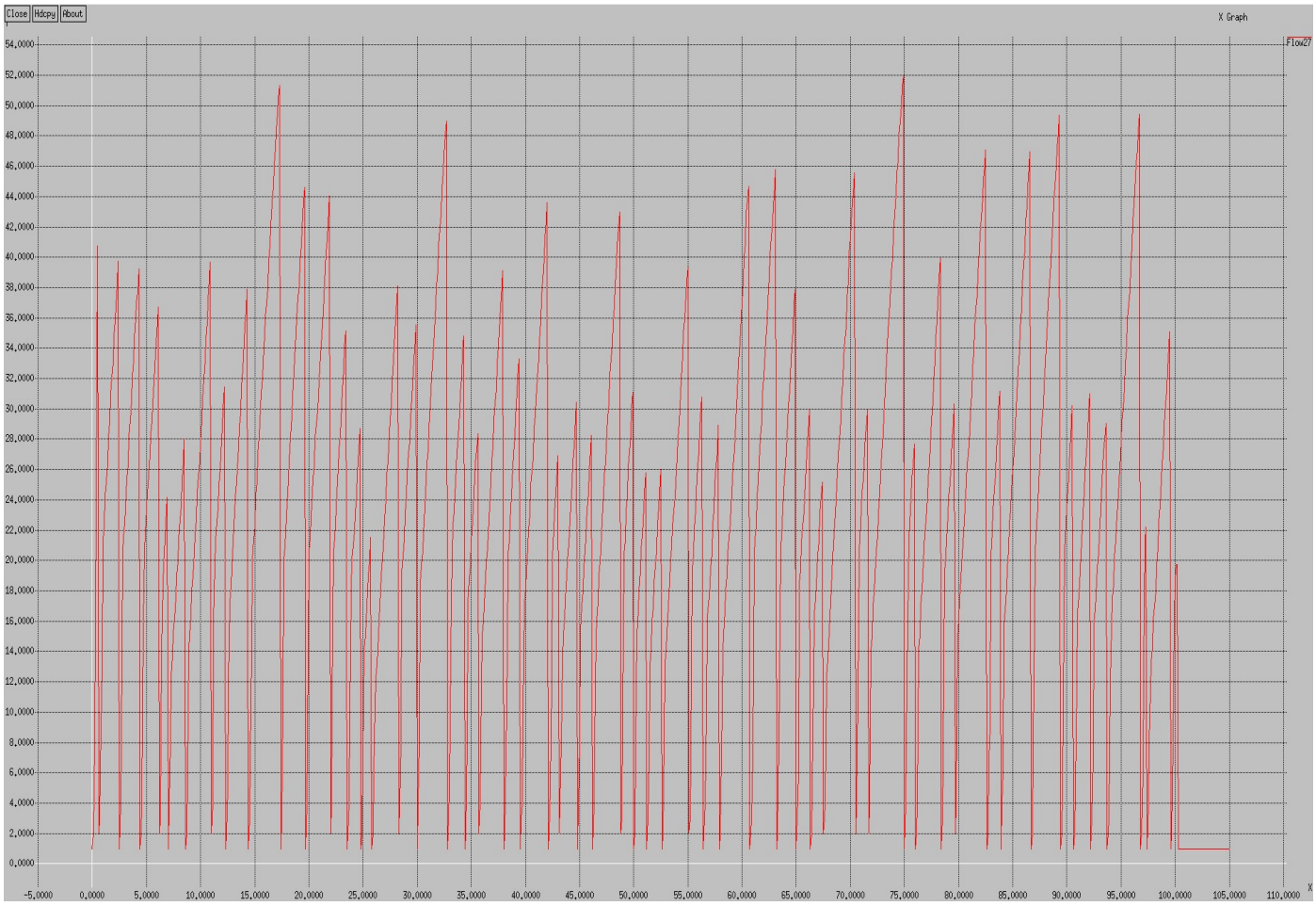
6.1.3 Υπόθεση β:

- Το μέγιστο όριο για το μέγεθος του παραθύρου αυξήθηκε σε 40 από 20 που είναι το προκαθορισμένο.

