ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ &
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση Εφαρμογής Μεταφοράς Αρχείων Βασισμένη στο Πρωτόκολλο SCTP
Πειράματα Αξιολόγησης Πρωτοκόλλου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φλωρίδης Μ. Ανδρέας

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ &
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση Εφαρμογής Μεταφοράς Αρχείων Βασισμένη στο Πρωτόκολλο SCTP Πειράματα Αξιολόγησης Πρωτοκόλλου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φλωρίδης Μ. Ανδρέας

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2005.

.....	.....	.....
Β. Μάγκλαρης	Ε. Συκάς	Σ. Παπαβασιλείου
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Επίκουρος Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2005

.....

Φλωρίδης Μ. Ανδρέας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περιεχόμενα

Πίνακας Πινάκων

Πίνακας Σχημάτων

Περίληψη

Το πρωτόκολλο SCTP είναι ένα νέο πρωτόκολλο μεταφοράς IP το οποίο συνυπάρχει με το TCP και το UDP και προσφέρει υπηρεσίες στο στρώμα μεταφοράς για πολλές εφαρμογές διαδικτύου.

Όπως και το TCP, το SCTP είναι πρωτόκολλο με σύνδεση και προσφέρει αξιόπιστες υπηρεσίες μεταφοράς, εξασφαλίζοντας ότι τα δεδομένα στο δίκτυο θα μεταδοθούν με την σειρά και χωρίς λάθη. Σε αντίθεση με το TCP, το SCTP προσφέρει ένα αριθμό υπηρεσιών, σημαντικές για μετάδοση σηματοδοσίας τηλεφωνίας που ταυτόχρονα μπορούν να ωφελήσουν και άλλες εφαρμογές.

Στα πλαίσια αυτής της διπλωματικής εργασίας έχει αναπτυχθεί μια εφαρμογή μεταφοράς αρχείων η οποία χρησιμοποιεί ως πρωτόκολλο μεταφοράς το SCTP (Stream Control Transmission Protocol) και η αντίστοιχη εφαρμογή που χρησιμοποιεί το TCP. Η εφαρμογή αποτελείται από ένα server και ένα client πρόγραμμα που υποστηρίζουν τη βασική λειτουργικότητα του πρωτοκόλλου FTP. Με τη βοήθεια της εφαρμογής έχουν πραγματοποιηθεί πειράματα για τη σύγκριση της απόδοσης των πρωτοκόλλων SCTP και TCP.

Το λογισμικό που έχει αναπτυχθεί υποστηρίζει δυνατότητα για ταυτόχρονη μεταφορά περισσοτέρων του ενός αρχείων. Με χρήση του SCTP κάθε αρχείο μεταφέρεται σε διαφορετικό stream στο ίδιο SCTP association. Ενώ στην περίπτωση του TCP για κάθε αρχείο χρησιμοποιείται και μια TCP σύνδεση. Τα προγράμματα παρέχουν μέτρηση του ρυθμού μετάδοσης που επιτυγχάνεται κατά τη μεταφορά κάθε αρχείου.

Το περιβάλλον ανάπτυξης είναι το λειτουργικό σύστημα Unix (FreeBSD) και η γλώσσα προγραμματισμού η C. Για το πρωτόκολλο SCTP έγινε χρήση της υλοποίησης SCTP sctplib-1.0 και του socket API σε επίπεδο χρήστη (userland).

Λέξεις Κλειδιά

SCTP, sctplib, socket API, FTP, client, server, multi-streaming, TCP, congestion control

Abstract

The Stream Control Transmission Protocol (SCTP) is a new IP transport protocol, existing at an equivalent level with UDP (User Datagram Protocol) and TCP (Transmission Control Protocol), which provide transport layer functions to many Internet applications.

Like TCP, SCTP provides a reliable transport service, ensuring that data is transported across the network without error and in sequence. Like TCP, SCTP is a session-oriented mechanism Unlike TCP, SCTP provides a number of functions that are critical for telephony signaling transport and at the same time can potentially benefit other applications.

In this diploma thesis we have developed a file transport application which uses the SCTP(Stream Control Transmission Protocol) transport protocol and the corresponding application that uses the TCP. The application is constituted by a server and client program that supports the basic functions of the FTP protocol. With the help of the application we have contacted experiments for the comparison of SCTP and TCP protocols.

The software that has been developed offers the possibility of simultaneous transport of more than one file. With use of SCTP each file is transported in different stream in the same SCTP association, while in the case of TCP each file uses a differed TCP connection. The program provides the throughput of the transmission of each file.

The development environment is the Unix operating system (FreeBSD) and
the program language is C. For the SCTP protocol we have used SCTP implementation sctplib-1.0 and the userland socket API.

Keywords

SCTP, sctplib, socket API, FTP, client, server, multi-streaming, TCP, congestion control

Εισαγωγή

Τα πρωτόκολλά που έχουν κυριαρχήσει στο σημερινό διαδίκτυο έχουν σχεδιαστεί εδώ και πολλά χρόνια , έχουν δοκιμαστεί και τροποποιηθεί πολλές φορές και έχουν ωριμάσει με την πάροδο του χρόνου. Πολλά καινούρια πρωτόκολλα έχουν κάνει την εμφάνιση τους ανά καιρούς όμως λίγα από αυτά καταφέρνουν να γίνουν ευρύτερος γνωστά. Ένα τέτοιο πρωτόκολλο είναι και το SCTP το οποίο ανήκει στο στρώμα μεταφοράς.

Στην παρούσα διπλωματική εργασία, αντικειμενικός μας στόχος ήταν η χρησιμοποίηση του πρωτοκόλλου αυτού σε μια εφαρμογή μεταφοράς αρχείων , έτσι ώστε να μπορούμε να κάνουμε πραγματικές μετρήσεις στην απόδοση του πρωτοκόλλου.

Η μετρήσεις και τα πειράματα δεν έχουν στόχο την γενική αξιολόγηση του SCTP αλλά την χρήση ενός από τα χαρακτηριστικά του, του multi-streaming έτσι ώστε να δούμε αν μπορούμε να έχουμε κέρδος από την μεταφορά δεδομένων σε πολλές ροές (streams) αντί σε διαφορετικές συνδέσεις TCP. Όλα αυτά έγιναν σε δίκτυα με διάφορα χαρακτηριστικά, αλλά το ενδιαφέρον μας επικεντρώνεται σε δίκτυα με μεγάλο γινόμενο bandwidth*delay.

Στο πρώτο και δεύτερο κεφάλαιο γίνεται μια περιγραφή των πρωτοκόλλων TCP και SCTP με έμφαση στο δεύτερο και στα ιδιαίτερα χαρακτηριστικά του, στους αλγόριθμους ελέγχου συμφόρησης και στο διαθέσιμο API για ανάπτυξη λογισμικού με χρήση του πρωτοκόλλου. Από το τρίτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα του πειραματικού μέρους τα οποία περιλαμβάνουν μεταφορά απλού αρχείου και δύο σενάρια μεταφοράς πολλαπλών αρχείων. Κλείνουμε με τα συμπεράσματα τις διπλωματικής εργασίας. Στα παραρτήματα μπορεί κανείς να βρει τον κώδικα της εφαρμογής καθώς και άλλα χρήσιμα εργαλεία που χρησιμοποιήθηκαν.

Κλείνοντας την σύντομη εισαγωγή, θα ήθελα να ευχαριστήσω θερμά τον σύμβουλό μου Καθηγητή Βασίλειο Μάγκλαρη αλλά και τον υποψήφιο διδάκτορα Βελένη Δημήτρη για την πολύτιμη βοήθεια και συμπαράσταση που μου έδωσαν κατά τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας.

1. ΤΟ ΠΡΩΤΟΚΟΛΛΟ TCP

Το TCP είναι ένα αξιόπιστο πρωτόκολλο μεταφοράς που χρησιμοποιείται κατά κόρο στο σημερινό διαδίκτυο. Μαζί με το UDP είναι τα ευρέως γνωστά πρωτόκολλα μεταφοράς σήμερα. Στο κεφάλαιο αυτό παρουσιάζουμε μια γενική περιγραφή του TCP και αναφερόμαστε εκτενέστερα στους αλγόριθμους συμφόρησης του και σε ορισμένες καινούριες τεχνολογίες που έχουν προστεθεί για την βελτίωση της απόδοση του.

1.1 Γενική περιγραφή Transport Control Protocol

Το TCP είναι το αξιόπιστο πρωτόκολλο μεταφοράς δεδομένων στο σημερινό διαδίκτυο. Για να εξασφαλίσει την αξιόπιστη μεταφορά δεδομένων το TCP βασίζεται σε διάφορες αρχές όπως την ανίχνευση λάθους, τις επαναληπτικές μεταδόσεις χαμένων πακέτων τις συνεχόμενες αναγνωρίσεις πακέτων, τα χρονόμετρα καθώς και τις επικεφαλίδες στα επιπρόσθετα πεδία των πακέτων.

Το TCP λέγεται ότι είναι στηριζόμενο σε σύνδεση γιατί μια εφαρμογή πριν ξεκινήσει να στέλνει πακέτα δεδομένων, πρέπει να έρθει σε συμφωνία με τον παραλήπτη, δηλαδή να στείλει προκαταρκτικά πακέτα ελέγχου τα οποία θα καθορίζουν κάποιες παραμέτρους του πρωτοκόλλου τα οποία θα εξασφαλίζουν την αξιόπιστη μεταφορά των δεδομένων. Το TCP είναι κατάλληλο για μεταφορά δεδομένων και προς τις δύο κατευθύνσεις. Επίσης η μεταφορά δεδομένων γίνεται μόνο μεταξύ δύο ακραίων κόμβων και όχι περισσοτέρων. Επίσης οι ενδιάμεσοι κόμβοι, δηλαδή οι δρομολογητές δεν γνωρίζουν καμιά πληροφορία για τις συνδέσεις αυτές.

Το πακέτο του TCP αποτελείται από δύο κύρια κομμάτια. Το πρώτο κομμάτι που συνήθως έχει μέγεθος 20 ψηφιολέξεις αποτελεί το μέρος των πεδίων της επικεφαλίδας. Τα κυριότερα από αυτά τα πεδία είναι η πηγή και ο προορισμός του πακέτου, ο μοναδικός κωδικός αριθμός του πακέτου και το μέγεθος του παραθύρου στον παραλήπτη. Το πεδίο αυτό χρησιμεύει για τον έλεγχο ροής δεδομένων, ώστε ο ρυθμός αποστολής των πακέτων να μειώνεται σε περίπτωση συμφόρησης στον παραλήπτη. Υπάρχουν και κάποια άλλα πεδία τα οποία δεν χρησιμεύουν πάντα όπως το πεδίο της χρονοσφραγίδας και του μέγιστου μεγέθους δεδομένων. Το δεύτερο κομμάτι αποτελεί το χώρο στον οποία θα τοποθετηθούν τα δεδομένα της εφαρμογής.

Όπως έχουμε αναφέρει το πρωτόκολλο αυτό είναι αξιόπιστο. Δηλαδή σε περίπτωση που έχουν χαθεί κάποια πακέτα, το πρωτόκολλο είναι αναγκασμένο να τα ξαναστείλει. Για να γνωρίζει πότε ένα πακέτο μπορεί να έχει χαθεί το πρωτόκολλο διατηρεί ένα χρονόμετρο του χρόνου μεταφοράς (round trip time RTT), δηλαδή του χρόνου που ένα πακέτο κάνει να φτάσει στον παραλήπτη και να έρθει πίσω η επιβεβαίωση στον αποστολέα. Εάν αυτό το χρονόμετρο λήξει, τότε υπάρχει η υπόνοια πως το πακέτο έχει χαθεί και γι' αυτό στέλνεται ξανά. Όποτε έχουμε επαναποστολή λόγω λήξης του χρονομέτρου, αρχικοποιούμε το χρονόμετρο αυτό στον διπλάσιο από τον προηγούμενο χρόνο.

Παρατηρούμε πως με την αύξηση του χρόνου στο χρονόμετρο, υπάρχει η πιθανότητα να χαθεί ένα πακέτο αλλά να περάσει αρκετή ώρα πριν αυτό εντοπιστεί. Για το σκοπό αυτό γίνεται χρήση του πεδίου του μοναδικού αύξοντα αριθμού στα πακέτα. Κάθε πακέτο που στέλνεται έχει ένα κωδικό αύξοντα αριθμό ο οποίος αυξάνεται κατά ένα με κάθε αποστολή πακέτων. Μόλις ένα πακέτο φτάσει στον προορισμό του, ο παραλήπτης φτιάχνει μια επιβεβαίωση παραλαβής την οποία και αποστέλλει πίσω στον αποδέκτη όπου πάνω υπάρχει ο μοναδικός κωδικός του πακέτου.

Εάν ένα πακέτο φτάσει στον παραλήπτη με ένα κωδικό αριθμό μεγαλύτερο από αυτό που θα περίμενε τότε αντί να στείλει επιβεβαίωση με τον αύξοντα αριθμό του πακέτου αυτού, στέλνει επιβεβαίωση με τον αύξοντα αριθμό του τελευταίου πακέτου που έφτασε στην σειρά. Αποτέλεσμα αυτής της διαδικασίας είναι, εάν ένα πακέτο έχει χαθεί, ο παραλήπτης να στέλνει επιβεβαιώσεις του τελευταίου πακέτου που έφτασε σε σειρά. Εάν ο αποστολέας πάρει τρεις διαδοχικές επιβεβαιώσεις του ιδίου πακέτου, τότε αναγνωρίζει πως το πακέτο έχει χαθεί και το ξαναστέλνει ακόμα κι αν το χρονόμετρο του χρόνου μεταφοράς δεν έχει λήξει. Η διαδικασία αυτή ονομάζεται γρήγορη επανεκκίνηση και είναι ένα από τα πλεονεκτήματα του πρωτοκόλλου αυτού.

Ο παραλήπτης κάθε σύνδεσης έχει ένα προσωρινό χώρο αποθήκευσης (buffer) στον οποίο αποθηκεύει τα πακέτα τα οποία έχουν φτάσει και βρίσκονται στη σωστή σειρά. Η εφαρμογή που σχετίζεται με την σύνδεση αυτή διαβάζει από αυτό τον αποθηκευτικό χώρο όποτε το θεωρήσει αυτή αναγκαίο και όχι με ένα σταθερό ρυθμό. Εάν ο ρυθμός αυτός είναι σχετικά αργός τότε ο αποθηκευτικός χώρος θα γεμίσει

σχετικά γρήγορα και τα πακέτα τα οποία θα φτάνουν από εκείνη την στιγμή και μετά θα ρίχνονταν αναγκαστικά από τον παραλήπτη.

Το TCP για να αποφύγει αυτό το πρόβλημα παρέχει ένα μηχανισμό ελέγχου ροής έτσι ώστε να εμποδίζει τον αποστολέα να στέλνει δεδομένα πιο γρήγορα από τον ρυθμό που ο παραλήπτης μπορεί να τα παραλαμβάνει. Για τον σκοπό αυτό ο παραλήπτης διατηρεί ένα παράθυρο το οποίο αντιπροσωπεύει τον ελεύθερο αποθηκευτικό χώρο που υπάρχει. Το μέγεθος αυτό στέλνεται στον αποστολέα με κάθε επιβεβαίωση που στέλνεται για πακέτα που έχουν παραληφθεί. Ο αποστολέας χρησιμοποιεί το δεδομένο αυτό ως εξής: Ο αποστολέας γνωρίζει το μέγεθος των δεδομένων που έχει στείλει στον παραλήπτη καθώς και το ποσοστό των δεδομένων αυτών που έχουν επιβεβαιωθεί ότι έφτασαν κανονικά στον παραλήπτη. Ο αποστολέας τότε γνωρίζει πως τα νέα δεδομένα που θα στείλει προστιθέμενα στα δεδομένα που δεν έχουν ακόμα επιβεβαιωθεί, δεν πρέπει να ξεπερνούν συνολικά τον ελεύθερο διαθέσιμο αποθηκευτικό χώρο του παραλήπτη.

Ο αλγόριθμος συμφόρησης του TCP χωρίζεται συνολικά σε τρεις φάσεις: Στην φάση της γραμμικής αύξησης και εκθετικής μείωσης, στην φάση της αργής εκκίνησης και στην φάση της αντίδρασης από γεγονότα λήξης χρόνου. Θα αναφερθούμε αναλυτικά σε αυτούς τους αλγόριθμους σε επόμενη παράγραφο.

Το TCP είναι κατασκευασμένο για να παρέχει δικαιοσύνη (fairness) ανάμεσα στις ροές του ιδίου τύπου. Δηλαδή εάν έχουμε ροές TCP που περνούν από το ίδιο μονοπάτι που έχει εύρος Χ και έχουμε Ψ ροές, τότε ο μέσος ρυθμός αποστολής δεδομένων για κάθε ροή θα είναι τελικά Χ/Ψ.

Το πρωτόκολλο TCP χρησιμοποιείται στο σημερινό διαδίκτυο γιατί παρέχει αξιόπιστη μεταφορά δεδομένων. Υπάρχουν πολλοί παράμετροι που ακόμα δεν έχει αξιοποιήσει αλλά το πρωτόκολλο αυτό συνεχίζει να εξελίσσεται ώστε να ανταποκρίνεται στις ανάγκες της σημερινής τεχνολογίας. Ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζει είναι ο συναγωνισμός με ροές διαφορετικού τύπου όπως το UDP που μεταφέρει πακέτα στα οποία δεν υπάρχει έλεγχος συμφόρησης.