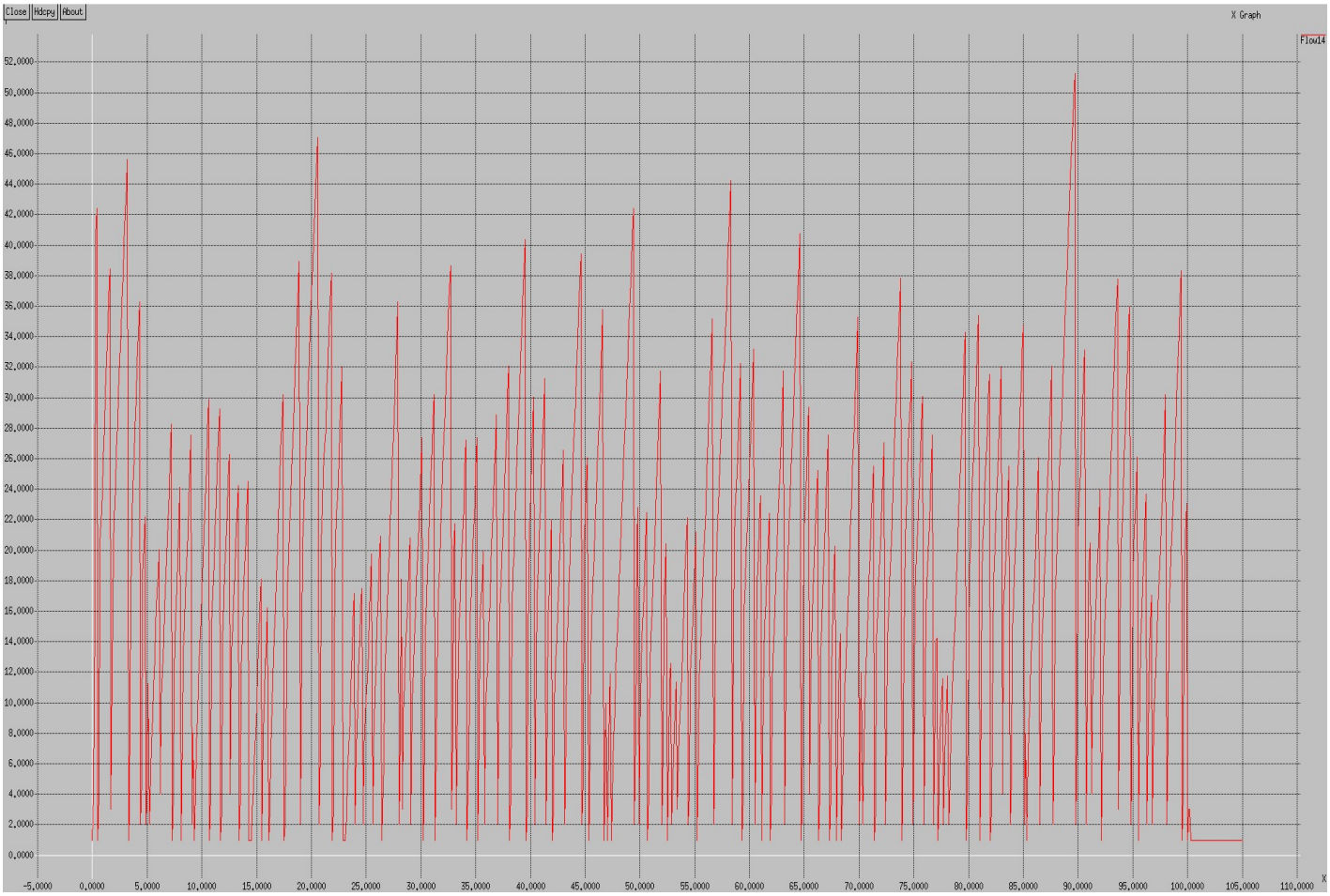
Στην πραγματικότητα διπλασιάστηκε το όριο του αριθμού πακέτων στο παράθυρο. Σε σχέση με το προηγούμενο (Σχ.2) εδώ παρατηρείται μεγαλύτερη συμφόρηση.



Σχ.5: Το παράθυρο συμφόρησης στο χρόνο, 32 flows

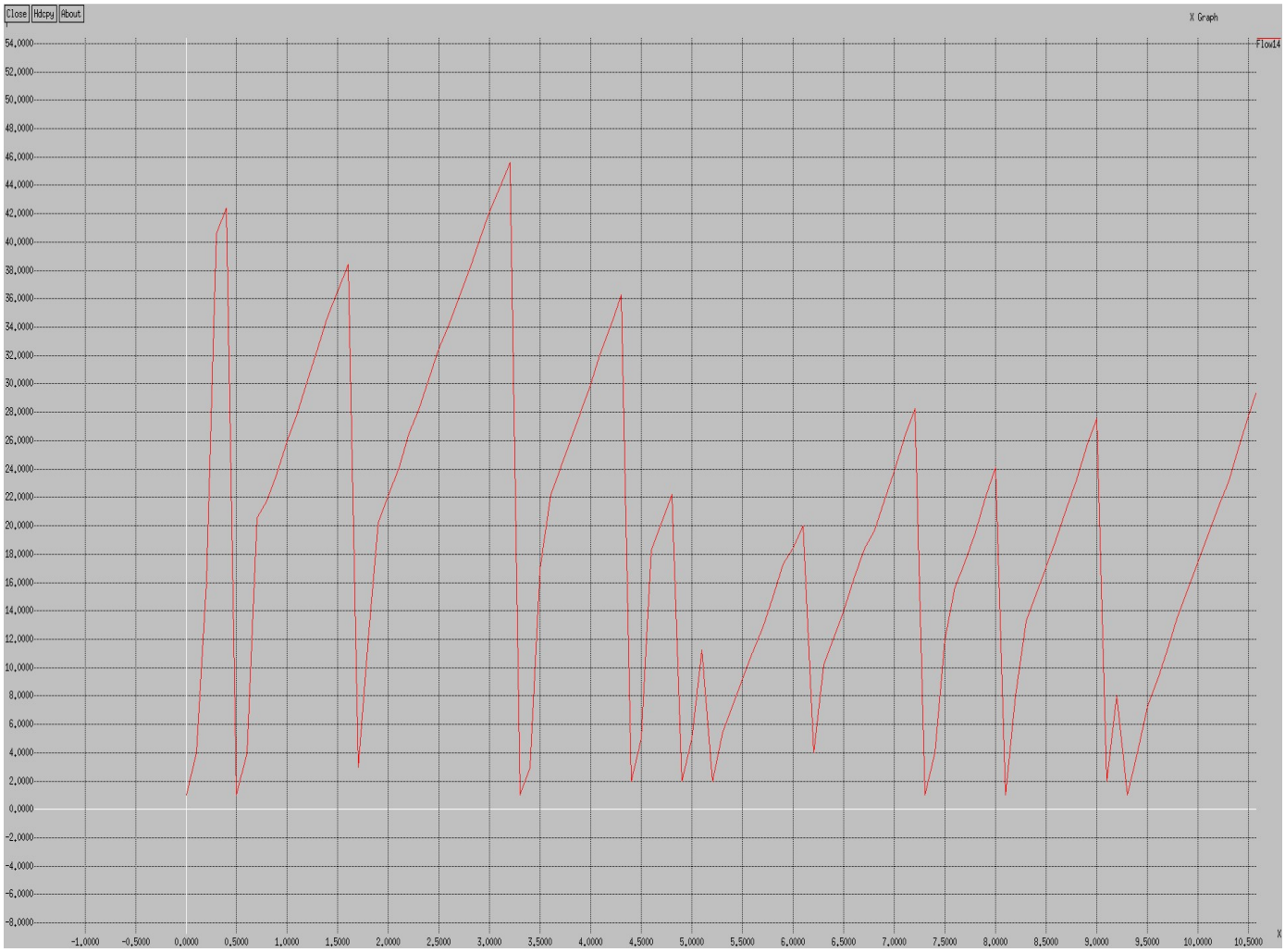


Σχ.6: Το παράθυρο συμφόρησης στο χρόνο, Flow27 στον κόμβο n17



Σχ.7: Το παράθυρο συμφόρησης στο χρόνο, Flow14 στον κόμβο n9

Στο Σχ.6 το Flow27 αυτή τη φορά δεν έχει ομαλή ροή και αντιμετωπίζει προβλήματα συμφόρησης. Από πιο κοντά στο Flow14 στην αρχή φαίνεται η φάση αργής εκκίνησης, η εξέλιξη στο χρόνο και η αύξηση αριθμών πακέτων.

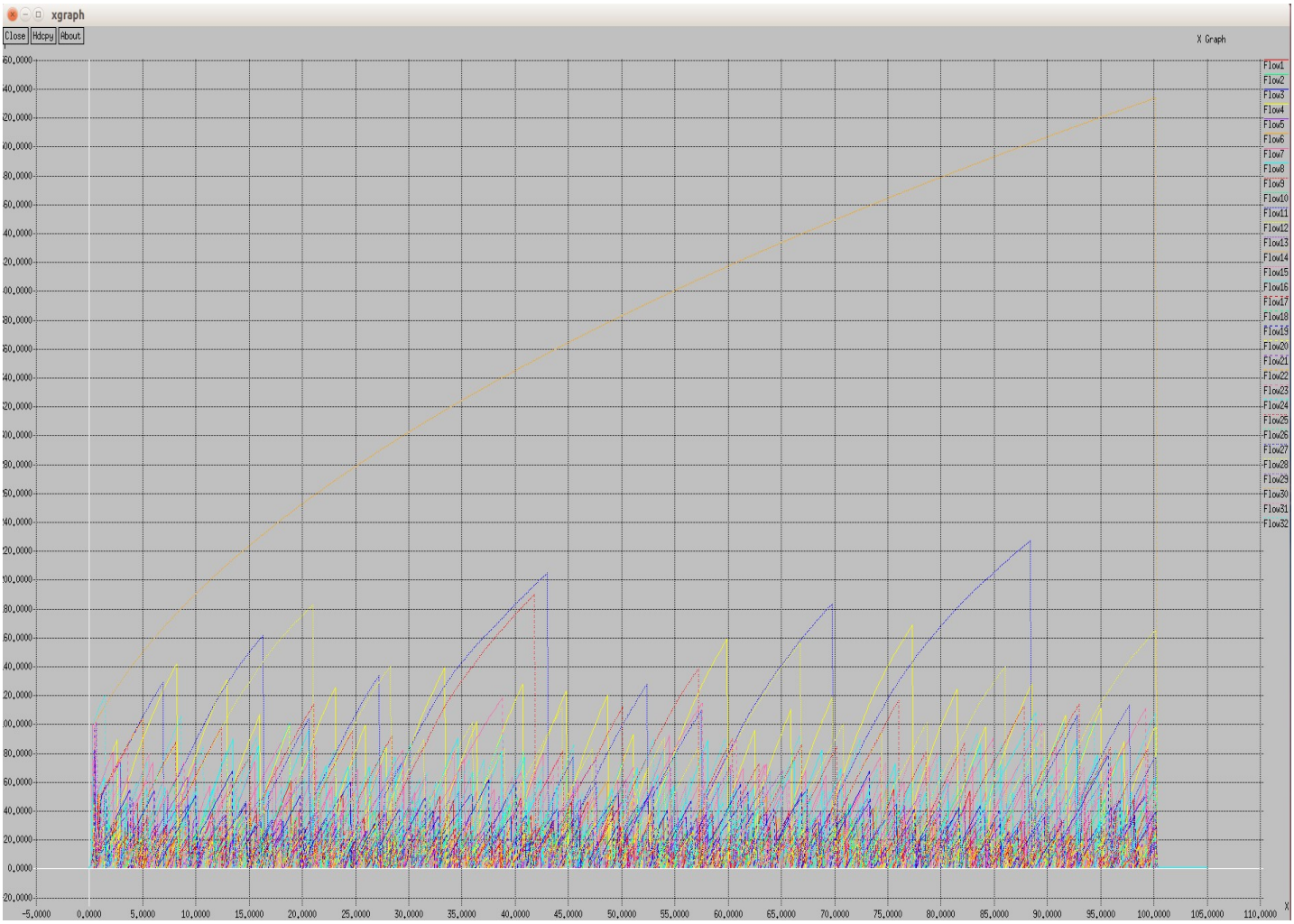


Σχ.8: Το παράθυρο συμφόρησης στο χρόνο, Flow14 στην αρχή

Εδώ φαίνεται η εκθετική αρχική φάση και στη συνέχεια τη χρονική στιγμή περίπου 0.4 όπου το παράθυρο έχει φτάσει τη τιμή 42 πέφτει στη τιμή 1 και ακολουθεί πάλι εκθετική αργή εκκίνηση. Ο αλγόριθμος αφού είδε τις δυνατότητες του δικτύου αυτή τη φορά στο μισό του προηγούμενου αλλάζει σε γραμμική αύξηση του ρυθμού αποστολής.

6.1.4 Υπόθεση γ:

- Το μέγιστο όριο για το μέγεθος του παραθύρου αυξήθηκε σε 100 πακέτα.