1.2 Αλγόριθμοι ελέγχου συμφόρησης

Θα αναφερθούμε αναλυτικά στους τέσσερις αλγορίθμους ελέγχου συμφόρησης: αργή έναρξη, αποφυγή συμφόρησης, γρήγορη αναμετάδοση και γρήγορη αποκατάσταση [1].

Σε μερικές καταστάσεις μπορεί να είναι ευεργετικό για έναν αποστολέα TCP να είναι πιο συντηρητικός από ότι οι αλγόριθμοι επιτρέπουν, εντούτοις το TCP δεν μπορεί να είναι επιθετικότερο από ότι οι πιο κάτω αλγόριθμοι επιβάλουν (δηλαδή δεν μπορεί να στέλλει δεδομένα όταν η τιμή του cwnd, που υπολογίζεται από τους ακόλουθους αλγορίθμους δεν θα το επέτρεπε).

1.2.1 Ορισμοί

Πριν από την περιγραφή των αλγορίθμων δίνουμε τον ορισμό διάφορων όρων που θα είναι χρήσιμοι σε όλο το υπόλοιπο του παρόντος κεφαλαίου.

Segment: Οποιοδήποτε IP πακέτο μεταφέρει TCP δεδομένα ή / και επιβεβαιώσεις.

Sender Maximum Segment Size (SMSS): Μέγιστο μήκος ενός segment (σε bytes, χωρίς να περιλαμβάνει τους TCP/IP headers) που μπορεί να μεταδώσει ο αποστολέας. Συνήθως βασίζεται στο MTU του δικτύου στο οποίο συνδέεται ο αποστολέας.

Receiver Maximum Segment Size (RMSS): Μέγιστο μήκος ενός segment που ο παραλήπτης μπορεί να δεχτεί. Το στέλνει ο παραλήπτης μέσω του MSS TCP option κατά την εγκατάσταση της TCP σύνδεσης. Αν δε σταλεί τιμή MSS χρησιμοποιείται η default τιμή των 536 bytes. Η τιμή του RMSS χρησιμοποιείται ως άνω φράγμα για το SMSS.

Full-sized Segment: Ένα segment μήκους SMSS bytes.

Receiver Window (rwnd): Το πιο πρόσφατο μήκος παραθύρου που ανακοίνωσε ο παραλήπτης στον αποστολέα.

Congestion Window (cwnd): Το μήκος του παραθύρου που χρησιμοποιεί ο αποστολέας για τη μετάδοση των δεδομένων. Πρέπει πάντα cwnd <= rwnd.

Initial Window (IW): Η αρχική τιμή του cwnd (αμέσως μετά την εγκατάσταση της σύνδεσης).

Loss Window (LW): Το cwnd γίνεται ίσο με LW όταν ο αποστολέας αντιληφθεί απώλεια πακέτου με τη λήξη του retransmission timer.

Restart Window (RW): Το cwnd γίνεται ίσο με RW όταν ο αποστολέας ξεκινά τη μετάδοση μετά από περίοδο σιωπής.

Flight Size: Το πλήθος των δεδομένων (bytes) που έχουν μεταδοθεί αλλά δεν έχει (ακόμα) επιβεβαιωθεί η λήψη τους.

1.2.2 Αργή έναρξης και αποφυγή συμφόρησης

Οι αλγόριθμοι αργής έναρξης και αποφυγής συμφόρησης χρησιμοποιείται από τον αποστολέα TCP για να ελέγξει το ποσό των δεδομένων που στέλνονται στο δίκτυο. Για να εφαρμοστούν αυτοί οι αλγόριθμοι, δύο μεταβλητές χρησιμοποιούνται ανά-σύνδεσης TCP. Το παράθυρο συμφόρησης (cwnd) που είναι ένα όριο στη πλευρά του αποστολέα, στο ποσό δεδομένων που ο τελευταίος μπορεί να διαβιβάσει στο δίκτυο πριν λάβει μια αναγνώριση (ack), και το διαφημισμένο παράθυρο του δέκτη (rwnd) είναι ένα όριο στην πλευρά του δέκτη στο ποσό ανεπιβεβαίωτων δεδομένων που έχουν σταλεί. Το ελάχιστο του cwnd και rwnd κυβερνά τη μετάδοση δεδομένων.

Μια άλλη μεταβλητή, το κατώφλι αργής έναρξης (ssthresh), χρησιμοποιείται για να καθορίσει εάν χρησιμοποιείται ο αλγόριθμος αργής έναρξης ή αποφυγής συμφόρησης στη μετάδοση δεδομένων, όπως θα αναλύσουμε πιο κάτω.

Η αρχή της μετάδοσης σε ένα δίκτυο με άγνωστες συνθήκες απαιτεί από το TCP να εξετάσει το δίκτυο για να καθορίσει τη διαθέσιμη χωρητικότητα , έτσι ώστε να αποφύγει τη συμφόρηση που θα προκληθεί λόγο μεγάλης ροής δεδομένων. Ο αλγόριθμος αργής έναρξης χρησιμοποιείται για αυτόν το λόγο στην αρχή μιας μεταφοράς, ή κατόπιν επισκευής απώλειας που ανιχνεύεται από το χρονόμετρο αναμετάδοσης.

Η αρχική τιμή του cwnd, είναι λιγότερη ή ίση με 2*SMSS ψηφιολέξεις και όχι περισσότερη από 2 τμήματα (segments).
Σημειώνουμε ότι επεκτάσεις του TCP επιτρέπουν να χρησιμοποιεί ένα μεγαλύτερο αρχικό παράθυρο (IW).Με αυτήν την επέκταση, ένας αποστολέας TCP χρησιμοποιεί

3 ή 4 τμήματα αρχικό παράθυρο, υπό τον όρο ότι το συνδυασμένο μέγεθος των τμημάτων δεν υπερβαίνει 4380 ψηφιολέξεις.

Η αρχική τιμή του ssthresh μπορεί να είναι αυθαίρετα υψηλή (παραδείγματος χάριν, μερικές εφαρμογές χρησιμοποιούν το μέγεθος του διαφημισμένου παραθύρου), αλλά αυτό μπορεί να μειωθεί ανάλογα με τη συμφόρηση. Ο αλγόριθμος αργής έναρξης χρησιμοποιείται όταν cwnd < ssthresh, ενώ ο αλγόριθμος αποφυγής συμφόρησης χρησιμοποιείται όταν cwnd > ssthresh. Όταν cwnd και ssthresh είναι ίσα ο αποστολέας μπορεί να χρησιμοποιήσει είτε την αργή έναρξη είτε την αποφυγή συμφόρησης.

Κατά τη διάρκεια της αργής έναρξης, το TCP αυξάνει το cwnd το πολύ κατά SMSS ψηφιολέξεις για κάθε λαμβανόμενο ack που αναγνωρίζει τα νέα δεδομένα. Η αργή έναρξη τελειώνει όταν τον cwnd υπερβαίνει το ssthresh (ή, όταν το φθάνει, όπως σημειώνεται πιο πάνω) ή όταν παρατηρείται συμφόρηση.

Κατά τη διάρκεια της αποφυγής συμφόρησης, το cwnd αυξάνεται κατά 1 μέγεθος τμήματος ανά μετ' επιστροφής χρόνο (RTT). Η αποφυγή συμφόρησης συνεχίζεται έως ότου ανιχνευτεί συμφόρηση. Ένας τύπος που χρησιμοποιείται συνήθως στην αναπροσαρμογή του cwnd κατά τη διάρκεια της αποφυγής συμφόρησης δίνεται από την εξίσωση:

2 cwnd + = SMSS*SMSS/cwnd (1)

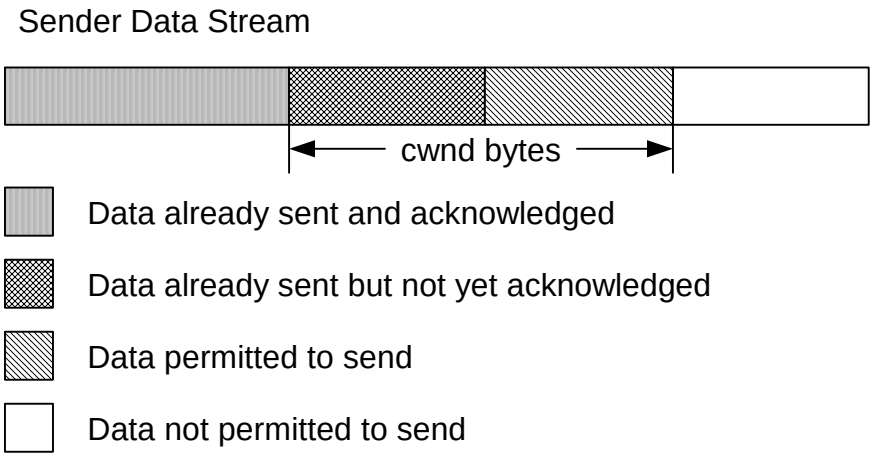
Αυτή η ρύθμιση εκτελείται σε κάθε εισερχόμενο μοναδικό ack. Η εξίσωση (1) παρέχει μια αποδεκτή προσέγγιση στην αρχή της αύξησης του cwnd κατά 1 μέγεθος τμήματος ανά RTT.

Όταν ένας αποστολέας TCP ανιχνεύει την απώλεια ενός τμήματος χρησιμοποιώντας το χρονόμετρο αναμετάδοσης, η τιμή του ssthresh τίθεται λιγότερη από την τιμή που δίνει η εξίσωση :

ssthresh = max (FlightSize/ 2, 2*SMSS) (2)

Το FlightSize είναι ο αριθμός δεδομένο που υπάρχουν μέσα το δίκτυο και άρα ανεπιβεβαίωτα.

Επιπλέον, μετά από μια διακοπή, το cwnd τίθεται λιγότερο από την παράθυρο απωλειών (LW), το οποίο είναι ίσο με 1 μέγεθος τμήματος (ανεξάρτητα από τη τιμή του IW). Επομένως, μετά από την αναμετάδοση του χαμένου τμήματος ο αποστολέας TCP χρησιμοποιεί τον αλγόριθμο αργής έναρξης για να αυξήσει το παράθυρο από 1 μέγεθος τμήματος στη νέα τιμή του ssthresh, όπου σε αυτό το σημείο θα αναλάβει και πάλι ο αλγόριθμος αποφυγής συμφόρησης.



Σχήμα 1: Παράθυρο συμφόρησης

1.2.3 Γρήγορη αναμετάδοση /Γρήγορη αποκατάσταση

Ένας δέκτης TCP στέλνει ένα άμεσο διπλό ack όταν φτάσει ένα εκτός σειράς τμήμα. Ο σκοπός αυτού του ack είναι να ενημερωθεί αποστολέας ότι ένα τμήμα έχει ληφθεί εκτός της σειράς σύμφωνα με τον αριθμό του.

Από την πλευρά του αποστολέα διπλά ACKs μπορούν να προκληθούν λόγο διαφόρων προβλημάτων στο δίκτυο.

Κατ' αρχάς, μπορούν να είναι προκληθούν από απορριφθέντα τμήματα. Σε αυτήν την περίπτωση, όλα τα τμήματα μετά από το χαμένο τμήμα θα προκαλέσουν διπλό ACK. Δεύτερον, διπλά ACKs μπορούν να προκληθούν από την ανακατάταξη των τμημάτων δεδομένων από το δίκτυο (όχι ένα σπάνιο γεγονός κατά μήκος μερικών διαδρομών μέσα στον δίκτύο). Τέλος, αντίγραφα ACKs μπορούν να προκληθούν από την αναπαραγωγή των τμημάτων ack ή δεδομένων από το δίκτυο.

Επιπλέον, ένας δέκτης TCP στέλνει ένα άμεσο ack όταν το εισερχόμενο τμήμα συμπληρώνει το σύνολο ή μέρος ένα κενού διαστήματος της ακολουθίας. Αυτό θα παραγάγει εγκυρότερες πληροφορίες για τον αποστολέα που επανέρχεται από μια απώλεια μέσω μιας διακοπής αναμετάδοσης, μιας γρήγορης αναμετάδοσης, ή ενός άλλου αλγόριθμου αποκατάστασης απώλειας, όπως ο NewReno.

Ο αποστολέας TCP χρησιμοποιεί τον αλγόριθμο γρήγορης αναμετάδοσης για να ανιχνεύσει και να διορθώσει πιθανές απώλειες, με βάση εισερχόμενα διπλά ACKs. Ο αλγόριθμος γρήγορης αναμετάδοσης χρησιμοποιεί την άφιξη 3 διπλών ACKs (4 ίδια ACKs χωρίς την άφιξη οποιωνδήποτε ενδιάμεσων πακέτων) σαν ένδειξη ότι ένα τμήμα έχει χαθεί. Μετά από την παραλαβή 3 διπλών ACKs, το TCP εκτελεί μια αναμετάδοση αυτού που εμφανίζεται να είναι το ελλείπον τμήμα, χωρίς να αναμένει το χρονόμετρο αναμετάδοσης να λήξει.

Αφότου ο αλγόριθμος γρήγορης αναμετάδοσης στέλνει αυτό που εμφανίζεται να είναι ελλείπον τμήμα, ο αλγόριθμος γρήγορης αποκατάστασης κυβερνά τη αποστολή νέων δεδομένων μέχρι να φτάσει ένα πρωτότυπο ack. Ο λόγος που δεν εφαρμόζεται ο αλγόριθμος αργής έναρξης είναι ότι η παραλαβή διπλών ACKs όχι μόνο δείχνει ότι ένα τμήμα έχει χαθεί, αλλά επίσης ότι αυτά τα τμήματα πιθανότατα έχουν φύγει από το δίκτυο (αν και ένας ογκώδης διπλασιασμός τμήματος από το δίκτυο μπορεί να ακυρώσει αυτό συμπέρασμα). Με άλλα λόγια, δεδομένου ότι ο δέκτης μπορεί να παραγάγει διπλό ack μόνο όταν φθάσει ένα τμήμα, εκείνο το τμήμα έχει αφήσει το δίκτυο και είναι στον buffer του δέκτη και άρα δεν καταναλώνει πλέων πόρους του δικτύου. Επιπλέον, αφού το ρολόι των ack συντηρείται, ο αποστολέας TCP μπορεί να συνεχίσει να διαβιβάζει νέα τμήματα (αν και η μετάδοση πρέπει να συνεχίσει χρησιμοποιώντας μειωμένη τιμή για το cwnd).

Οι αλγόριθμοι γρήγορης αναμετάδοσης και γρήγορης αποκατάστασης είναι συνήθως υλοποιημένοι μαζί ως εξής:

1. Όταν το τρίτο διπλό ack παραλαμβάνεται, θέτουμε το ssthresh σε όχι μεγαλύτερο από την αξία που δίνεται στην εξίσωση 2.

2. Αναμεταδίδουμε το χαμένο τμήμα και θέτουμε το cwnd στο ssthresh συν 3*SMSS. Αυτό τεχνητά "διογκώνει" το παράθυρο συμφόρησης κατά τον αριθμό από τα τμήματα (τρία) που έχουν αφήσει το δίκτυο και που ο δέκτης έχει αποθηκεύσει.
3. Για κάθε πρόσθετο διπλό ack που λαμβάνεται, αυξάνουμε το cwnd κατά SMSS. Αυτό διογκώνει τεχνητά το παράθυρο συμφόρησης με σκοπό να απεικονίσει το πρόσθετο τμήμα που έχει αφήσει το δίκτυο.
4. Στέλνουμε ένα τμήμα, εάν επιτρέπεται από τη νέα τιμή του cwnd και του διαφημιζόμενου παράθυρου του δέκτη.
5. Όταν φτάσει το επόμενο ACK που επιβεβαιώνει νέα δεδομένα θέτουμε το cwnd ίσο με ssthresh (η αξία που τίθεται στο βήμα 1). Αυτό καλείται "ξεφούσκωμα" του παράθυρου.
Αυτό το ack πρέπει να είναι η αναγνώριση αναμετάδοση από το βήμα 1, ένα RTT μετά από την αναμετάδοση (αν και μπορεί να φθάσει πιο σύντομα αν συμβεί σημαντική εκτός σειράς παράδοση τμημάτων δεδομένων στο δέκτη).Επιπλέον, αυτό το ack πρέπει να αναγνωρίσει όλα τα μεσάζοντα τμήματα που στάλισαν μεταξύ του χαμένου τμήματος και της παραλαβής του τρίτου διπλό ack, εάν κανένα από αυτά δεν χάθηκε.

1.3 Νέες τεχνολογίες για το TCP

Η απόδοση TCP εξαρτάται όχι από το ίδιο το ρυθμό μεταφοράς, αλλά μάλλον από το γινόμενο του ρυθμού μεταφοράς και της μετ' επιστροφής καθυστέρηση (RTT). Αυτό το "bandwidth * delay " μετρά το ποσό δεδομένων που "θα γέμιζε το σωλήνα", είναι ο χώρος απομονωτών που απαιτείται από τον αποστολέα και το δέκτη για να λάβουν τη μέγιστη ρυθμαπόδοση σε μια TCP σύνδεση σε μια διαδρομή, π.χ. ο αριθμός μη αναγνωρισμένων δεδομένων που το TCP πρέπει να χειριστεί προκειμένου να κρατηθεί γεμάτο το σύνολο του σωλήνα.

Τα προβλήματα απόδοσης του TCP προκύπτουν όταν το bandwidth * delay είναι μεγάλο. Αναφερόμαστε σε μια διαδρομή δικτύου που λειτουργεί σε αυτήν την περιοχή ως ένας "πολύ, παχύς σωλήνας ", και ένα δίκτυο που περιέχει αυτήν την διαδρομή ως "LFN" (προφέρεται σαν "elephan(t)").

Μεγάλης χωρητικότητας πακέτων δορυφορικά κανάλια (π.χ. , Δίκτυα Ευρείας ζώνης DARPA) είναι LFN. Παραδείγματος χάριν, ένα δορυφορικό κανάλι ταχύτητας DS1 έχει bandwidth * delay 10**6 bit ή περισσότερο, αυτό αντιστοιχεί σε 100 ανεπιβεβαίωτα τεμάχια TCP 1200 ψηφιολέξεων το κάθε ένα. Επίγεια με οπτική ίνα δίκτυα μπορούν να περιέλθουν επίσης στην κατηγορία LFN, για παράδειγμα, μια καθυστέρηση 30ms σε ένα εύρος ζώνης DS3 (45Mbps) δίκτυο επίσης υπερβαίνει τα 10**6 bits.

Υπάρχουν τρία θεμελιώδη προβλήματα απόδοσης με το TCP σε διαδρομές που χαρακτηρίζονται LFN [2]:

(1) Όριο μεγέθους παραθύρων
Η επικεφαλίδα του TCP χρησιμοποιεί έναν τομέα 16 bit για να αναφέρει μέγεθος παραθύρου δέκτη στον αποστολέα. Επομένως, το μεγαλύτερο παράθυρο που μπορεί να χρησιμοποιηθεί είναι 2**16 = 65000 ψηφιολέξεις.
Για να παρακάμψουμε αυτό το πρόβλημα, καθορίζεται μια νέα επιλογή TCP, "Κλίμακα παραθύρων", για να επιτρέψουμε παράθυρα μεγαλύτερα από 2**16. Αυτή η επιλογή καθορίζει έναν υπονοούμενο παράγοντα κλίμακας, ο οποίος χρησιμοποιείται για να πολλαπλασιάσει την αξία μεγέθους παραθύρου που βρίσκεται σε μια TCP επικεφαλίδα για να λάβει το αληθινό μέγεθος παραθύρου.

(2) Αποκατάσταση από τις απώλειες
Οι απώλειες πακέτων σε ένα LFN μπορούν να έχουν μια καταστροφική επίδραση στη ρυθμαπόδοση. Μερικές πιο παλιές υλοποιήσεις TCP θα ανάγκαζαν τη σωλήνωση δεδομένων να «στραγγίξει» με κάθε απώλεια πακέτων, απαιτούσαν μια δράση αργής-έναρξης για να ανακτήσουν. Σήμερα, οι αλγόριθμοι γρήγορης αναμετάδοσης και γρήγορης αποκατάστασης αναλαμβάνουν να λύσουν αυτά τα προβλήματα.

Η συνδυασμένη επίδραση τους είναι να ανακτήσει η σύνδεση από μια απώλεια πακέτων ανά παράθυρο, χωρίς να γίνει “άδειασμα” της σωλήνωσης. Εντούτοις, περισσότερο από μια απώλεια πακέτου ανά παράθυρο οδηγεί τυπικά σε διάλειμμα αναμετάδοσης και η σωλήνωση στραγγίζει και ξανά εφαρμόζει αργή έναρξη.

Επέκταση του μεγέθους παραθύρων για να ταιριάζει με την χωρητικότητα ενός LFN έχει ως αποτέλεσμα μια αντίστοιχη αύξηση της πιθανότητας περισσότερα από ένα πακέτα ανά παράθυρο να χαθούν. Αυτό θα μπορούσε να έχει μια καταστρεπτική επίδραση επάνω στη ρυθμαπόδοση του TCP πάνω από LFN.

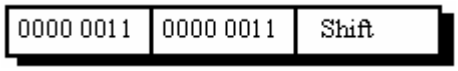
Για να γενικευθεί ο μηχανισμός αποκατάστασης γρήγορης αναμετάδοσης /γρήγορης αποκατάστασης για να μπορεί να χειρίζεται πολλαπλές απώλειες πακέτων ανά παράθυρο, απαιτούνται εκλεκτικά acknowledgments. Αντίθετα από τα κανονικά συσσωρευτικά acknowledgments του TCP, τα εκλεκτικά acknowledgments δίνουν στον αποστολέα μια πλήρης εικόνα του ποια τεμάχια που περιμένει στη σειρά ο δέκτης δεν έχουν φθάσει ακόμα.

Εντούτοις, στα μη- LFN δίκτυα, τα εκλεκτικά acknowledgments μειώνουν τον αριθμό τα πακέτα που αναμεταδίδονται αλλά δεν βελτιώνουν διαφορετικά την απόδοση, παρά μόνο εισάγουν πολυπλοκότητας κάνοντας αμφισβητήσιμη την αξίας τους.

(3) Μέτρηση της μετ' επιστροφής χρόνου
Το TCP υλοποιεί εγγυημένη παράδοση δεδομένων αναμεταδίδοντας τα τεμάχια που δεν έχουν επιβεβαιωθεί μέσα σε ένα συγκεκριμένο χρόνο, το χρόνο αναμετάδοσης (RTO). Ο ακριβής και δυναμικός καθορισμός μιας κατάλληλης τιμής για αυτό το χρονόμετρο είναι αναγκαίος για την καλά απόδοση του TCP. Ο RTO καθορίζεται υπολογίζοντας τη μέση τιμή και την διασπορά των μετ'επιστροφή μετρημένων χρόνων.

1.3.1 Επιλογή κλίμακας παραθύρων

Η επιλογή κλίμακας παραθύρων που είναι τρεις-ψηφιολέξεις μπορεί να σταλεί σε ένα τμήμα SYN από το TCP. Έχει δύο σκοπούς: (1) να δείξει ότι το TCP είναι έτοιμο να κάνει επιλογή κλίμακας παραθύρων δέκτη και αποστολέα, και (2) να ανταλλάξει τον παράγοντας κλίμακας που εφαρμόζεται στο παράθυρο παραλαβής. Κατά συνέπεια, ένα TCP που είναι έτοιμο να χρησιμοποιήσει παράθυρο κλίμακας πρέπει να στείλει την επιλογή, ακόμα κι αν ο παράγοντας κλίμακάς του είναι 1. Ο παράγοντας κλίμακας περιορίζεται σε δύναμη του δύο και είναι κωδικοποιημένος λογαριθμικά, έτσι ώστε να μπορεί να εφαρμοστεί με δυαδικές διαδικασίες μετατόπισης.



Σχήμα 2: Επιλογή κλίμακας παραθύρων TCP (WSopt)

Αυτή η επιλογή είναι μια προσφορά, όχι μια υπόσχεση και οι δύο πλευρές πρέπει να στείλουν τις επιλογές κλίμακας παραθύρων τους στα τμήματα SYN τους για να επιτραπεί η κλίμακα παραθύρου σε καθεμία κατεύθυνση.

Εάν η κλίμακα παραθύρων ενεργοποιηθεί, το TCP που έστειλε αυτήν την επιλογή θα κάνει δεξιά μετατόπιση του αληθινού του παραθύρου κατά "shift" θέσεις για τη μετάδοση μέσα στο SEG.WND. Η τιμή του shift μπορεί να είναι μηδέν (προσφέροντας τη κλίμακα, αλλά εφαρμόζοντας έναν παράγοντα κλίμακας 1 για λάβει το παράθυρο).

Αυτή η επιλογή μπορεί να σταλεί σε έναν αρχικό < SYN > τμήμα. Μπορεί επίσης να σταλεί με ένα < SYN,ACK > τμήμα, αλλά μόνο εάν η επιλογή κλίμακας παραθύρων παραλήφθηκε στο αρχικό < SYN > τμήμα. Μια επιλογή κλίμακας παραθύρων σε ένα τμήμα χωρίς κομμάτι SYN αγνοείται.

1.3.2 Βελτιώσεις στην αποκατάσταση απώλειας πακέτου

Η χρήση των αλγορίθμων Fast Retransmit & Fast Recovery για την αποκατάσταση απώλειας πακέτου δίνουν τη δυνατότητα στον αποστολέα να αναμεταδώσει κατά μέγιστο 1 χαμένο segment ανά RTT. Δεδομένου ότι σε χρόνο ίσο με 1 RTT είναι δυνατό να μεταδοθούν κατά μέγιστο segments ενός παραθύρου, η απώλεια περισσότερων του ενός segments στο ίδιο παράθυρο καθιστά τους αλγορίθμους Fast Retransmit & Fast Recovery αναποτελεσματικούς. Ο λόγος είναι ότι η αναμετάδοση n segments απαιτεί χρόνο n RTTs και επιπλέον κάθε RTT αποτελεί ξεχωριστή εκτέλεση του κύκλου Fast Retransmit & Fast Recovery με όλο και μικρότερο ssthresh.

Όπως αναφέραμε πάρα πάνω σε δίκτυα με υψηλό γινόμενο Bandwidth*Delay η αξιοποίηση του διαθέσιμου εύρους ζώνης προϋποθέτει τη χρήση μεγάλων παραθύρων. Με δεδομένο το ρυθμό σφαλμάτων και απωλειών, η χρήση μεγαλύτερων

τιμών παραθύρου αυξάνει την πιθανότητα περισσότερων της μιας απωλειών πακέτου στο ίδιο παράθυρο.
Παρουσιάζουμε στη συνέχεια δύο μεθόδους για τη βελτίωση του TCP ως προς την αποκατάσταση απώλειας πακέτου παρουσία περισσότερων της μιας απωλειών στο ίδιο παράθυρο.

1.3.2.1 New-Reno TCP

Το New-Reno TCP είναι μια παραλλαγή του Reno TCP που διαφοροποιείται μόνο στον τρόπο που τερματίζεται η φάση του Fast Recovery. Συγκεκριμένα εισάγεται η έννοια της «Μερικής Επιβεβαίωσης» (Partial ACK). Partial ACK ονομάζεται κάθε ACK που επιβεβαιώνει μέρος μόνο από τα δεδομένα που είχαν μεταδοθεί όταν ξεκίνησε το Fast Retransmit.

Στο Reno TCP η λήψη ενός partial ACK από τον αποστολέα τερματίζει το Fast Recovery. Στο New-Reno TCP το partial ACK εκλαμβάνεται ως ένδειξη ότι το επόμενο από αυτό που επιβεβαιώνεται segment επίσης χάθηκε και επομένως δεν λήγει η φάση του Fast Recovery. Με τη λήψη του partial ACK ο αποστολέας θα αναμεταδώσει αμέσως το ζητούμενο segment. Το Fast Recovery θα ολοκληρωθεί μόνο όταν επιβεβαιωθούν όλα τα δεδομένα που είχαν ήδη μεταδοθεί όταν ξεκίνησε το Fast Retransmit.

Το New Reno αναμεταδίδει 1 χαμένο segment ανά RTT όπως και το Reno TCP. Ωστόσο αποφεύγει τη διαδοχική μείωση στο μισό του ssthresh για κάθε ένα από τα χαμένα segments του ίδιου παραθύρου. Αυτό δεν μπορεί να το αποφύγει το Reno TCP, το οποίο για κάθε χαμένο segment στο ίδιο παράθυρο θα εκτελέσει ξεχωριστό κύκλο Fast Retransmit – Fast Recovery.

1.3.2.2 Selective Acknowledgements

Οι επιβεβαιώσεις στο TCP, όπως τουλάχιστον τις έχουμε δει μέχρι το σημείο αυτό, έχουν μια «συγκεντρωτική» (cumulative) φύση. Δεν επιβεβαιώνουν τη λήψη συγκεκριμένων segments αλλά ολόκληρου του data stream μέχρι ενός σημείου. Αν υπάρξει ένα κενό στο stream του παραλήπτη ο αποστολέας δεν θα πληροφορηθεί από τις επιβεβαιώσεις ότι segments «δεξιότερα» του κενού έχουν παραληφθεί.

Αυτή ακριβώς την πληροφορία επιχειρούν να μεταφέρουν τα Selective Acknowledgements (SACK). Τα SACKs δεν αντικαθιστούν τα cumulative ACKs αλλά δρουν συμπληρωματικά ως προς αυτά. Υλοποιούνται με τη χρήση 2 TCP options. Τα options αυτά είναι το SACK-permitted και το SACK. Το πρώτο ανταλλάσσεται κατά την εκκίνηση της σύνδεσης και χρησιμοποιείται για τη διαπραγμάτευση μεταξύ αποστολέα και παραλήπτη για τη χρήση ή όχι των SACKs. Το SACK option περιλαμβάνεται σε ACK segments και μεταφέρει την πληροφορία του Selective Acknowledgement. Πιο συγκεκριμένα ένα SACK option περιγράφει τμήματα του data stream που έχει λάβει ο παραλήπτης και που βρίσκονται «δεξιότερα» από το μεγαλύτερο επιβεβαιωμένο (με cumulative ACK) sequence number.

Η πληροφορία αυτή δίνει τη δυνατότητα στον αποστολέα να αναμεταδώσει χαμένα segments χωρίς να χρειάζεται να περιμένει 1 RTT για το καθένα από αυτά όπως συμβαίνει στο Reno και στο New-Reno TCP. Έτσι η χρήση των SACKs είναι δυνατό να επιταχύνει σημαντικά τη διαδικασία αποκατάστασης απώλειας πακέτων όταν έχουν χαθεί περισσότερα του ενός segments από το ίδιο παράθυρο.

2. ΤΟ ΠΡΩΤΟΚΟΛΛΟ SCTP

Το πρωτόκολλο SCTP [3] είναι ένα καινούριο πρωτόκολλο μεταφοράς που μπορεί να λειτουργεί σε ίδιο επίπεδο με το TCP στο οποίο αναφερθήκαμε στο πρώτο κεφάλαιο. Στο κεφάλαιο που ακολουθεί κάνουμε μια γενική περιγραφή του πρωτοκόλλου, αναφέρουμε τα ιδιαίτερα χαρακτηριστικά του και τις διαφορές του με το TCP, περιγράφουμε τους αλγόριθμους συμφόρησης που χρησιμοποιεί και περιγράφουμε το API που υπάρχει για προγραμματισμό διαφόρων εφαρμογών.

2.1 Γενική περιγραφή του SCTP

Το SCTP (Stream Control Transmission Protocol) είναι ένα νέο πρωτόκολλο μεταφοράς δεδομένων το οποίο αναπτύχθηκε από ερευνητές της βιομηχανίας και από πανεπιστήμια, με στόχο την μεταφορά τηλεπικοινωνιακών σημάτων σε δίκτυα βασιζόμενα στο IP. Είναι ένα πρωτόκολλο που λειτουργεί με την αρχή του παραθύρου, αξιόπιστο και με ένα μηχανισμό ελέγχου συμφόρησης παρόμοιο με αυτό του TCP. Έχει όμως πολλά νέα πλεονεκτήματα τα οποία το κάνουν πιο κατάλληλο για streaming media content.

Τα νέα αυτά πλεονεκτήματα που παρέχονται από το SCTP περιλαμβάνουν τα πολλαπλά streams ανάμεσα σε μια σύνδεση μεταξύ δύο άκρων, την ομαδοποίηση δεδομένων και έλεγχο πληροφορίας μαζί μέσα στο ίδιο πακέτο και ευελιξία στην σειρά παράδοσης των πακέτων (τα πακέτα δεν είναι απαραίτητο να παραδίδονται με την σωστή σειρά). Το SCTP υποθέτει πως δεν υπάρχει η δυνατότητα QoS (quality of service) στους δρομολογητές, ούτε AQM (active queue management) και ECN (explicit congestion notification). Μπορεί όμως να επωφεληθεί από την ύπαρξη τέτοιων μηχανισμών.

Το κύριο νέο στοιχείο που υλοποιεί το SCTP είναι η έννοια της συσχέτισης (association) η οποία είναι μια γενίκευση της σύνδεσης του TCP. Μια συσχέτιση ανάμεσα σε δύο SCTP άκρα είναι μια σύνδεση μεταξύ τους, όπου τα δεδομένα της εφαρμογής αλλά και τα δεδομένα ελέγχου, μπορούν να μεταδοθούν με ευελιξία και αξιοπιστία όσον αφορά την σειρά παράδοσης των πακέτων. Η έννοια της συσχέτισης στο SCTP δημιουργεί κύριες λειτουργίες οι οποίες κανονικά θα χρειάζονται από τις

streaming εφαρμογές και θα παρέχει ένα API για χρήση από την εφαρμογή ώστε να έχει πρόσβαση στις λειτουργίες αυτές.