Σχ.9: Μέγιστο όριο για το μέγεθος του παραθύρου 100

Η εικόνα διαφέρει πολύ σε σχέση με το Σχ.2 όπου το παράθυρο σε αρκετές ροές έφτανε σε μεγάλες τιμές, εδώ υπάρχουν πιο συχνά μικρότερα παράθυρα σε όλες τις ροές και μέγιστη τιμή παραθύρου 200.

Η διαφορετική συμπεριφορά των δυο ροών στην υπόθεση α βασίζεται στην τοπολογία της προσομοίωσης, το n9 είναι ένας κεντρικός κόμβος που συνδέεται με 6 links για να επικοινωνεί με άλλους κόμβους και τα links αυτά χρησιμοποιούνται ταυτόχρονα και από άλλα ζευγάρια, για αυτό και το Flow14 εμφανίζει νωρίτερα και μεγαλύτερη συμφόρηση και άρα μικρότερα παράθυρα σε σχέση με το Flow27 όπου δεν παρατηρείτε απώλεια πακέτων με αποτέλεσμα το cwnd να αυξάνεται σε κάθε RTT.

Κεφάλαιο 7 Συμπεράσματα

Στην πτυχιακή μας εργασία επανεξετάσαμε τις υπάρχουσες προτάσεις αποφυγής συμφόρησης TCP και τους μηχανισμούς που περιλαμβάνουν. Σκοπός ήταν να αναδείξουμε τα κίνητρα μιας νέας κατεύθυνσης για τη χρησιμοποίηση του δικτύου στο μέγιστο. Στις προσομοιώσεις παρατηρήσαμε την συμπεριφορά του παραθύρου συμφόρησης ορίζοντας μεγαλύτερη τιμή στο όριο αποστολής πακέτων, με αποτέλεσμα να προκαλείται μεγαλύτερη συμφόρηση.

Αυτό μας δείχνει πόσο σημαντικό είναι το cwnd δηλ, το μήκος του παραθύρου που χρησιμοποιεί ο αποστολέας για τη μετάδοση των δεδομένων, να είναι μικρότερο ή ίσο με το πιο πρόσφατο μήκος παραθύρου που ανακοίνωσε ο παραλήπτης στον αποστολέα. Η τιμή που ορίζει ο παραλήπτης χρησιμοποιείται ως άνω φράγμα στο cwnd για την αποφυγή συμφόρησης. Χωρίς τον περιορισμό που θέτει ο έλεγχος συμφόρησης στον αποστολέα το cwnd θα ήταν πάντα ίσο με το παράθυρο του παραλήπτη και θα είχε ως αποτέλεσμα να λαμβάνει υπόψη μόνο το buffer space, όπου κάτι τέτοιο οδηγεί σε συμφόρηση.

Η εφαρμογή αλγορίθμων ελέγχου συμφόρησης σε ένα δίκτυο το οποίο κατακλύζεται από συμφόρηση, κατέδειξε το σημαντικό ρόλο που κατέχει ο έλεγχος συμφόρησης στην αποστολή και λήψη μηνυμάτων αλλά και στην κινητικότητα του δικτύου. Όσο τα δίκτυα αναπτύσσονται τόσο μεγαλώνει η ανάγκη για την εύρεση νέων πιο σύγχρονων τεχνικών για τον έλεγχο της συμφόρησης. Αναπόφευκτα θα συνεχίσει να αποτελεί έρευνα για τα μετέπειτα χρόνια.

Το TCP θα πρέπει να βελτιώσει την απόδοσή του για να ταιριάξει στα επόμενα δίκτυα και να αυξήσει την ταχύτητα μετάδοσης. Για την ανάπτυξη πρωτοκόλλων με μεγάλο bandwidth delay product θα πρέπει να υπάρχει μια ιδιαίτερα προσεκτική ισορροπία μεταξύ της αυξημένης επιθετικότητας της δικαιοσύνης και της ασφάλειας, για αυτό και θα πρέπει να χρησιμοποιηθούν οι πληροφορίες καθυστέρησης ως δείκτες για το κατάλληλο επίπεδο της επιθετικότητας.