Τα κύρια χαρακτηριστικά του SCTP είναι ότι τα πακέτα χωρίζονται σε τεμάχια (chunks), και κάθε τεμάχιο μπορεί να ανήκει σε διαφορετικό λογικό stream. Τα δεδομένα που περιέχει το κάθε τεμάχιο μπορεί να είναι διαφορετικών μεγεθών, φτάνει το ολικό μέγεθος του πακέτου να είναι μικρότερο ή ίσο με το μέγιστο ρυθμό μεταφοράς δεδομένων της σύνδεσης. Ορισμένα τεμάχια είναι προκαθορισμένα όπως τα τεμάχια αρχικοποίησης της σύνδεσης, μεταφοράς δεδομένων και μεταφοράς επιβεβαιώσεων παραλαβής. Ο αποστολέας έχει την δυνατότητα να κωδικοποιήσει πολλά τεμάχια σε ένα πακέτο και η αντίστοιχη αποκωδικοποίηση γίνεται στον παραλήπτη. Σε αυτή την διαδικασία υπάρχουν δύο βασικές παράμετροι που πρέπει να λαμβάνονται υπόψη, η αξιοπιστία και η σειρά στην παράδοση των δεδομένων.

Η αξιοπιστία καθορίζεται ως ο αριθμός των επαναλαμβανόμενων αποστολών πακέτων για πακέτα που έχουν χαθεί, όπως καθορίζεται από τον αποστολέα μετά από μελέτη των επιβεβαιώσεων παραλαβής πακέτων που στέλνει ο παραλήπτης. Μια αξιοπιστία βαθμού μηδέν σημαίνει πως δεν πρέπει να έχουμε επαναλαμβανόμενες αποστολές, ενώ μια αξιοπιστία βαθμού ένα σημαίνει πως πρέπει να επιχειρήσουμε το πολύ μία επαναλαμβανόμενη αποστολή κ.ο.κ.

Η σειρά στην παράδοση μπορεί να χρησιμοποιηθεί ώστε να αποφύγουμε το λεγόμενο head-of-line blocking το οποίο εμφανίζεται στο TCP. Εάν χρησιμοποιηθεί η σημαία για την παράδοση εκτός σειράς, το SCTP θα περάσει τα δεδομένα στην εφαρμογή με την σειρά με τα οποία τα έχει παραλάβει, αντί να περιμένει για επανάληψη της αποστολής πακέτων, εκεί που έχει διαπιστωθεί ένα κενό στην ακολουθία των αυξόντων αριθμών των πακέτων.

Ο μηχανισμός ελέγχου συμφόρησης του SCTP είναι σχεδόν ο ίδιος με αυτό του TCP New Reno. Κρατείται ένα παράθυρο συμφόρησης για κάθε συσχέτιση και ακολουθεί τον αλγόριθμο AIMD με τις φάσεις της αργής αρχής (slow start), της αποφυγής συμφόρησης (congestion avoidance), της γρήγορης αναμετάδοσης (fast retransmit) και της γρήγορης ανάκτησης (fast recovery). Αυτό οδηγεί σε καλή συμπεριφορά σε

σχέση με το TCP και οι ροές ανάμεσα σε SCTP και TCP λαμβάνουν δίκαιο ποσοστό όταν συναγωνίζονται για απόκτηση εύρους ζώνης στην ίδια σύνδεση.

Το SCTP είναι ένα νέο IP πρωτόκολλο μεταφοράς δεδομένων που υπάρχει σε παράλληλο επίπεδο με το UDP (User Datagram Protocol) και το TCP (Transmission Control Protocol), το οποίο παρέχει πολλές λειτουργίες μεταφοράς δεδομένων σε πολλές εφαρμογές διαδικτύου.

Όπως και το TCP το SCTP παρέχει ένα αξιόπιστο τρόπο μεταφοράς δεδομένων και εγγυάται ότι τα δεδομένα μεταφέρονται στο δίκτυο χωρίς σφάλματα και στην σωστή σειρά. Όπως το TCP είναι ένας μηχανισμός που στηρίζεται σε τμήματα (session-oriented) πράγμα που σημαίνει ότι δημιουργείται μια σχέση ανάμεσα στα άκρα των τμημάτων του SCTP πριν αρχίσουν να στέλνονται τα πακέτα των δεδομένων. Αυτή η σχέση διατηρείται μέχρι όλα τα δεδομένα να σταλούν με επιτυχία.

Σε αντίθεση με το TCP το SCTP παρέχει ένα αριθμό λειτουργιών που είναι κρίσιμες για μεταφορά τηλεφωνικών σημάτων και ταυτόχρονα μπορεί να ωφελήσει άλλες εφαρμογές που χρειάζονται τρόπο μεταφοράς με επιπρόσθετη απόδοση και αξιοπιστία.

Τα κύρια χαρακτηριστικά του SCTP είναι ότι είναι ένα πρωτόκολλο μονής ροής (unicast) και υποστηρίζει την ανταλλαγή δεδομένων ανάμεσα σε ακριβώς δύο ακραία σημεία, παρόλο που αυτά μπορούν να αναπαρασταθούν με πολλαπλές IP διευθύνσεις. Το SCTP παρέχει αξιόπιστη μεταφορά δεδομένων, και αναγνωρίζει πότε τα δεδομένα έχουν απορριφθεί, αλλάξει σειρά, διπλασιαστεί ή φθαρεί. Επίσης είναι σε θέση να μεταδώσει ξανά δεδομένα τα οποία δεν έχουν φτάσει στο παραλήπτη όπως τα έστειλε ο αποστολέας. Η αποστολή του SCTP είναι full duplex. Είναι επίσης προσαρμοσμένο με βάση τα μηνύματα (message oriented) σε αντίθεση με το TCP το οποίο είναι προσαρμοσμένο με βάση τα bytes (bytes oriented). Τέλος προσαρμόζει το ρυθμό αποστολής δεδομένων με παρόμοιο τρόπο με το TCP και μειώνει το ρυθμό αποστολής δεδομένων όσο αυξάνεται ο φόρτος του δικτύου. Είναι σχεδιασμένο να συμπεριφέρεται μη ανταγωνιστικά σε σχέση με το TCP προσπαθώντας να χρησιμοποιεί το ίδιο εύρος ζώνης (bandwidth).

Το όνομα του πρωτοκόλλου προέρχεται από την συνάρτηση multi-streaming που παρέχει το SCTP. Αυτή η συνάρτηση επιτρέπει στα δεδομένα να τεμαχιστούν σε πολλά μικρότερα κομμάτια και να τοποθετηθούν σε πολλαπλά streams που έχουν την ιδιότητα να παραδοθούν στον παραλήπτη σε ανεξάρτητη σειρά σε σχέση με την σειρά αποστολής. Επομένως εάν χαθεί κάποιο μήνυμα που βρίσκεται σε ένα stream θα επηρεάσει μόνο την παράδοση μέσα στο συγκεκριμένο stream.

Σε αντίθεση το TCP υποθέτει μόνο ένα stream δεδομένων και εγγυάται ότι η παράδοση σε αυτό το stream θα γίνει σε σειρά σε σχέση με την σειρά αποστολής των bytes μέσα στο stream. Καθώς αυτό είναι επιθυμητό για την παράδοση ενός αρχείου ή μιας σειράς δεδομένων, προκαλεί επιπρόσθετη καθυστέρηση όταν χάνονται μηνύματα ή δημιουργούνται σφάλματα στο δίκτυο. Εάν συμβεί αυτό το TCP πρέπει να καθυστερήσει την παράδοση των δεδομένων μέχρι να αποκατασταθεί η σωστή σειρά παράδοσης των δεδομένων, είτε με την παραλαβή πακέτου που είναι εκτός σειράς, είτε με την αναμετάδοση του συγκεκριμένου πακέτου που έχει χαθεί.

Για ορισμένες εφαρμογές το χαρακτηριστικό της παραλαβής πακέτων στην σωστή σειρά δεν είναι απόλυτα αναγκαίο. Στην μετάδοση τηλεφωνικών σημάτων είναι μόνο αναγκαίο να διατηρήσουμε την σειρά σε πακέτα που επηρεάζουν την ίδια πηγή, για παράδειγμα το ίδιο τηλεφώνημα ή το ίδιο κανάλι. Τα άλλα μηνύματα είναι ασθενώς συνδεδεμένα και μπορούν να παραδοθούν χωρίς να χρειάζεται να διατηρήσουμε την σειρά παράδοσης.

Ακόμη ένα παράδειγμα για την χρήση της συνάρτησης multi-streaming είναι η παράδοση κειμένων πολυμέσων, όπως οι ιστοσελίδες, όταν αυτό γίνεται σε ένα μόνο session. Αφού τα κείμενα πολυμέσων αποτελούνται από αντικείμενα από διαφορετικά μεγέθη και διαφορετικούς τύπους, το multi-streaming επιτρέπει την μεταφορά αυτών των αντικειμένων να είναι με μερική σειρά αντί με ολική σειρά και αυτό μπορεί να έχει ως αποτέλεσμα την απόδοση στην μεταφορά των δεδομένων.

Την ίδια στιγμή η μεταφορά γίνεται σε μια και μόνο συσχέτιση (association) έτσι ώστε όλα τα streams να είναι υποκείμενα σε μία και μόνο ροή και σε ένα μηχανισμό ελέγχου συμφόρησης με αποτέλεσμα να μειώνεται το επιπρόσθετο κόστος που χρειαζόταν στο επίπεδο μεταφοράς δεδομένων.

Το SCTP καταφέρνει να εφαρμόσει το μηχανισμό multi-streaming με την ύπαρξη ανεξαρτησίας ανάμεσα στην μετάδοση των δεδομένων και την παραλαβή των δεδομένων. Συγκεκριμένα κάθε πακέτο στο πρωτόκολλο χρησιμοποιεί δύο σύνολα αριθμών. Το πρώτο είναι για την μετάδοση των δεδομένων που ελέγχει την μετάδοση των μηνυμάτων και την ανίχνευση χαμένων μηνυμάτων, και το δεύτερο είναι για το ζεύγος stream id και stream sequence number το οποίο χρειάζεται για να καθορίσουμε την σειρά της παράδοσης των παραλαμβανόμενων δεδομένων.

Αυτή η ανεξαρτησία των μηχανισμών επιτρέπει στον παραλήπτη να καθορίσει αμέσως πότε προκύπτει ένα κενό στην σειρά παραλαβής των δεδομένων, καθώς και κατά πόσο τα μηνύματα που έχουν παραληφθεί μετά την ανίχνευση του κενού, ανήκουν στο επηρεαζόμενο stream. Εάν το μήνυμα ανήκει στο επηρεαζόμενο stream θα υπάρχει ένα κενό στον αύξοντα αριθμό του stream, ενώ εάν το μήνυμα ανήκει σε άλλο stream το κενό αυτό δεν θα φαίνεται. Επομένως ο παραλήπτης μπορεί να συνεχίσει να παραλαμβάνει μηνύματα από τα streams που δεν επηρεάζονται από το χάσιμο του μηνύματος και να αποθηκεύει τα μηνύματα από το επηρεαζόμενο stream μέχρι να παραλάβει το πακέτο που θα συμπληρώσει το κενό.

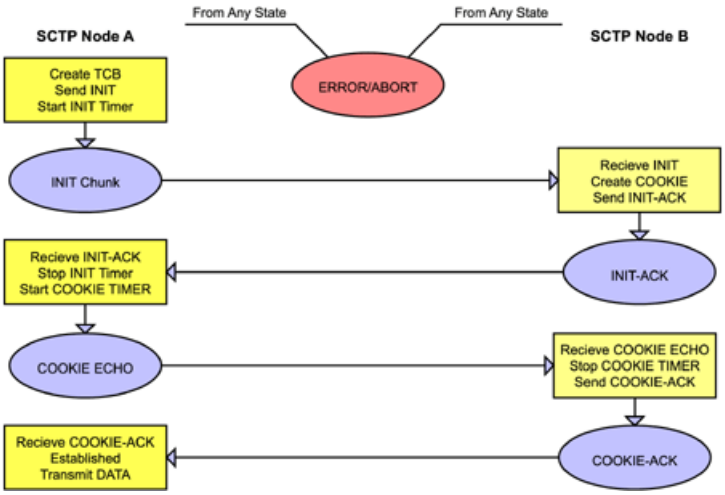
Ακόμα ένα χαρακτηριστικό του SCTP είναι το multi-homing ή αλλιώς η δυνατότητα για ένα ακραίο κόμβο να υποστηρίξει πολλαπλές IP διευθύνσεις. Το κέρδος από το multi-homing είναι ότι υπάρχει η πιθανότητα μια ροή (session) να συνεχίσει να υποστηρίζει την μεταφορά δεδομένων σε περίπτωση βλάβης του δικτύου. Σε μια κανονική ροή (session) μια βλάβη σε ένα τοπικό σημείο του δικτύου μπορεί να ισοπεδώσει την μεταφορά δεδομένων, ενώ μια βλάβη στον πυρήνα του δικτύου μπορεί να δημιουργήσει προσωρινή ανικανότητα στην μεταφορά δεδομένων μέχρι τα πρωτόκολλα να βρουν ένα εναλλακτικό μονοπάτι. Η χρήση του multi-homed SCTP χρησιμοποιείται για να υποστηρίξει την αρχική σύνδεση και η χρήση του γίνεται μόνο για επαναληπτικούς σκοπούς δηλαδή όταν έχουμε αποστολή πακέτων που δεν έχουν φτάσει στον προορισμό τους.

Μια διεύθυνση ορίζεται ως η κύρια διεύθυνση (primary address) και χρησιμοποιείται ως η διεύθυνση στην οποία όλα τα πακέτα θα αποστέλλονται. Τα πακέτα τα οποία στέλνονται ξανά γιατί δεν έχουν φτάσει στον προορισμό τους, χρησιμοποιούν τις

εναλλακτικές διευθύνσεις ώστε να βελτιώσουν τις πιθανότητες να φτάσουν στον τελικό προορισμό τους. Η ανικανότητα να σταλούν τα πακέτα από την κύρια διεύθυνση για κάποιο χρονικό διάστημα, έχει ως αποτέλεσμα όλα τα πακέτα να σταλούν από τις εναλλακτικές διευθύνσεις. Αυτό γίνεται μέχρι ο μηχανισμός heartbeat να μπορέσει να βρει και πάλι επαφή με την κύρια διεύθυνση.

Για να υποστηρίξουν το multi-homing τα άκρα του SCTP ανταλλάσσουν λίστες με τις διευθύνσεις κατά την αρχικοποίηση της σύνδεσης (session). Κάθε άκρο πρέπει να είναι σε θέση να λαμβάνει μηνύματα από οποιαδήποτε διεύθυνση που σχετίζεται με το άκρο αυτό. Για κάθε συγκεκριμένο session υπάρχει ένα port number που είναι μοναδικό ανάμεσα σε όλες τις διευθύνσεις του άκρου. Για να ενισχυθεί η ασφάλεια της σύνδεσης για κάθε νέο συμβάν που λαμβάνει χώρα, στέλνονται επιβεβαιώσεις στον παραλήπτη.

Η διαδικασία αρχικοποίησης βασίζεται σε μια σειρά ανταλλαγής τεσσάρων μηνυμάτων όπου τα δεδομένα αρχίζουν να ανταλλάσσονται κατά την αποστολή του τρίτου και τέταρτου μηνύματος, όταν δηλαδή η σύνδεση (association) έχει ήδη επικυρωθεί. Ένας μηχανισμός cookie έχει προστεθεί μέσα στην ακολουθία μηνυμάτων για να προστατεύσει από κάποιας μορφής επίθεσης στην σύνδεση. Ο μηχανισμός αυτός προστατεύει κυρίως από ένα τυφλό αποστολέα (blind attacker) ο οποίος στέλλει πακέτα αρχικοποίησης INIT με σκοπό να σπαταλήσει την μνήμη του SCTP εξυπηρετητή. Ο εξυπηρετητής όμως αντί να σπαταλά μνήμη κάθε φορά για ένα μπλοκ ελέγχου μετάβασης, δημιουργεί ένα cookie με τις πληροφορίες του μπλοκ και με μια διάρκεια ζωής και μια υπογραφή αυθεντικότητας την οποία θα στείλει μόλις πάρει πίσω το μήνυμα INIT ACK από τον παραλήπτη. Ο παραλήπτης όμως αφού είναι τυφλός δεν θα στείλει ποτέ το μήνυμα αυτό και ο εξυπηρετητής δεν θα σπαταλήσει επιπλέον μνήμη για να δημιουργήσει ένα νέο μπλοκ. Ένας κανονικός παραλήπτης SCTP θα πάρει το cookie και θα επιστρέψει ένα πακέτο επιβεβαίωσης της παραλαβής το οποίο θα χρησιμοποιηθεί από τον εξυπηρετητή για να αναδημιουργήσει το μπλοκ.



Σχήμα 3: Τετραπλή χειραψία κατά την αρχικοποίηση μια SCTP συσχέτισης.

Οι επαναληπτικές αποστολές πακέτων δεδομένων μπορούν να προκύψουν είτε από λήξη του χρονομέτρου επαναληπτικών αποστολών πακέτων είτε από την παραλαβή επιβεβαιώσεων SACK οι οποίες υποδεικνύουν ότι κάποια πακέτα δεδομένων δεν έχουν παραληφθεί κανονικά. Για να μειωθεί το ενδεχόμενο συμφόρησης ο ρυθμός των επαναληπτικών αποστολών είναι περιορισμένος. Το χρονόμετρο ρυθμίζεται με βάση τους υπολογισμούς για το μέσο χρόνο που χρειάζεται ένα πακέτο για να φτάσει στον προορισμό του.