Κεφάλαιο 8 Ns2 tcl κώδικας

```
#===================================================================#
                        Initialization
#===================================================================#

#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open out.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile

#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    close $tracefile
    close $namfile
#    exec nam out.nam &
    exit 0
}


#===================================================================#
                    set max bound on window size
#===================================================================#

Agent/TCP set window_ 20


#===================================================================#
                    set queue Type & buffer sizes
#===================================================================#

set queueType       DropTail

#===================================================================#
                    Set Mb to all connections
#===================================================================#

set Mb          20Mb
```

```
#=================================================================#
                        Set FTP Start-Stop
#=================================================================#

set time_start     0.0
set time_stop      100.0

#=================================================================#
                        Set simulation end time
#=================================================================#

$ns at 105.0 "finish"

#=================================================================#
                        Nodes Definition
#=================================================================#

#Create 19 nodes
for {set i 1} {$i <= 19} {incr i 1} {
set n$i [$ns node]
}

#=================================================================#
                        Links Definition
#=================================================================#

#Links between nodes

$ns duplex-link $n1 $n3 $Mb 10ms $queueType
$ns duplex-link $n1 $n8 $Mb 10ms $queueType
$ns duplex-link $n1 $n2 $Mb 10ms $queueType
$ns duplex-link $n2 $n4 $Mb 10ms $queueType
$ns duplex-link $n2 $n5 $Mb 10ms $queueType
$ns duplex-link $n2 $n6 $Mb 10ms $queueType
$ns duplex-link $n3 $n2 $Mb 10ms $queueType
$ns duplex-link $n3 $n7 $Mb 10ms $queueType
$ns duplex-link $n3 $n10 $Mb 10ms $queueType
$ns duplex-link $n4 $n9 $Mb 10ms $queueType
$ns duplex-link $n5 $n6 $Mb 10ms $queueType
$ns duplex-link $n5 $n8 $Mb 10ms $queueType
$ns duplex-link $n6 $n7 $Mb 10ms $queueType
$ns duplex-link $n6 $n8 $Mb 10ms $queueType
$ns duplex-link $n7 $n8 $Mb 10ms $queueType
$ns duplex-link $n8 $n9 $Mb 10ms $queueType
$ns duplex-link $n8 $n13 $Mb 10ms $queueType
$ns duplex-link $n9 $n10 $Mb 10ms $queueType
```

```
$ns duplex-link $n9 $n11 $Mb 10ms $queueType
$ns duplex-link $n9 $n12 $Mb 10ms $queueType
$ns duplex-link $n9 $n17 $Mb 10ms $queueType
$ns duplex-link $n10 $n16 $Mb 10ms $queueType
$ns duplex-link $n11 $n15 $Mb 10ms $queueType
$ns duplex-link $n12 $n13 $Mb 10ms $queueType
$ns duplex-link $n12 $n14 $Mb 10ms $queueType
$ns duplex-link $n12 $n18 $Mb 10ms $queueType
$ns duplex-link $n13 $n14 $Mb 10ms $queueType
$ns duplex-link $n13 $n18 $Mb 10ms $queueType
$ns duplex-link $n14 $n15 $Mb 10ms $queueType
$ns duplex-link $n14 $n18 $Mb 10ms $queueType
$ns duplex-link $n15 $n16 $Mb 10ms $queueType
$ns duplex-link $n15 $n19 $Mb 10ms $queueType
$ns duplex-link $n16 $n17 $Mb 10ms $queueType
$ns duplex-link $n17 $n19 $Mb 10ms $queueType
$ns duplex-link $n19 $n18 $Mb 10ms $queueType

#=============================================================#
                        Agent Definition
#=============================================================#

#Setup a TCP connection          n1 to n9
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n9 $sink1
$ns connect $tcp1 $sink1
$tcp1 set fid_ 1

#Setup a TCP connection          n1 to n19
set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n19 $sink2
$ns connect $tcp2 $sink2
$tcp2 set fid_ 2

#Setup a TCP connection          n2 to n18
set tcp3 [new Agent/TCP]
$ns attach-agent $n2 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n18 $sink3
$ns connect $tcp3 $sink3
$tcp3 set fid_ 3
```

```
#Setup a TCP connection          n3 to n4
set tcp4 [new Agent/TCP]
$ns attach-agent $n3 $tcp4
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
$ns connect $tcp4 $sink4
$tcp4 set fid_ 4

#Setup a TCP connection          n3 to n11
set tcp5 [new Agent/TCP]
$ns attach-agent $n3 $tcp5
set sink5 [new Agent/TCPSink]
$ns attach-agent $n11 $sink5
$ns connect $tcp5 $sink5
$tcp5 set fid_ 5

#Setup a TCP connection          n3 to n17
set tcp6 [new Agent/TCP]
$ns attach-agent $n3 $tcp6
set sink6 [new Agent/TCPSink]
$ns attach-agent $n17 $sink6
$ns connect $tcp6 $sink6
$tcp6 set fid_ 6

#Setup a TCP connection          n4 to n3
set tcp7 [new Agent/TCP]
$ns attach-agent $n4 $tcp7
set sink7 [new Agent/TCPSink]
$ns attach-agent $n3 $sink7
$ns connect $tcp7 $sink7
$tcp7 set fid_ 7

#Setup a TCP connection          n4 to n10
set tcp8 [new Agent/TCP]
$ns attach-agent $n4 $tcp8
set sink8 [new Agent/TCPSink]
$ns attach-agent $n10 $sink8
$ns connect $tcp8 $sink8
$tcp8 set fid_ 8

#Setup a TCP connection          n4 to n13
set tcp9 [new Agent/TCP]
$ns attach-agent $n4 $tcp9
set sink9 [new Agent/TCPSink]
$ns attach-agent $n13 $sink9
$ns connect $tcp9 $sink9
```

```
$tcp9 set fid_ 9

#Setup a TCP connection          n5 to n12
set tcp10 [new Agent/TCP]
$ns attach-agent $n5 $tcp10
set sink10 [new Agent/TCPSink]
$ns attach-agent $n12 $sink10
$ns connect $tcp10 $sink10
$tcp10 set fid_ 10

#Setup a TCP connection          n6 to n14
set tcp11 [new Agent/TCP]
$ns attach-agent $n6 $tcp11
set sink11 [new Agent/TCPSink]
$ns attach-agent $n14 $sink11
$ns connect $tcp11 $sink11
$tcp11 set fid_ 11

#Setup a TCP connection          n7 to n16
set tcp12 [new Agent/TCP]
$ns attach-agent $n7 $tcp12
set sink12 [new Agent/TCPSink]
$ns attach-agent $n16 $sink12
$ns connect $tcp12 $sink12
$tcp12 set fid_ 12

#Setup a TCP connection          n8 to n15
set tcp13 [new Agent/TCP]
$ns attach-agent $n8 $tcp13
set sink13 [new Agent/TCPSink]
$ns attach-agent $n15 $sink13
$ns connect $tcp13 $sink13
$tcp13 set fid_ 1

#Setup a TCP connection          n9 to n1
set tcp14 [new Agent/TCP]
$ns attach-agent $n9 $tcp14
set sink14 [new Agent/TCPSink]
$ns attach-agent $n1 $sink14
$ns connect $tcp14 $sink14
$tcp14 set fid_ 13

#Setup a TCP connection          n9 to n19
set tcp15 [new Agent/TCP]
$ns attach-agent $n9 $tcp15
set sink15 [new Agent/TCPSink]
```

```
$ns attach-agent $n19 $sink15
$ns connect $tcp15 $sink15
$tcp15 set fid_ 14

#Setup a TCP connection          n10 to n4
set tcp16 [new Agent/TCP]
$ns attach-agent $n10 $tcp16
set sink16 [new Agent/TCPSink]
$ns attach-agent $n4 $sink16
$ns connect $tcp16 $sink16
$tcp16 set fid_ 15

#Setup a TCP connection          n10 to n13
set tcp17 [new Agent/TCP]
$ns attach-agent $n10 $tcp17
set sink17 [new Agent/TCPSink]
$ns attach-agent $n13 $sink17
$ns connect $tcp17 $sink17
$tcp17 set fid_ 16

#Setup a TCP connection          n11 to n3
set tcp18 [new Agent/TCP]
$ns attach-agent $n11 $tcp18
set sink18 [new Agent/TCPSink]
$ns attach-agent $n3 $sink18
$ns connect $tcp18 $sink18
$tcp18 set fid_ 17

#Setup a TCP connection          n12 to n5
set tcp19 [new Agent/TCP]
$ns attach-agent $n12 $tcp19
set sink19 [new Agent/TCPSink]
$ns attach-agent $n5 $sink19
$ns connect $tcp19 $sink19
$tcp19 set fid_ 18

#Setup a TCP connection          n13 to n4
set tcp20 [new Agent/TCP]
$ns attach-agent $n13 $tcp20
set sink20 [new Agent/TCPSink]
$ns attach-agent $n4 $sink20
$ns connect $tcp20 $sink20
$tcp20 set fid_ 19

#Setup a TCP connection          n13 to n10
set tcp21 [new Agent/TCP]
```

```
$ns attach-agent $n13 $tcp21
set sink21 [new Agent/TCPSink]
$ns attach-agent $n10 $sink21
$ns connect $tcp21 $sink21
$tcp21 set fid_ 20

#Setup a TCP connection          n13 to n17
set tcp22 [new Agent/TCP]
$ns attach-agent $n13 $tcp22
set sink22 [new Agent/TCPSink]
$ns attach-agent $n17 $sink22
$ns connect $tcp22 $sink22
$tcp22 set fid_ 21

#Setup a TCP connection          n14 to n6
set tcp23 [new Agent/TCP]
$ns attach-agent $n14 $tcp23
set sink23 [new Agent/TCPSink]
$ns attach-agent $n6 $sink23
$ns connect $tcp23 $sink23
$tcp23 set fid_ 22

#Setup a TCP connection          n15 to n8
set tcp24 [new Agent/TCP]
$ns attach-agent $n15 $tcp24
set sink24 [new Agent/TCPSink]
$ns attach-agent $n8 $sink24
$ns connect $tcp24 $sink24
$tcp24 set fid_ 23

#Setup a TCP connection          n16 to n7
set tcp25 [new Agent/TCP]
$ns attach-agent $n16 $tcp25
set sink25 [new Agent/TCPSink]
$ns attach-agent $n7 $sink25
$ns connect $tcp25 $sink25
$tcp25 set fid_ 24

#Setup a TCP connection          n16 to n18
set tcp26 [new Agent/TCP]
$ns attach-agent $n16 $tcp26
set sink26 [new Agent/TCPSink]
$ns attach-agent $n18 $sink26
$ns connect $tcp26 $sink26
$tcp26 set fid_ 25
```

```
#Setup a TCP connection          n17 to n3
set tcp27 [new Agent/TCP]
$ns attach-agent $n17 $tcp27
set sink27 [new Agent/TCPSink]
$ns attach-agent $n3 $sink27
$ns connect $tcp27 $sink27
$tcp27 set fid_ 26

#Setup a TCP connection          n17 to n13
set tcp28 [new Agent/TCP]
$ns attach-agent $n17 $tcp28
set sink28 [new Agent/TCPSink]
$ns attach-agent $n13 $sink28
$ns connect $tcp28 $sink28
$tcp28 set fid_ 27

#Setup a TCP connection          n18 to n2
set tcp29 [new Agent/TCP]
$ns attach-agent $n18 $tcp29
set sink29 [new Agent/TCPSink]
$ns attach-agent $n2 $sink29
$ns connect $tcp29 $sink29
$tcp29 set fid_ 28

#Setup a TCP connection          n18 to n16
set tcp30 [new Agent/TCP]
$ns attach-agent $n18 $tcp30
set sink30 [new Agent/TCPSink]
$ns attach-agent $n16 $sink30
$ns connect $tcp30 $sink30
$tcp30 set fid_ 29

#Setup a TCP connection          n19 to n1
set tcp31 [new Agent/TCP]
$ns attach-agent $n19 $tcp31
set sink31 [new Agent/TCPSink]