Σε μια ενεργή σύνδεση με σχεδόν παρόμοιες αποστολές πακέτων για κάθε session οι επιβεβαιώσεις SACK είναι πιο πιθανή αιτία να προκαλέσουν επαναληπτικές αποστολές πακέτων από ότι η λήξη του χρονομέτρου. Για να μειώσουμε ακόμα περισσότερο την πιθανότητα να στείλουμε ξανά αλλά αχρείαστα ένα πακέτο, ένας κανόνας τεσσάρων SACK χρησιμοποιείται. Δηλαδή οι επαναληπτικές αποστολές πακέτων συμβαίνουν μόνο εφόσον ο αποστολέας έχει παραλάβει το τέταρτο SACK το οποίο υποδεικνύει ότι ένα πακέτο δεν έχει παραληφθεί. Ο κανόνας αυτός έχει σκοπό να αποκλείσει επαναληπτικές αποστολές οι οποίες θα συνέβαιναν επειδή κάποια πακέτα έχουν παραληφθεί εκτός σειράς.

Αρχικά διατηρούμε ένα αριθμό ο οποίος υποδεικνύει τον αριθμό των επαναληπτικών αποστολών πακέτων σε μια συγκεκριμένη διεύθυνση χωρίς να πάρουμε πίσω επιτυχώς κάποια επιβεβαίωση της αποστολής. Όταν αυτός ο αριθμός ξεπεράσει κάποιο αρχικά καθορισμένο μέγιστο αριθμό, η συγκεκριμένη διεύθυνση ορίζεται ως ανενεργή (inactive), γίνεται κάποια μορφής ειδοποίηση στην εφαρμογή και το SCTP

ξεκινά να χρησιμοποιεί μια εναλλακτική διεύθυνση για να μπορεί να αποστείλει τα πακέτα δεδομένων.

Επίσης πακέτα κτύπου καρδιάς (heartbeat) στέλνονται περιοδικά σε όλες τις μη χρησιμοποιούμενες διευθύνσεις (δηλαδή σε όλες τις εναλλακτικές διευθύνσεις), και ένας μετρητής διατηρείται στον αριθμό των πακέτων κτύπου καρδιάς που έχουν σταλεί στις διευθύνσεις αυτές χωρίς να έχουμε λάβει μια επιβεβαίωση κτύπου καρδιάς. Όταν αυτός ο μετρητής ξεπεράσει μια αρχικά καθορισμένη μέγιστη τιμή, οι διευθύνσεις αυτές ορίζονται ως ανενεργές.

Τα μηνύματα κτύπου καρδιάς συνεχίζονται να στέλνονται σε ανενεργές διευθύνσεις μέχρι να πάρουμε πίσω μια επιβεβαίωση. Εάν συμβεί κάτι τέτοιο η συγκεκριμένη διεύθυνση ορίζεται και πάλι ως ενεργή (active). Ο ρυθμός της αποστολής των πακέτων κτύπου καρδιάς είναι συνάρτηση του χρόνου αποστολής-παραλαβής πακέτων (round trip time RTO) και κάποιου επιπρόσθετου χρόνου για ταξινόμηση των δεδομένων.

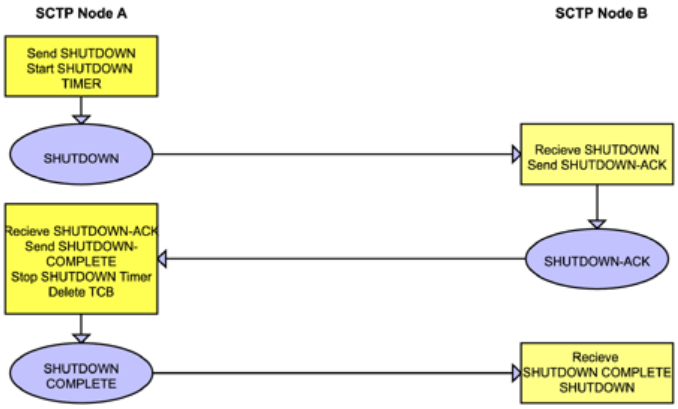
Παρόμοια με την αποτυχία μονοπατιού υπάρχει και η αποτυχία κόμβου. Ένας μετρητής διατηρείται για όλες τις πιθανές διευθύνσεις και αυξάνεται όταν δεν έχουμε επιβεβαίωση επαναληπτικής αποστολής πακέτου ή όταν ένα πακέτο κτύπου καρδιάς δεν πάρει επιβεβαίωση αποστολής. Όταν αυτός ο μετρητής ξεπεράσει μια αρχικά καθορισμένη μέγιστη τιμή το άκρο ορίζεται ως μη προσβάσιμο και η σύνδεση SCTP τερματίζεται.

Μετά από ανάλυση των ωφελημάτων που προκύπτουν, αποφασίστηκε η προσθήκη μιας μεθόδου μη αξιόπιστης μεταφοράς δεδομένων στο SCTP. Η σχεδίαση που προέκυψε επιτρέπει την συνύπαρξη αξιόπιστων και μη αξιόπιστων ροών μέσα σε μια σύνδεση (association) SCTP.
Κατά την αρχικοποίηση μιας σύνδεσης ο παραλήπτης περιλαμβάνει μια παράμετρο για να ενημερώσει τον αποστολέα ότι είναι σε θέση να υποστηρίξει μη αξιόπιστες ροές δεδομένων και ο αποστολέας υποδεικνύει ποιες ροές είναι αυτές

Το κλείσιμο του SCTP χρησιμοποιεί μια διαδικασία 3-μηνυμάτων για να επιτρέψει ένα ομαλό κλείσιμο ,όπου κάθε κόμβος έχει την επιβεβαίωση των τεμαχίων

δεδομένων λαμβανόμενων από τον απομακρυσμένο κόμβο πριν από την ολοκλήρωση του κλεισίματος. Μια ακυρωτική διαδικασία παρέχεται επίσης για τις περιπτώσεις λάθους όταν ένα άμεσο κλείσιμο πρέπει να πραγματοποιηθεί.

Το SCTP δεν υποστηρίζει τη λειτουργία "μιας μισάνοιχτης" σύνδεσης όπως μπορεί να εμφανιστεί στο TCP, όταν μια πλευρά δείχνει ότι δεν έχει άλλα στοιχεία να στείλει, αλλά η άλλη πλευρά μπορεί να συνεχίσει να στέλνει. Το SCTP υποθέτει ότι μόλις αρχίσει η διαδικασία κλεισίματος και οι δύο πλευρές θα σταματήσουν να στέλνουν νέα στοιχεία σε ολόκληρη την ένωση, και μόνο ανάγκη να τακτοποιηθούν οι αναγνωρίσεις των προηγουμένως σταλμένων στοιχείων υπάρχει.



Σχήμα 4: Ομαλός τερματισμός μιας SCTP συσχέτισης.

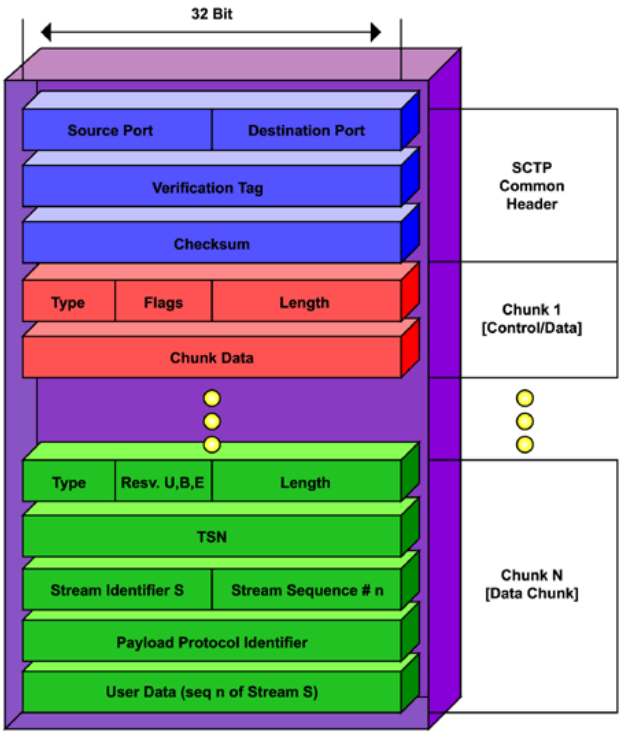
Το μήνυμα του SCTP περιλαμβάνει μια κοινή επικεφαλίδα συν ένα ή περισσότερα τεμάχια, τα όποια μπορούν να είναι τεμάχια ελέγχου ή δεδομένα. Η κοινή επικεφαλίδα έχει την θύρα πηγής και προορισμού για να επιτρέψει την πολυπλεξία διαφορετικών SCTP ενώσεων στην ίδια διεύθυνση, μια ετικέτα τριάντα δύο bit επαλήθευσης η οποία φρουρεί ενάντια στην εισαγωγή ενός ληγμένου ή ψεύτικου μηνύματος στην ένωση SCTP, και αθροιστικό έλεγχο τριάντα δύο bit(αυτό έχει τροποποιηθεί για να χρησιμοποιεί το πολυωνυμικό CRC-32c) για την ανίχνευση λάθους.

Κάθε τεμάχιο περιλαμβάνει τον τύπο τεμαχίου, πεδίο σημαίας, μήκος και αξία. Τα τεμάχια ελέγχου ενσωματώνουν τη διαφορετικές σημαίες και παραμέτρους ανάλογα με τον τύπο τεμαχίου. Τα τεμάχια δεδομένων ιδιαίτερα ενσωματώνουν τις σημαίες για έλεγχο της κατάτμησης και της επανασυναρμολόγησης, και παράμετρους για το

TSN, την ταυτότητα ρευμάτων και τον αριθμός ακολουθίας ρευμάτων, και ένα αριθμό πρωτοκόλλου ωφέλιμου φορτίου.

Η ταυτότητα πρωτοκόλλου ωφέλιμου φορτίου έχει περιληφθεί για μελλοντική ευελιξία. Προβλέπεται ότι οι λειτουργίες του προσδιορισμού και του πρωτοκόλλου φορτίου και της πολύπλεξης αριθμού θύρας δεν θα συνδέονται τόσο στενά στο μέλλον όπως είναι σήμερα.
Η ταυτότητα πρωτοκόλλου ωφέλιμου φορτίου θα επιτρέψει στο πρωτόκολλο που μεταφέρεται από το SCTP να είναι προσδιορισμένος ανεξάρτητος του αριθμού της πόρτας που χρησιμοποιείται.

Η μορφή μηνυμάτων του SCTP επιτρέπει φυσικά την υποστήριξη της συσσώρευσης πολλαπλών τεμαχίων δεδομένων και ελέγχου σε ένα ενιαίο μήνυμα, για να βελτιωθεί η αποδοτικότητα της μεταφοράς.



Σχήμα 5: Μορφή ενός πακέτου SCTP

2.2 Έλεγχος συμφόρησης SCTP

Ο έλεγχος συμφόρησης είναι μια από τις βασικές λειτουργίες στο SCTP. Για μερικές εφαρμογές, μπορεί να είναι πιθανό ότι επαρκείς πόροι θα είναι διατιθέμενοι στο SCTP για να βεβαιώσει τη γρήγορη παράδοση χρονικά κρίσιμων δεδομένων έτσι εμφανίζεται να είναι απίθανο, κατά τη διάρκεια των κανονικών διαδικασιών, ότι οι μεταδόσεις αντιμετωπίζουν αυστηρές συνθήκες συμφόρησης. Εντούτοις το SCTP πρέπει να λειτουργεί και υπό δυσμενείς συνθήκες λειτουργίας.

Σε τέτοιες καταστάσεις το SCTP πρέπει να ακολουθεί τα σωστά βήματα ελέγχου συμφόρησης για να επανέρχεται από τη συμφόρηση προκειμένου να παραδοθούν όσο πιο γρήγορα γίνεται τα δεδομένα. Ελλείψει της συμφόρησης δικτύου, αυτοί οι προληπτικοί αλγόριθμοι ελέγχου συμφόρησης δεν πρέπει να παρουσιάζουν κανέναν αντίκτυπο στην απόδοση του πρωτοκόλλου.

Μια εφαρμογή επιτρέπεται πάντα να υιοθετήσει περισσότερο συντηρητικούς αλγόριθμους ελέγχου συμφόρησης από αυτός που αναφέρονται κατωτέρω.

Οι αλγόριθμοι ελέγχου συμφόρησης που χρησιμοποιούνται από SCTP είναι βασισμένοι στο [RFC2581]. Θα απαριθμήσουμε αρχικά τις διαφορές στη σχεδίαση του πρωτοκόλλου του TCP και του SCTP, και θα περιγράψουμε έπειτα το σχέδιο ελέγχου συμφόρησης του SCTP. Η περιγραφή χρησιμοποιεί την ίδια ορολογία όπως στον έλεγχο συμφόρησης TCP.

Υπενθυμίζουμε ότι ο έλεγχος συμφόρησης SCTP εφαρμόζεται πάντα σε ολόκληρη την ένωση και όχι στα μεμονωμένα ρεύματα.

2.2.1 Διαφορές στο SCTP από τον έλεγχο συμφόρησης του TCP

Τα κενά κομμάτια επιβεβαίωσης στο SCTP φέρνουν την ίδια σημασιολογική έννοια όπως τα SACK του TCP. Το TCP λαμβάνει υπόψη τις πληροφορίες που φέρονται τις επιβεβαιώσεις σαν συμβουλευτικές πληροφορίες μόνο. Το SCTP λαμβάνει υπόψη τις πληροφορίες που φέρονται μέσα στα κενά τμήματα των SACK ως συμβουλευτικές. Στο SCTP, οποιαδήποτε τεμάχια δεδομένων που έχει αναγνωριστεί από τα SACK,

συμπεριλαμβανομένων των δεδομένων που έφθασαν στον παραλήπτη εκτός σειράς, δεν θεωρούνται πλήρως παραδοθείς μέχρι το συσσωρευτικό σημείο των TSN ack ξεπερνά το TSN του τεμαχίου δεδομένων.

Συνεπώς, η τιμή του cwnd ελέγχει τον αριθμό τον δεδομένων που δεν έχουν αναγνωριστεί, παρά (όπως στην περίπτωση του TCP χωρίς SACK) το ανώτερο όριο μεταξύ του μεγαλύτερου αναγνωρισμένου αριθμού ακολουθίας και το πιο πρόσφατο τμήμα δεδομένων που μπορεί να σταλεί μέσα στο παράθυρο συμφόρησης.
Τα SACK του SCTP οδηγούν σε διαφορετικές υλοποιήσεις γρήγορης αναμετάδοσης και γρήγορης αποκατάστασης από το TCP χωρίς SACK.

Η μεγαλύτερη διαφορά μεταξύ SCTP και του TCP, εντούτοις, είναι το multi-homing. Το SCTP είναι σχεδιασμένο να δημιουργεί ανθεκτική επικοινωνία μεταξύ δύο σημείων κάθε ένα από τα οποία μπορεί να απαντά σε περισσότερες από μια διευθύνσεις μεταφοράς. Οι ενδεχομένως διαφορετικές διευθύνσεις μπορούν οδηγήσουν σε διαφορετικές πορείες στο δίκτυο μεταξύ των δύο σημείων, κατά συνέπεια ιδανικά χρειάζεται ένα χωριστό σύνολο παραμέτρων ελέγχου συμφόρησης για κάθε μια από τις διαδρομές.

Οι τρέχοντες αλγόριθμοι ελέγχου συμφόρησης για mutli-homing κάνουν τις ακόλουθες υποθέσεις:

Ο αποστολέας χρησιμοποιεί συνήθως την ίδια διεύθυνση προορισμού μέχρι να υπάρξει άλλη εντολή από το ανώτερο στρώμα, το SCTP μπορεί να αλλάξει σε έναν εναλλακτικό προορισμό όταν μια διεύθυνση χαρακτηριστεί ανενεργή. Επίσης, το SCTP μπορεί να αναμεταδώσει σε διαφορετική διεύθυνση μεταφοράς από την αρχική.

Ο αποστολέας κρατά μια ξεχωριστή παράμετρο ελέγχου συμφόρησης καθορισμένη για κάθε ένας από τους προορισμούς που μπορεί να στείλει (όχι για κάθε ένα ζευγάρι πηγή-προορισμού αλλά ένα για κάθε προορισμό). Οι παράμετροι πρέπει να λήγουν εάν η διεύθυνση δεν χρησιμοποιείται για μια αρκετά μεγάλη χρονική περίοδο.

Για κάθε διεύθυνση προορισμού, ένα σημείο εκτελεί αργή έναρξη κατά την πρώτη μετάδοση σε εκείνη την διεύθυνση.

Να σημειώσουμε ότι το TCP εγγυάται τη παράδοση με σωστή σειρά των δεδομένων στο πρωτόκολλο ανώτερου-στρώματος μέσα σε μια ενιαία σύνοδο TCP. Αυτό σημαίνει ότι όταν το TCP παρατηρεί ένα χάσμα στο λαμβανόμενο αριθμό ακολουθίας, περιμένει μέχρι το χάσμα να γεμίσει πριν παραδώσει τα δεδομένα που παραλήφθηκαν με αριθμοί υψηλότεροι από αυτό του ελλείποντος δεδομένου. Από την άλλη πλευρά, το SCTP μπορεί να παραδώσει τα δεδομένα στο πρωτόκολλο ανώτερου στρώματός ακόμα κι αν υπάρχει χάσμα σε TSN εάν οι αριθμοί ακολουθίας ρευμάτων είναι στη σειρά για το συγκεκριμένο ρεύμα (π.χ., τα ελλείποντα τεμάχια δεδομένων είναι για έναν διαφορετικό stream) ή εάν έχει ζητηθεί χωρίς σωστή σειρά παράδοση. Αν και αυτό δεν έχει επιπτώσεις στη τιμή του cwnd, μπορεί να επηρεάσει τον υπολογισμό του rwnd.

2.2.2 Αργή έναρξη και αποφυγή συμφόρησης στο SCTP

Οι αλγόριθμοι αργής έναρξης και αποφυγής συμφόρησης χρησιμοποιούνται από ένα τερματικό σημείο για να ελέγχει τον αριθμό των δεδομένων που εισέρχεται στο δίκτυο. Ο έλεγχος συμφόρησης στο SCTP υιοθετείται όσον αφορά μια ένωση, και όχι σε ένα μεμονωμένο ρεύμα. Σε μερικές καταστάσεις μπορεί να είναι καλύτερα για έναν αποστολέα SCTP να είναι πιο συντηρητικός από αυτό που οι αλγόριθμοι επιτρέπουν εντούτοις, ένας αποστολέας SCTP δεν μπορεί να είναι πιο επιθετικός από τους ακόλουθους αλγορίθμους.

Όπως και το TCP, ένα τερματικό σημείο SCTP χρησιμοποιεί τις ακόλουθες τρεις μεταβλητές ελέγχου για να καθορίσει το ρυθμό μετάδοσής του.

Το διαφημισμένο μέγεθος παραθύρου του δέκτη (rwnd, σε ψηφιολέξεις), το οποίο τίθεται από το δέκτη βασισμένο στο διαθέσιμο χώρο απομονωτών για εισερχόμενα πακέτα. Αυτή η μεταβλητή διατηρείται σε ολόκληρη την ένωση.

Το παράθυρο ελέγχου συμφόρησης (cwnd, σε ψηφιολέξεις), ρυθμίζεται από τον αποστολέα βασισμένο στην κίνηση του δικτύου. Αυτή η μεταβλητή διατηρείται αν διεύθυνση προορισμού.

Το κατώφλι αργής έναρξης (ssthresh, σε ψηφιολέξεις), το όποιο χρησιμοποιείται από τον αποστολέα για να διακρίνει τις φάσεις αργής αρχής και αποφυγής συμφόρησης. Αυτή η μεταβλητή διατηρείται αν διεύθυνση προορισμού.

Το SCTP απαιτεί επίσης μια πρόσθετη μεταβλητή ελέγχου την partial_bytes_acked ,η όποια χρησιμοποιείται κατά τη διάρκεια της φάσης αποφυγής συμφόρησης για να διευκολύνει τη ρύθμιση του cwnd.

Αντίθετα από το TCP, ένας αποστολέας SCTP κρατά ένα σύνολο αυτών των μεταβλητών ελέγχου cwnd, ssthresh για κάθε διεύθυνση προορισμού του δέκτη του (όταν είναι multι-homed). Μόνο ένα rwnd κρατείται για ολόκληρη την ένωση.

2.2.2.1 Αργή έναρξη

Η έναρξη μετάδοσης δεδομένων σε ένα δίκτυο με άγνωστες συνθήκες μετά από μια αρκετά μεγάλη περίοδο αδράνειας απαιτεί από το SCTP για να εξετάσει το δίκτυο για να καθορίσει τη διαθέσιμη ικανότητα. Ο αλγόριθμος αργής έναρξης χρησιμοποιείται για αυτόν το λόγο όπως και στο TCP στην αρχή μιας μεταφοράς, ή μετά από τη διόρθωση απώλειας που ανιχνεύεται από το χρονόμετρο αναμετάδοσης.

Η αρχική τιμή του cwnd πριν από τη μετάδοση δεδομένων ή μετά από αρκετά μεγάλη αδρανή περίοδος ΠΡΕΠΕΙ να είναι <= 2*MTU.

Η αρχική τιμή του cwnd μετά από ένα διάλειμμα αναμετάδοσης δεν είναι περισσότερη από 1*MTU.

Η αρχική τιμή του ssthresh μπορεί να είναι αυθαίρετα υψηλή.

Όποτε η τιμή του cwnd είναι μεγαλύτερη από μηδέν, το τερματικό σημείο επιτρέπεται

να έχει cwnd ψηφιολέξεις μη επιβεβαιωμένων δεδομένων στη συγκεκριμένη διεύθυνση μεταφοράς.

Όταν ένας κόμβος δεν διαβιβάζει δεδομένα σε μια δεδομένη διεύθυνση μεταφοράς το cwnd της διεύθυνσης μεταφοράς πρέπει να προσαρμοστεί σε:
max(cwnd/2, 2*MTU) ανά RTO.

2.2.2.2 Αποφυγή συμφόρησης

Όταν η τιμή του cwnd είναι μεγαλύτερη από το ssthresh, το cwnd πρέπει να αυξηθεί κατά 1*MTU ανά RTT εάν ο αποστολέας έχει cwnd ή περισσότερες ψηφιολέξεις δεδομένων που εκκρεμούν για την αντίστοιχη διεύθυνση μεταφοράς.
Στην πράξη μια εφαρμογή μπορεί να επιτύχει αυτόν τον στόχο με τον ακόλουθο τρόπο:

Η τιμή του partial_bytes_acked αρχικοποιείται σε 0.

Όταν το cwnd είναι μεγαλύτερο από το ssthresh, σε κάθε άφιξη SACK που προωθεί το συσσωρευτικό σημείο TSN ack, αυξάνεται κατά το συνολικό αριθμό ψηφιολέξεων όλων των νέων τεμαχίων που αναγνωρίζονται σε εκείνο το SACK συμπεριλαμβανομένων των τεμαχίων που αναγνωρίζονται από το νέο συσσωρευτικό TSN ack και από το κενό κομμάτι του ack.

Όταν το partial_bytes_acked είναι ίσο με ή μεγαλύτερο από cwnd και πριν από την άφιξη του SACK ο αποστολέας είχε για cwnd περισσότερες ψηφιολέξεις εκκρεμόντων δεδομένων, αύξηση του cwnd κατά ένα MTU, και αναθεώρηση του partial_bytes_acked σε partial_bytes_acked-cwnd.

Το ίδιο με την αργή έναρξη, όταν ο αποστολέας δεν διαβιβάζει δεδομένα σε μια δεδομένη διεύθυνση μεταφοράς, το cwnd της διεύθυνσης μεταφοράς προσαρμόζεται στο max(cwnd/ 2, 2*MTU) ανά RTO.

Όταν όλα το δεδομένα που διαβιβάζονται από τον αποστολέα έχουν αναγνωριστεί από το δέκτη, το partial_bytes_acked μηδενίζεται.

2.2.2.3 Έλεγχος συμφόρησης

Κατά την ανίχνευση απωλειών πακέτων από το SACK ο κόμβος κάνει τα εξής:

ssthresh = max(cwnd/2, 2*MTU)

cwnd = ssthresh

Βασικά μια απώλεια πακέτου προκαλεί το cwnd να πέσει στο μισό.

Όταν το χρονόμετρο T3-rtx λήξει σε μια διεύθυνση, το SCTP εκτελεί αλγόριθμο αργής έναρξης με:

ssthresh = max(cwnd/2, 2*MTU)

cwnd = 1*MTU

και βεβαιώνεται ότι μόνο ένα πακέτο SCTP θα είναι στη διαδρομή για εκείνη τη διεύθυνση μέχρι το σημείο τέλους λάβει την αναγνώριση για επιτυχή παράδοση των δεδομένων σε εκείνη την διεύθυνση.

2.2.2.4 Γρήγορη αναμετάδοση στις εκθέσεις χάσματος

Ελλείψει της απώλειας δεδομένων, ένα σημείο τέλους εκτελεί καθυστερημένη αναγνώριση. Εντούτοις, όποτε ένα σημείο τέλους παρατηρεί μια τρύπα στην άφιξης ακολουθίας TSN, αρχίζει αποστολή SACK κάθε φορά που ένα πακέτο φθάνει φέρνοντας δεδομένα έως ότου γεμίσουν την "τρύπα".

Όποτε ένα σημείο τέλους λαμβάνει έναν SACK που δείχνει κάποια απώλεια TSN, περιμένει 3 περαιτέρω ενδείξεις απώλειας (μέσω επόμενου SACK) στο ίδιο TSN πριν να λάβει μέτρα γρήγορης αναμετάδοσης.

Όταν το TSN αναφέρεται όπως ελλείπον στο τέταρτο διαδοχικό SACK, ο αποστολέας δεδομένων κάνει τις εξής κινήσεις:

1) Χαρακτηρίζει το ελλείπον τεμάχιο δεδομένων προς αναμετάδοση,

2) Ρυθμίζει το ssthresh και το cwnd της διεύθυνσης προορισμού της οποίας ανήκουν τα τελευταία ελλείποντα τεμάχια δεδομένων, σύμφωνα με τον τύπο που περιγράφεται στην παράγραφο ελέγχου συμφόρησης,

3) Καθορίζει πόσα από τα πιο πρόσφατα τεμάχια δεδομένων που χαρακτηρίστηκαν για αναμετάδοση θα χωρέσουν ενιαίο πακέτο, σύμφωνα με τον περιορισμό του MTU διαδρομής της διεύθυνσης μεταφοράς προορισμού στην οποία το πακέτο στέλνεται,

4) Ξανά ξεκινά το χρονόμετρο T3-rtx μόνο εάν το τελευταίο SACK αναγνώρισε το χαμηλότερο αριθμό TSN που στέλνεται σε εκείνη την διεύθυνση, ή ο κόμβος αναμεταδίδει το πρώτο ανεπιβεβαίωτο τεμάχιο δεδομένων που στάλθηκε σε αυτή τη διεύθυνση.

Πριν από τις ανωτέρω ρυθμίσεις, εάν το λαμβανόμενο SACK επίσης αναγνωρίζει τα νέα τεμάχια δεδομένων και προωθεί το συσσωρευτικό TSN ack, πρέπει να εφαρμοστούν πρώτα οι κανόνες ρύθμισης του cwnd που καθορίζονται στις παραγράφους αργής έναρξης και αποφυγής συμφόρησης.

2.3 Το API του πρωτοκόλλου SCTP

Για την ευκολία μετάβασης από το TCP στο SCTP έχει σχεδιαστεί ένα API βασισμένο σε sockets [4]. Μια τέτοια υλοποίηση χρησιμοποιήσαμε για τον προγραμματισμό της εφαρμογής μας σε SCTP.

Το API υποστηρίζει δύο διεπαφές:
1) Ένα σε πολλά
Το σκεπτικό είναι παρόμοιο με αυτό που ορίζεται για πρωτόκολλα χωρίς σύνδεση σαν το UDP. Ένα socket SCTP τύπου ένα- σε -πολλά είναι ικανό να ελέγχει πολλές συσχετίσεις SCTP. Αυτό είναι παρόμοιο με socket UDP το οποίο μπορεί να επικοινωνήσει με πολλούς δέκτες. Σε κάθε ένα από αυτές τις συσχετίσεις δίνεται ένας μοναδικός αριθμός ταυτότητας για να ξεχωρίζουν. Υπενθυμίζουμε εδώ ότι το SCTP είναι πρωτόκολλο με σύνδεση και δεν υποστηρίζει εκπομπές σε όλους ή σε πολλούς δέκτες όπως το UDP.

2) Ένα προς ένα

Αυτή η διεπαφή ακολουθεί σκεπτικό παρόμοιο με αυτό ενός TCP socket που είναι πρωτόκολλο με σύνδεση. Ελέγχει μόνο μία συσχέτιση (association). Ο σημαντικότερος λόγος που καθορίστηκε αυτή η διεπαφή είναι για να επιτρέψει σε εφαρμογές γραμμένες πάνω από πρωτόκολλα με σύνδεση να τροποποιηθούν σε SCTP χωρίς μεγάλες δυσκολίες, καθώς και για να διευκολυνθούν προγραμματιστές με εμπειρία σε πρωτόκολλα με σύνδεση.

Ακόμα ένας λόγος τέτοιας διεπαφής είναι να εξασφαλιστεί ότι μηχανισμοί διαφόρων λειτουργικών όπως την select() που έχουν να κάνουν με socket θα συνεχίσουν να δουλεύουν .

Για την υλοποίηση της εφαρμογής μας χρησιμοποιήσαμε τον δεύτερο τρόπο καθώς ήταν πιο προσιτός προγραμματιστικά και δεν είχε κανένα μειονέκτημα σε σχέση με τον πρώτο. Ακόμα κάνει πιο εύκολη τη σύγκριση μεταξύ TCP και SCTP αφού προγραμματίζονται και με κοινή λογική.

Ένας τυπικός εξυπηρετητής τύπου ένα- προς -ένα θα καλέσει τις κλήσεις συστήματος που ακολουθούν για να ετοιμάσει ένα κόμβο SCTP που να μπορεί να εξυπηρετεί κλήσεις.

Πίνακας 1: Κλήσεις τύπου TCP διεπαφής για ετοιμασία εξυπηρετητή

α/α	Κλήση	Ορισμός
1	socket ()	Η κλήση socket() δημιουργεί ένα περιγραφέα ενός socket που αντιστοιχεί στον συγκεκριμένο SCTP κόμβο.
2	bind()	Η κλήση bind() χρησιμοποιείται για να διαγράψει την αρχική διεύθυνση που συσχετίζεται με το συγκεκριμένο SCTP τερματικό.
3	listen()	Η κλήση listen() ετοιμάζει ένα SCTP τερματικό να δεχθεί εισερχόμενους συνδέσμους(associations).
4	accept()	Η κλήση accept() μπλοκάρει τον κόμβο μέχρι μια δημιουργηθεί μία νέα συσχέτιση όπου και επιστέφεται ένας νέος περιγραφέας socket.

5	close()	Η κλήση close() καλείται για να τερματιστεί μια συσχέτιση.

Καθώς ένας εξυπηρετητής καλεί τις πιο πάνω κλήσεις, ένας πελάτης από την άλλη πλευρά πρέπει να καλέσει τις κλήσεις που ακολουθούν για να εγκαταστήσει μια σύνδεση.

Πίνακας 2: Κλήσεις τύπου TCP διεπαφής για πελάτη

α/α	Κλήση	Ορισμός
1	socket ()	Η κλήση socket() δημιουργεί ένα περιγραφέα ενός socket που αντιστοιχεί στον συγκεκριμένο SCTP κόμβο.
2	connect()	Η κλήση connect() χρησιμοποιείται για να ξεκινήσει μια συσχέτιση μεταξύ ενός απομακρυσμένου κόμβου.
3	sendmsg()	Η κλήση sendmsg() για να στείλουμε μια αίτηση στον εξυπηρετητή.
4	recvmsg()	Η κλήση recvmsg() για να λάβουμε ένα μήνυμα από τον εξυπηρετητή.
5	close()	Η κλήση close() καλείται για να τερματιστεί μια συσχέτιση.

3. ΕΦΑΡΜΟΓΗ ΜΕΤΑΦΟΡΑΣ ΑΡΧΕΙΩΝ

Για να προχωρήσουμε στο πειραματικό μέρος της έρευνας μας και να μπορέσουμε να πάρουμε μετρήσεις για την απόδοση των πρωτοκόλλων χρησιμοποιήσαμε μια εφαρμογή ανταλλαγής αρχείων (FTP) που να δουλεύει αντίστοιχα πάνω από τα δύο πρωτόκολλα.

Για να είναι αντικειμενική η σύγκριση των πρωτοκόλλων, προγραμματίσαμε ένα εξυπηρετητή (server) και ένα πελάτη (client) για να γίνεται ανταλλαγή αρχείων σύμφωνα με το πρωτόκολλο FTP [5]. Τα ίδια αυτά προγράμματα τροποποιήσαμε κάνοντας χρήση του SCTP API που περιγράψαμε σε προηγούμενη παράγραφο και συγκεκριμένα τις διεπαφής ένα προς ένα για να δουλεύουν πάνω από το πρωτόκολλο SCTP.

Και στις δύο περιπτώσεις ο εξυπηρετητής ακούει σε μια θύρα την οποία δίνει ο χρήστης στην γραμμή εντολών όταν τον βάζει σε λειτουργία. Ακολούθως ο πελάτης μπορεί να ενωθεί με τον εξυπηρετητή και να χρησιμοποιήσει διάφορες εντολές από το πρωτόκολλο FTP.