$ns attach-agent $n1 $sink31
$ns connect $tcp31 $sink31
$tcp31 set fid_ 30

#Setup a TCP connection          n19 to n9
set tcp32 [new Agent/TCP]
$ns attach-agent $n19 $tcp32
set sink32 [new Agent/TCPSink]
$ns attach-agent $n9 $sink32
$ns connect $tcp32 $sink32
```

```
#=====================================================================#
                        Applications Definition
#=====================================================================#

#Setup a FTP Application over TCP connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at $time_start "$ftp1 start"
$ns at $time_stop "$ftp1 stop"

#Setup a FTP Application over TCP connection
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns at $time_start "$ftp2 start"
$ns at $time_stop "$ftp2 stop"

#Setup a FTP Application over TCP connection
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ns at $time_start "$ftp3 start"
$ns at $time_stop "$ftp3 stop"

#Setup a FTP Application over TCP connection
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$ns at $time_start "$ftp4 start"
$ns at $time_stop "$ftp4 stop"

#Setup a FTP Application over TCP connection
set ftp5 [new Application/FTP]
$ftp5 attach-agent $tcp5
$ns at $time_start "$ftp5 start"
$ns at $time_stop "$ftp5 stop"

#Setup a FTP Application over TCP connection
set ftp6 [new Application/FTP]
$ftp6 attach-agent $tcp6
$ns at $time_start "$ftp6 start"
$ns at $time_stop "$ftp6 stop"

#Setup a FTP Application over TCP connection
set ftp7 [new Application/FTP]
$ftp7 attach-agent $tcp7
$ns at $time_start "$ftp7 start"
$ns at $time_stop "$ftp7 stop"
```

```
#Setup a FTP Application over TCP connection
set ftp8 [new Application/FTP]
$ftp8 attach-agent $tcp8
$ns at $time_start "$ftp8 start"
$ns at $time_stop "$ftp8 stop"

#Setup a FTP Application over TCP connection
set ftp9 [new Application/FTP]
$ftp9 attach-agent $tcp9
$ns at $time_start "$ftp9 start"
$ns at $time_stop "$ftp9 stop"

#Setup a FTP Application over TCP connection
set ftp10 [new Application/FTP]
$ftp10 attach-agent $tcp10
$ns at $time_start "$ftp10 start"
$ns at $time_stop "$ftp10 stop"

#Setup a FTP Application over TCP connection
set ftp11 [new Application/FTP]
$ftp11 attach-agent $tcp11
$ns at $time_start "$ftp11 start"
$ns at $time_stop "$ftp11 stop"

#Setup a FTP Application over TCP connection
set ftp12 [new Application/FTP]
$ftp12 attach-agent $tcp12
$ns at $time_start "$ftp12 start"
$ns at $time_stop "$ftp12 stop"

#Setup a FTP Application over TCP connection
set ftp13 [new Application/FTP]
$ftp13 attach-agent $tcp13
$ns at $time_start "$ftp13 start"
$ns at $time_stop "$ftp13 stop"

#Setup a FTP Application over TCP connection
set ftp14 [new Application/FTP]
$ftp14 attach-agent $tcp14
$ns at $time_start "$ftp14 start"
$ns at $time_stop "$ftp14 stop"

#Setup a FTP Application over TCP connection
set ftp15 [new Application/FTP]
$ftp15 attach-agent $tcp15
$ns at $time_start "$ftp15 start"
```

```
$ns at $time_stop "$ftp15 stop"

#Setup a FTP Application over TCP connection
set ftp16 [new Application/FTP]
$ftp16 attach-agent $tcp16
$ns at $time_start "$ftp16 start"
$ns at $time_stop "$ftp16 stop"

#Setup a FTP Application over TCP connection
set ftp17 [new Application/FTP]
$ftp17 attach-agent $tcp17
$ns at $time_start "$ftp17 start"
$ns at $time_stop "$ftp17 stop"

#Setup a FTP Application over TCP connection
set ftp18 [new Application/FTP]
$ftp18 attach-agent $tcp18
$ns at $time_start "$ftp18 start"
$ns at $time_stop "$ftp18 stop"

#Setup a FTP Application over TCP connection
set ftp19 [new Application/FTP]
$ftp19 attach-agent $tcp19
$ns at $time_start "$ftp19 start"
$ns at $time_stop "$ftp19 stop"

#Setup a FTP Application over TCP connection
set ftp20 [new Application/FTP]
$ftp20 attach-agent $tcp20
$ns at $time_start "$ftp20 start"
$ns at $time_stop "$ftp20 stop"

#Setup a FTP Application over TCP connection
set ftp21 [new Application/FTP]
$ftp21 attach-agent $tcp21
$ns at $time_start "$ftp21 start"
$ns at $time_stop "$ftp21 stop"

#Setup a FTP Application over TCP connection
set ftp22 [new Application/FTP]
$ftp22 attach-agent $tcp22
$ns at $time_start "$ftp22 start"
$ns at $time_stop "$ftp22 stop"

#Setup a FTP Application over TCP connection
set ftp23 [new Application/FTP]
```

```
$ftp23 attach-agent $tcp23
$ns at $time_start "$ftp23 start"
$ns at $time_stop "$ftp23 stop"

#Setup a FTP Application over TCP connection
set ftp24 [new Application/FTP]
$ftp24 attach-agent $tcp24
$ns at $time_start "$ftp24 start"
$ns at $time_stop "$ftp24 stop"

#Setup a FTP Application over TCP connection
set ftp25 [new Application/FTP]
$ftp25 attach-agent $tcp25
$ns at $time_start "$ftp25 start"
$ns at $time_stop "$ftp25 stop"

#Setup a FTP Application over TCP connection
set ftp26 [new Application/FTP]
$ftp26 attach-agent $tcp26
$ns at $time_start "$ftp26 start"
$ns at $time_stop "$ftp26 stop"

#Setup a FTP Application over TCP connection
set ftp27 [new Application/FTP]
$ftp27 attach-agent $tcp27