Για πρακτικούς λόγους υλοποιήσαμε μόνο εντολές που είναι χρήσιμες για τους σκοπούς εκτέλεσης των πειραμάτων μας, καθώς και μερικές βοηθητικές εντολές για την γενική λειτουργικότητα του προγράμματος. Συγκεκριμένα στα προγράμματα υπάρχουν για χρήση οι εντολές:

open	-- αίτηση για ένωση με συγκεκριμένο εξυπηρετητή FTP.
Exit	-- τερματισμός κελύφους προγράμματος πελάτη.
help	-- εμφάνιση διαθέσιμων εντολών και οδηγίες χρήσης.
close	-- τερματισμός σύνδεσης με τον εξυπηρετητή.
user	-- εγγραφή στον εξυπηρετητή με όνομα και κωδικό.
pwd	-- εμφάνιση τρέχοντος καταλόγου στον εξυπηρετητή.
cwd	-- αλλαγή τρέχοντος καταλόγου στον εξυπηρετητή
list	-- εμφάνιση λίστας αρχείων στον εξυπηρετητή
get	-- “κατέβασμα” αρχείου από τον εξυπηρετητή.

Μερικές από τις πιο πάνω εντολές έχουν ορίσματα τα οποία παρουσιάζουμε ολοκληρωμένα στον πίνακα που ακολουθεί.

Πίνακας 3: Ορίσματα διαθέσιμων εντολών

Εντολή	Ορίσματα
open	<host-name> <port/service>
exit	-
help	-
close	-
user	<username> <password>
pwd	-
list	<path>
cwd	<path>
get	<file name>

Όταν ο πελάτης εκτελέσει την εντολή open και ενωθεί με τον εξυπηρετητή ανταλλάσσει τις κατάλληλες εντολές, όπως περιγράφονται στο RFC του πρωτοκόλλου FTP από την σύνδεση που έχει δημιουργηθεί την οποία θα αποκαλούμε σύνδεση εντολών.

Σύμφωνα με το πρωτόκολλο FTP η ανταλλαγή δεδομένων και συγκεκριμένα τα αποτελέσματα της εντολής LIST και της εντολής GET δηλαδή η λίστα αρχείων τρέχοντος καταλόγου και όποια αρχεία ζητήσει ο χρήστης, μεταφέρονται από μια ξεχωριστή σύνδεση.

Όταν ο χρήστης εκτελέσει μια από αυτές τις εντολές, γίνονται διαφανή από τον χρήστη η απαραίτητες ενέργειες για να επιτευχθεί μια νέα σύνδεση μεταφοράς δεδομένων ανάμεσα σε εξυπηρετητή και πελάτη. Στην υλοποίηση αυτή ο χρήστης είναι αυτός που ακούει σε κάποια θύρα την οποία ανακοινώνει στον εξυπηρετητή για να κάνει την σύνδεση. Η συνεννόηση αυτή γίνεται με την εσωτερική εντολή PORT σύμφωνα με το πρωτόκολλο μεταφοράς αρχείων.

Τα δύο προγράμματα υποθέτουν μεταφορά δυαδικών αρχείων [7], και το αρχείο στον χρήστη μεταφέρεται στον τρέχον κατάλογο και με ονομασία ίδια με αυτή του αρχείου που ζητείται.

```
FTP Shell started --- Enter command (see help) ...

FTP> open 10.1.1.2 12345
220 Connected to Florides FTP server.
FTP> user ftp florides
230 Guest login ok, access restrictions apply.
FTP> help
FTP Shell    Version 1.0  Available commands are :

exit            --  terminates current subshell.
help            --  prints list of available commands.
open            -- connect to the specified ftp host.
close           -- close current connection with a server.
user            -- login to the server.
pwd             -- prints current working directory.
cwd             -- change working directory.
list            -- list files in specified directory.
get             -- download file from server.

FTP> get file
reveive buffer size = 800000
PORT 10,1,1,1,240,39
200 PORT command successful.
150 Opening data connection.
connection from 10.1.1.2:52280
reveive buffer size = 800744
FTP> ---- STATUS file---- Rate:0.02 MBytes/sec ----
---- STATUS file---- Rate:0.01 MBytes/sec ----
---- STATUS file---- Rate:0.01 MBytes/sec ----
```

Σχήμα 6: Πρόγραμμα πελάτη

Για την διεξαγωγή των πειραμάτων απαιτείται μεταφορά πολλών αρχείων παράλληλα και συγκεκριμένα ανάλογα με το πρωτόκολλο μεταφοράς που χρησιμοποιείται υλοποιήθηκαν οι παρακάτω δυνατότητες στα αντίστοιχα προγράμματα:

Πρωτόκολλο μεταφοράς TCP: κάθε εντολή get δημιουργεί μια καινούρια διεργασία στον πελάτη και στον εξυπηρετητή έτσι ώστε το αρχείο που μεταφέρεται να εκτελείται στο παρασκήνιο και ο χρήστης να μπορεί να εκτελεί άλλες εντολές στο προσκήνιο (προφανώς με συνεχόμενες εντολές get έχουμε παράλληλη μεταφορά αρχείων). Υπενθυμίζουμε ότι η μεταφορά αρχείου γίνεται σε ανεξάρτητη σύνδεση TCP από την σύνδεση εντολών και συνεπώς κάθε καινούρια εντολή get θα δημιουργεί άλλη σύνδεση TCP. Συνοψίζοντας, όταν ο χρήστης κάνει παράλληλη μεταφορά αρχείων κάθε ένα από αυτά μεταφέρεται από δική του σύνδεση TCP και σύμφωνα με το πρωτόκολλο TCP υπόκειται σε ξεχωριστό έλεγχο συμφόρησης.

Πρωτόκολλο μεταφοράς SCTP: Σε αυτή την περίπτωση όταν ο χρήστης επιθυμεί παράλληλη μεταφορά αρχείων δίνει την εντολή get <file 1> <file 2> <file 3> κ.τ.λ. Τότε ο εξυπηρετητής δημιουργεί όσα νήματα όσα είναι και τα αρχεία που ζητήθηκαν, και αναθέτει σε κάθε νήμα την εγγραφή ενός αρχείου. Στο πρωτόκολλο SCTP όμως δεν δημιουργούμε καινούρια σύνδεση ή συσχέτιση (association) όπως κάνουμε στο TCP αλλά γράφουμε τα αρχεία στην ίδια συσχέτιση αλλά σε διαφορετική ροή (stream). Ο πελάτης διαβάζει τα δεδομένα από την σύνδεση δεδομένων και ανάλογα από πια ροή έρχονται τα αποθηκεύει στο κατάλληλο αρχείο.

Σε αυτό το σημείο πρέπει να σημειώσουμε μια προγραμματιστική διαφορά στη κάθε υλοποίηση. Στο πρόγραμμα που δουλεύει σε TCP γίνεται χρήση καινούριων διεργασιών (κλήση fork()) για την « παραλληλοποίηση » ενώ στο αντίστοιχο σε SCTP γίνεται χρήση νημάτων (threads). Ο λόγος αυτής της διαφοροποίησης είναι ότι στην τρέχουσα βιβλιοθήκη SCTP δεν κληρονομείται η υπάρχουσα συσχέτιση ανάμεσα στις διεργασίες παιδιά και διεργασία πατέρα και κατά συνέπεια δεν μπορούν να γράφουν όλες στην ίδια συσχέτιση. Με την χρήση νημάτων το πρόβλημα ξεπερνιέται αφού όλα τα νήματα έχουν κοινή συσχέτιση. Θεωρητικά η δημιουργία νημάτων είναι μια πιο γρήγορη διαδικασία για το λειτουργικό από την δημιουργία νέας διεργασίας αλλά αυτό δεν επηρεάζει την απόδοση στις μετρήσεις που θα πραγματοποιήσουμε.

Με την χρήση των προγραμμάτων αυτών θέλουμε να δούμε πώς επηρεάζεται η απόδοση της μεταφοράς δεδομένων όταν γίνετε η κάθε μεταφορά σε ανεξάρτητη σύνδεση που υπόκειται σε ξεχωριστό έλεγχο συμφόρησης, και όταν όλες οι μεταφορές ελέγχονται από ένα κοινό αλγόριθμο συμφόρησης. Αυτό εξηγεί στις σχεδιαστικές επιλογές που αναφέραμε πιο πάνω.

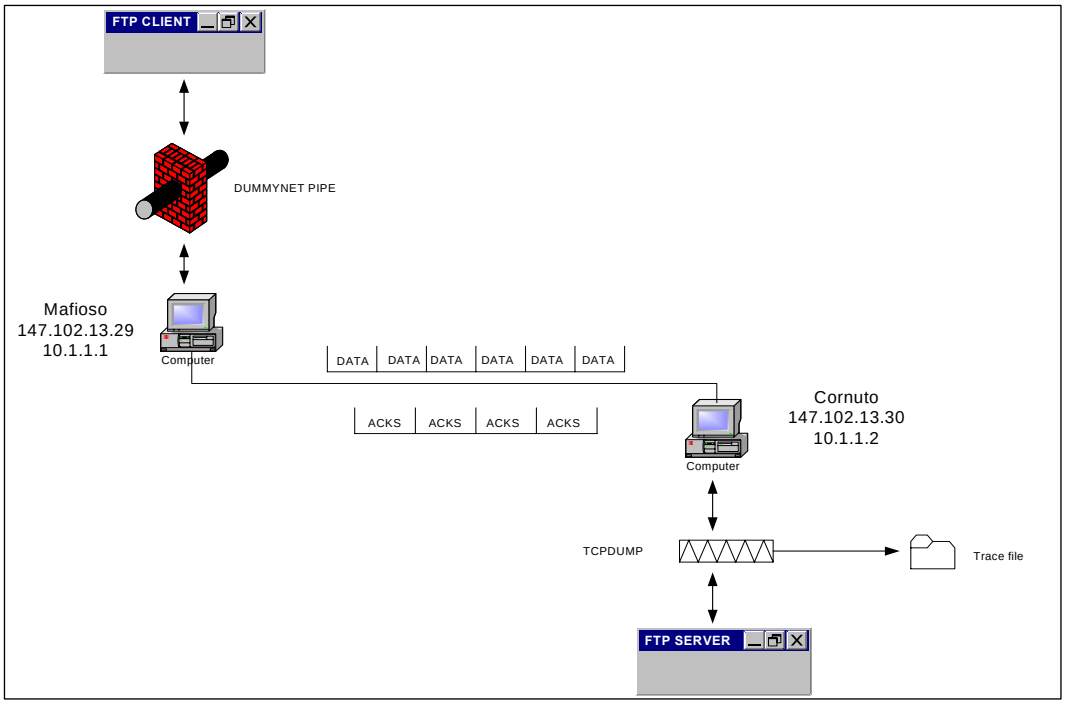
4. ΜΕΤΡΗΣΕΙΣ ΣΕ ΜΕΤΑΦΟΡΑ ΕΝΟΣ ΑΡΧΕΙΟΥ

Η υλοποίηση των εφαρμογών που περιγράψαμε στο προηγούμενο κεφάλαιο θα μας εξυπηρετήσει στην διεξαγωγή πραγματικών πειραμάτων με μεταφορές αρχείων πάνω από τα δύο πρωτόκολλα (TCP και SCTP), για να μπορέσουμε να δούμε αλλά και να συγκρίνουμε την λειτουργία τους. Το πρώτο μέρος των μετρήσεων περιλαμβάνει μεταφορά ενός και μόνο αρχείου και την σύγκριση της απόδοσης στα δύο πρωτόκολλα για να δείξουμε ότι στο SCTP μπορεί να χρησιμοποιηθεί και στις τρέχουσες εφαρμογές που χρησιμοποιούν TCP.

4.1 Τοπολογία δικτύου πειραμάτων

Οι μετρήσεις και τα πειράματα αυτού του κεφαλαίου έχουν πραγματοποιηθεί πάνω από πραγματικό δίκτυο υπολογιστών με την βοήθεια και μερικών άλλων προγραμμάτων τα οποία θα περιγράψουμε στη συνέχεια.

Το δίκτυο πάνω στο οποίο μεταφέρουμε τα δεδομένα αποτελείται από δύο υπολογιστές συνδεδεμένους μεταξύ τους στους οποίους τρέχουν ο εξυπηρετητής και ο πελάτης αντίστοιχα.



Σχήμα 7: Δομή δικτύου πειραμάτων

Στον υπολογιστή Mafioso τρέχει η εφαρμογή του πελάτη καθώς και το εργαλείο DUMMYNET. Το εργαλείο αυτό εξομοιώνει ή και επιβάλλει ουρές με περιορισμούς στο εύρος ζώνης, με καθυστερήσεις, απώλειες πακέτων και φαινόμενα πολλαπλών διαδρομών. Δουλεύει παρεμβαίνοντας στα πακέτα κατά την διαδρομή τους στις διάφορες στοίβες πρωτοκόλλων και αναγκάζοντας τα να περάσουν μέσα από τις λεγόμενες σωληνώσεις ή ουρές που εξομοιώνουν διάφορες συνθήκες δικτύων ανάλογα με τα χαρακτηριστικά τους. Σημαντικό χαρακτηριστικό αυτού του εργαλείου είναι ότι η οποιοιδήποτε επεξεργασία γίνεται από τον πυρήνα του λειτουργικού και ο πρόσθετος χρόνος που απαιτείται είναι πολύ μικρός.

Στον υπολογιστή Cornuto τρέχει η εφαρμογή του εξυπηρετητή και το πρόγραμμα TCPDUMP. Το τελευταίο είναι πρόγραμμα καταγραφής πακέτων, το οποίο έχει δυνατότητα να αναγνωρίζει και πακέτα του πρωτοκόλλου SCTP. Οποιοδήποτε πακέτο περνά από το πρόγραμμα αυτό το οποίο πληρεί τις προϋποθέσεις που έχει δώσει ο χρήστης καταγράφεται σε αρχείο.

Όταν γίνεται μια μεταφορά αρχείου δημιουργείται το αρχείο με τις επικεφαλίδες όλων των πακέτων που κινήθηκαν πάνω στο δίκτυο. Από το αρχείο αυτό μπορεί κανείς να πάρει όλες τις πληροφορίες που χρειάζεται για την λειτουργία των εμπλεκόμενων πρωτοκόλλων. Για παράδειγμα μπορεί κανείς να μελετήσει ένα διάγραμμα της χρονικής ανταλλαγής πακέτων και να παρατηρήσει πότε υπήρχαν απώλειες πακέτων, πότε αυτά τα πακέτα αναμεταδόθηκαν, την συχνότητα αποστολής πακέτων, την λειτουργία του πρωτοκόλλου όπως τα πακέτα αρχικοποίησης μιας συνόδου και πολλά άλλα. Όσο αφορά την απόδοση ενός πρωτοκόλλου μπορούμε να κάνουμε διαγράμματα της ρυθμαπόδοσης που είναι το καλύτερο μέτρο σύγκρισης.

Ακόμα για να μπορέσουμε να μελετήσουμε τους αλγόριθμους συμφόρησης και τον τρόπο που τα πρωτόκολλα αυτά αντιδρούν σε απώλειες πακέτων είναι πολύ χρήσιμο να έχουμε ένα διάγραμμα του παραθύρου συμφόρησης. Επειδή η πληροφορία αυτή χρησιμοποιείται εσωτερικά κατά την λειτουργία του πρωτοκόλλου στον αποστολέα και δεν υπάρχει καταγραμμένη στα πακέτα δεν υπάρχει άμεσος τρόπος να έχουμε αυτό το διάγραμμα. Η πιο καλή προσέγγιση αυτού του διαγράμματος είναι ο υπολογισμός των ανεπιβεβαίωτων πακέτων που βρίσκονται στο δίκτυο. Αν σε μια

σύνδεση υπάρχει συνεχής ροή δεδομένων από την μια πλευρά, όπως συμβαίνει και στην περίπτωση της δικής μας εφαρμογής, τότε ο αριθμός αυτός αποτελεί μια ενδεικτική τιμή του παραθύρου συμφόρησης του αποστολέα.

4.2 Αναλυτικά παραδείγματα μεταφοράς ενός αρχείου

Στην πρώτη φάση των πειραμάτων παρουσιάζουμε τα σημαντικότερα από τα διαγράμματα που αναφέραμε παραπάνω για μετρήσεις που έγιναν κατά την ανταλλαγή ενός και μόνο αρχείου πάνω από τα πρωτόκολλα TCP και SCTP. Σε κάθε περίπτωση με την βοήθεια του εργαλείου DUMMYNET αλλάζαμε τις συνθήκες του δικτύου έτσι ώστε να δούμε την συμπεριφορά των πρωτοκόλλων σε δίκτυα με μικρή ή μεγάλη καθυστέρηση, πολλές ή λίγες απώλειες πακέτων, μεγάλο ή μικρό εύρος ζώνης. Τα διαγράμματα είναι για ενδεικτικές περιπτώσεις μόνο. Μια γενική σύγκριση θα γίνει σε επόμενη παράγραφο με βάση την ρυθμαπόδοση σε κάθε περίπτωση.