$ns at $time_start "$ftp27 start"
$ns at $time_stop "$ftp27 stop"

#Setup a FTP Application over TCP connection
set ftp28 [new Application/FTP]
$ftp28 attach-agent $tcp28
$ns at $time_start "$ftp28 start"
$ns at $time_stop "$ftp28 stop"

#Setup a FTP Application over TCP connection
set ftp29 [new Application/FTP]
$ftp29 attach-agent $tcp29
$ns at $time_start "$ftp29 start"
$ns at $time_stop "$ftp29 stop"

#Setup a FTP Application over TCP connection
set ftp30 [new Application/FTP]
$ftp30 attach-agent $tcp30
$ns at $time_start "$ftp30 start"
$ns at $time_stop "$ftp30 stop"
```

```
#Setup a FTP Application over TCP connection
set ftp31 [new Application/FTP]
$ftp31 attach-agent $tcp31
$ns at $time_start "$ftp31 start"
$ns at $time_stop "$ftp31 stop"

#Setup a FTP Application over TCP connection
set ftp32 [new Application/FTP]
$ftp32 attach-agent $tcp32
$ns at $time_start "$ftp32 start"
$ns at $time_stop "$ftp32 stop"


#=============================================================#
                    Obtain CWND from TCP agent
#=============================================================#

 proc plotWindow {tcpSource outfile} {
   global ns

   set time 0.1
   set now [$ns now]
   set cwnd [$tcpSource set cwnd_]
   set wnd [$tcpSource set window_]

   puts  $outfile  "$now $cwnd"

   $ns at [expr $now+0.1] "plotWindow $tcpSource  $outfile"
 }


set outfile1 [open  "Flow1"  w]
set outfile2 [open  "Flow2"  w]
set outfile3 [open  "Flow3"  w]
set outfile4 [open  "Flow4"  w]
set outfile5 [open  "Flow5"  w]
set outfile6 [open  "Flow6"  w]
set outfile7 [open  "Flow7"  w]
set outfile8 [open  "Flow8"  w]
set outfile9 [open  "Flow9"  w]
set outfile10 [open  "Flow10"  w]
set outfile11 [open  "Flow11"  w]
set outfile12 [open  "Flow12"  w]
set outfile13 [open  "Flow13"  w]
set outfile14 [open  "Flow14"  w]
set outfile15 [open  "Flow15"  w]
```

```
set outfile16 [open  "Flow16"  w]
set outfile17 [open  "Flow17"  w]
set outfile18 [open  "Flow18"  w]
set outfile19 [open  "Flow19"  w]
set outfile20 [open  "Flow20"  w]
set outfile21 [open  "Flow21"  w]
set outfile22 [open  "Flow22"  w]
set outfile23 [open  "Flow23"  w]
set outfile24 [open  "Flow24"  w]
set outfile25 [open  "Flow25"  w]
set outfile26 [open  "Flow26"  w]
set outfile27 [open  "Flow27"  w]
set outfile28 [open  "Flow28"  w]
set outfile29 [open  "Flow29"  w]
set outfile30 [open  "Flow30"  w]
set outfile31 [open  "Flow31"  w]
set outfile32 [open  "Flow32"  w]

$ns at 0.0 "plotWindow $tcp1 $outfile1"
$ns at 0.0 "plotWindow $tcp2 $outfile2"
$ns at 0.0 "plotWindow $tcp3 $outfile3"
$ns at 0.0 "plotWindow $tcp4 $outfile4"
$ns at 0.0 "plotWindow $tcp5 $outfile5"
$ns at 0.0 "plotWindow $tcp6 $outfile6"
$ns at 0.0 "plotWindow $tcp7 $outfile7"
$ns at 0.0 "plotWindow $tcp8 $outfile8"
$ns at 0.0 "plotWindow $tcp9 $outfile9"
$ns at 0.0 "plotWindow $tcp10 $outfile10"
$ns at 0.0 "plotWindow $tcp11 $outfile11"
$ns at 0.0 "plotWindow $tcp12 $outfile12"
$ns at 0.0 "plotWindow $tcp13 $outfile13"
$ns at 0.0 "plotWindow $tcp14 $outfile14"
$ns at 0.0 "plotWindow $tcp15 $outfile15"
$ns at 0.0 "plotWindow $tcp16 $outfile16"
$ns at 0.0 "plotWindow $tcp17 $outfile17"
$ns at 0.0 "plotWindow $tcp18 $outfile18"
$ns at 0.0 "plotWindow $tcp19 $outfile19"
$ns at 0.0 "plotWindow $tcp20 $outfile20"
$ns at 0.0 "plotWindow $tcp21 $outfile21"
$ns at 0.0 "plotWindow $tcp22 $outfile22"
$ns at 0.0 "plotWindow $tcp23 $outfile23"
$ns at 0.0 "plotWindow $tcp24 $outfile24"
$ns at 0.0 "plotWindow $tcp25 $outfile25"
$ns at 0.0 "plotWindow $tcp26 $outfile26"
$ns at 0.0 "plotWindow $tcp27 $outfile27"
$ns at 0.0 "plotWindow $tcp28 $outfile28"
```

```
$ns at 0.0 "plotWindow $tcp29 $outfile29"
$ns at 0.0 "plotWindow $tcp30 $outfile30"
$ns at 0.0 "plotWindow $tcp31 $outfile31"
$ns at 0.0 "plotWindow $tcp32 $outfile32"



$ns run
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] V. Jacobson, Michael J. Karels, "Congestion avoidance and control," ACM SIGCOMM, November 1988.

[2] K. Oyeyinka, A. Oluwatope, A. Akinwale, O. Folorunso, G. Aderounmu and O. Abiona, "TCP Window Based Congestion Control -Slow-Start Approach," Communications and Network, Vol. 3 No. 2, 2011

[3] Ryan King, Richard Baraniuk, Rudolf Riedi, "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP", Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005.

[4] Habibullah Jamal, Kiran Sultan, "Performance Analysis of TCP Congestion Control Algorithms", International journal of computers and communications, Issue 1, Volume 2, 2008.

[5] G. Thaker, J. Cain, "Interactions Between Routing and Flow Control Algorithm" , IEEE Transactions on Communications, Mar 1986

[6] The Network Simulator - ns-2 (http://www.isi.edu/nsnam/ns/index.html)

- Tcl Tutorial (http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html)
- Οδηγίες χρήσεως xgraph (http://www.xgraph.org/)
- Studying TCP's Congestion Window using NS (http://www.mathcs.emory.edu/~cheung/Courses/558-old/Syllabus/90-NS/3-Perf-Anal/TCP-CWND.html)