4.2.1 Παράδειγμα με Bandwidth:1 Mbit/s ,Delay:1ms,Errors:0%

Στο πρώτο παράδειγμα εξομοιώνουμε ένα δίκτυο με χαρακτηριστικά:

Bandwidth: 1 Mbit/s

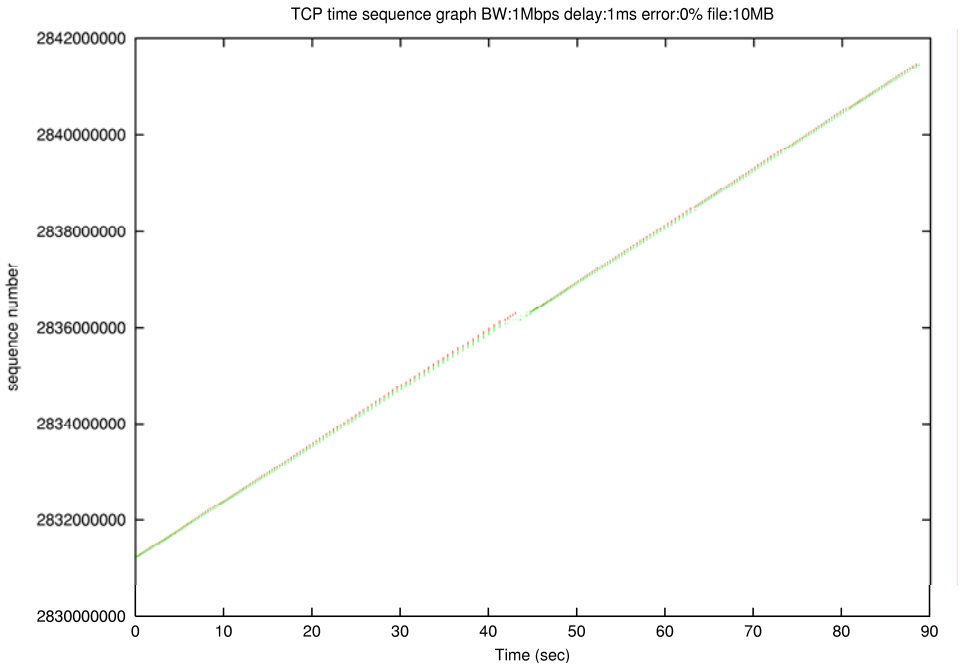
Delay: 1ms

Errors: 0%

Το αρχείο που χρησιμοποιήσαμε για την μεταφορά είναι μεγέθους 10MB.

Σε όλα τα παραδείγματα ο χρόνος καθυστέρησης δίνεται ανά κατεύθυνση, που σημαίνει ότι ο χρόνος μετ’ επιστροφής είναι ο διπλάσιος. Ακόμα η τιμή για τα λάθη είναι αυτή που προσθέτουμε επιπλέον της φυσικής. Επειδή οι μετρήσεις γίνονται σε πραγματικό δίκτυο υπάρχουν τα λάθη που προέρχονται από το φυσικό δίκτυο και δεν μπορούμε να τα ελέγξουμε. Τα λάθη αυτά βρέθηκαν να έχουν μια μέγιστη τάξη του 10^{-3} .

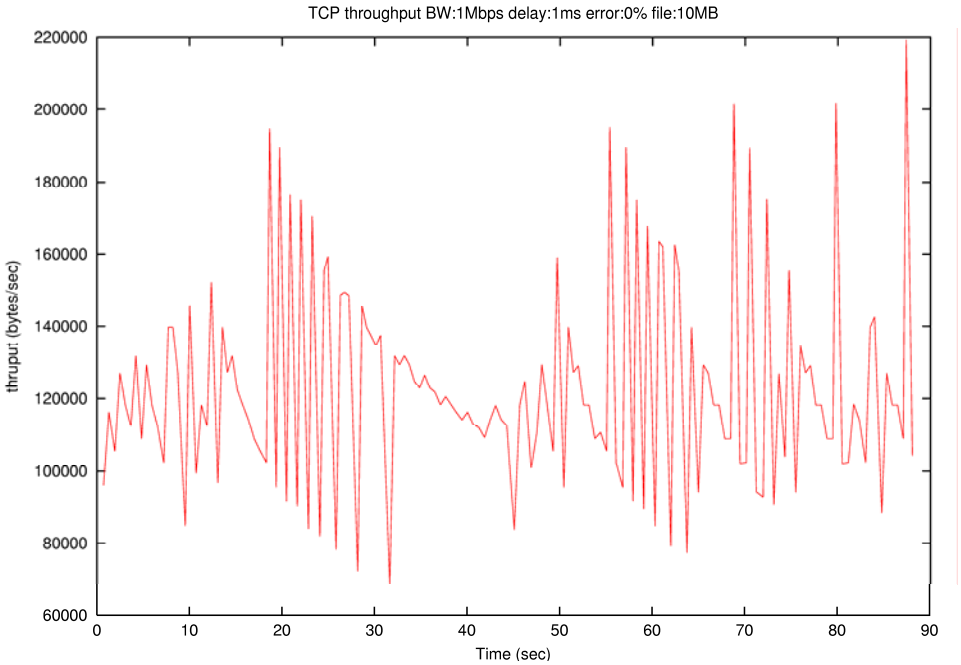
Για το πρωτόκολλο TCP έχουμε τα πιο κάτω αποτελέσματα:



Σχήμα 8: Time sequence graph in TCP(BW:1Mbps,De:1ms,Er:0%,File:10MB)

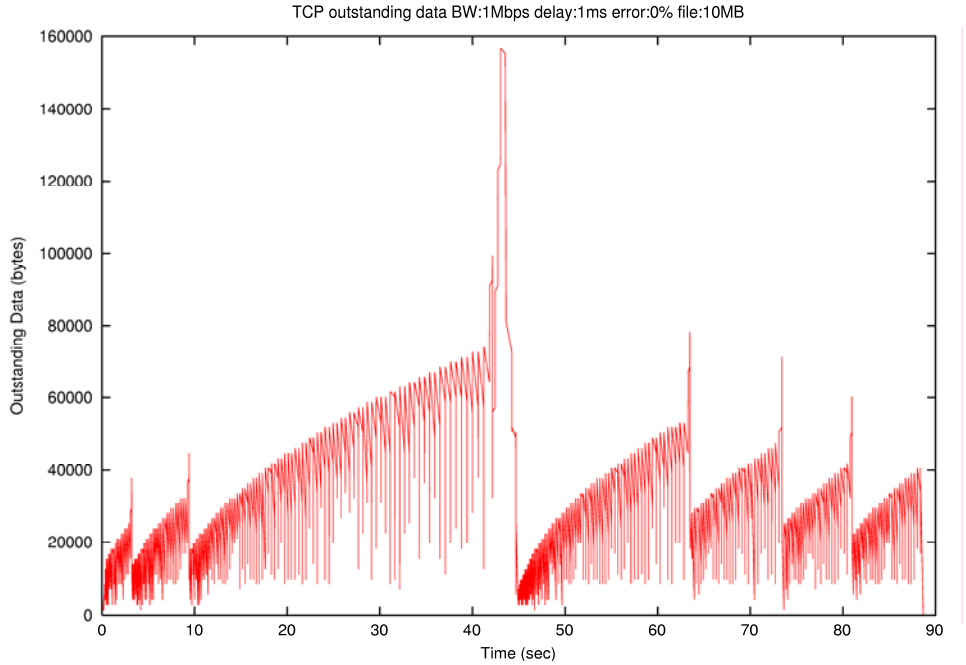
Με κόκκινο χρώμα παρουσιάζονται τα πακέτα δεδομένων, ενώ με πράσινο τα πακέτα αναγνώρισης των δεδομένων.

Παρατηρούμε ότι το πρώτο πακέτο ξεκινά από ένα τυχαίο αριθμό σειράς σύμφωνα με τις προδιαγραφές του πρωτοκόλλου ο οποίος καθορίζεται κατά την αρχικοποίηση της σύνδεσης στην διαδικασία της τριπλής χειραψίας.



Σχήμα 9: Ρυθμαπόδοση σύνδεσης TCP (BW:1Mbps,De:1ms,Er:0%,File:10MB)

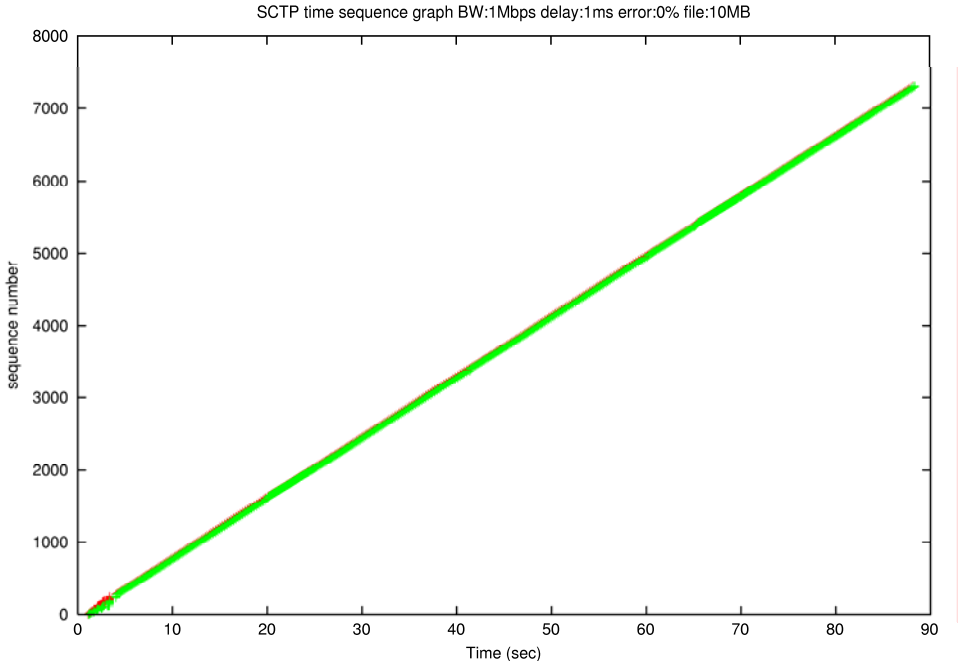
Στο διάγραμμα της ρυθμαπόδοσης μπορούμε να παρατηρήσουμε ότι κυμαίνεται στην τιμή των 125KB/sec. Αυτή η τιμή προέρχεται από το ρυθμό μετάδοσης που έχουμε καθορίσει για αυτό το παράδειγμα στα 1Mbit/sec.



Σχήμα 10: Ανεπιβεβαίωτα πακέτα TCP (BW:1Mbps,De:1ms,Er:0%,File:10MB)

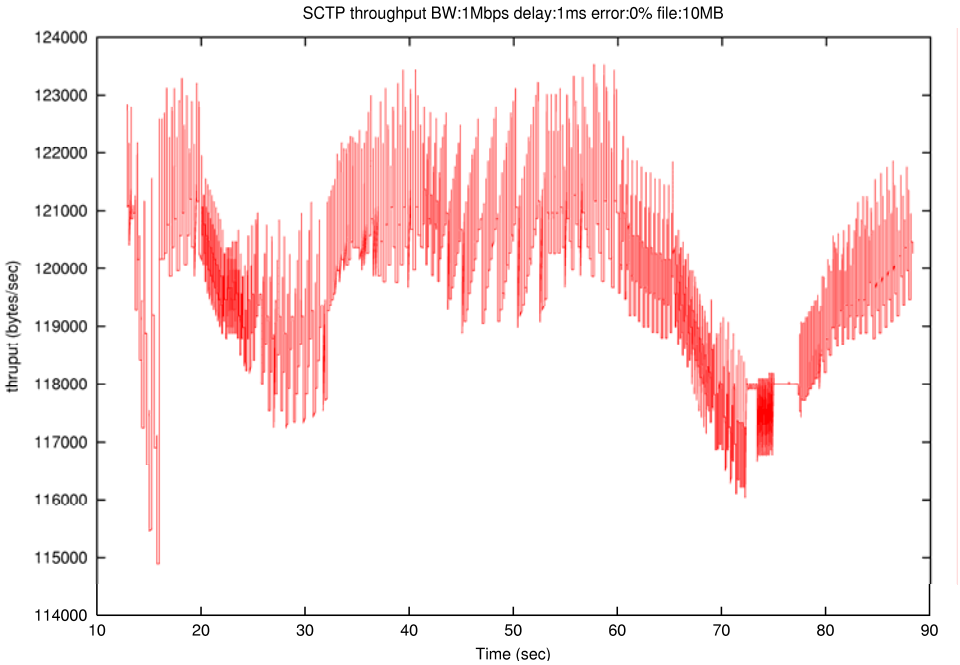
Στην γραφική με τα ανεπιβεβαίωτα πακέτα μπορούμε να πάρουμε μια γενική εικόνα του παραθύρου συμφόρησης όπως αυτό διαμορφώνεται στον αποστολέα για να ρυθμίσει το ρυθμό αποστολής δεδομένων στο δίκτυο.

Τα αντίστοιχα διαγράμματα παρουσιάζουμε και για το πρωτόκολλο SCTP.



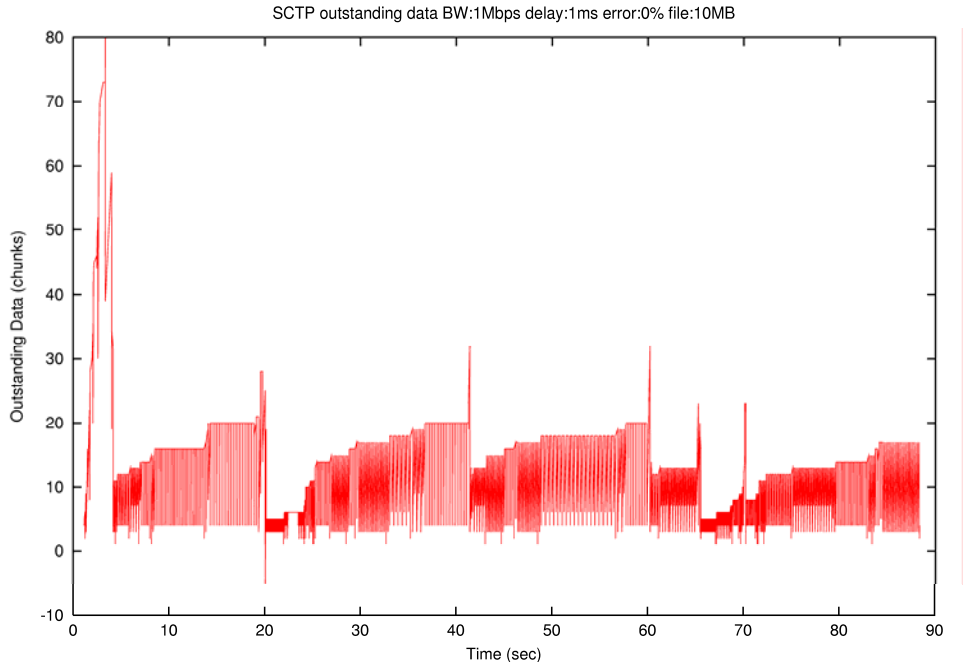
Σχήμα 11: Time sequence graph in SCTP(BW:1Mbps,De:1ms,Er:0%,File:10MB)

Η γραφική έχει γίνει με κανονικοποιημένο τον αριθμό σειράς τεμαχίου. Η διαφορά των δύο πρωτοκόλλων είναι ότι στο TCP ο αριθμός σειράς αυξάνεται ανάλογα με τις ψηφιολέξεις που περιέχει το κάθε πακέτο. Στο SCTP είναι ανεξάρτητος του φορτίου, δεν αναφέρεται σε πακέτο αλλά σε τεμάχιο και αυξάνεται ανάλογα με τον αριθμό τεμαχίων.



Σχήμα 12: Ρυθμαπόδοση σύνδεσης SCTP(BW:1Mbps,De:1ms,Er:0%,File:10MB)

Όπως και στο TCP μπορούμε εύκολα να διακρίνουμε ότι ο μέσος ρυθμός μετάδοσης κυμαίνεται στα 125KB/sec για τον ίδιο λόγο που εξηγήσαμε. Αυτό στη γενική περίπτωση σημαίνει ότι το δίκτυο και οι συνθήκες που επικρατούν σε αυτό δεν αποτελούν εμπόδιο στην μετάδοση δεδομένων, η οποία έχει ως άνω όριο τον μέγιστο ρυθμό μετάδοσης της κάρτας δικτύου.



Σχήμα 13: Ανεπιβεβαίωτα πακέτα SCTP (BW:1Mbps,De:1ms,Er:0%,File:10MB)

Τα ανεπιβεβαίωτα πακέτα που βρίσκονται στο δίκτυο δείχνουν πόσο "γεμάτος είναι ο σωλήνας". Υπενθυμίζουμε ότι δίκτυα με μεγάλο γινόμενο bandwidth*delay έχουν πιο μεγάλη χωρητικότητα. Σε αυτό το παράδειγμα έχουμε περίπου ένα μέγιστο παράθυρο 20 τεμαχίων που ανταποκρίνεται στο γινόμενο αυτό. Τονίζουμε ότι το αντίστοιχο διάγραμμα στο TCP είναι σε bytes και όχι σε τεμάχια, και αυτό είναι μια λειτουργική διαφορά των δύο πρωτοκόλλων.

4.2.2 Παράδειγμα με Bandwidth:2 Mbit/s ,Delay:150ms,Errors:0%

Στο δεύτερο παράδειγμα διαμορφώνουμε το δίκτυο με τα εξής χαρακτηριστικά:

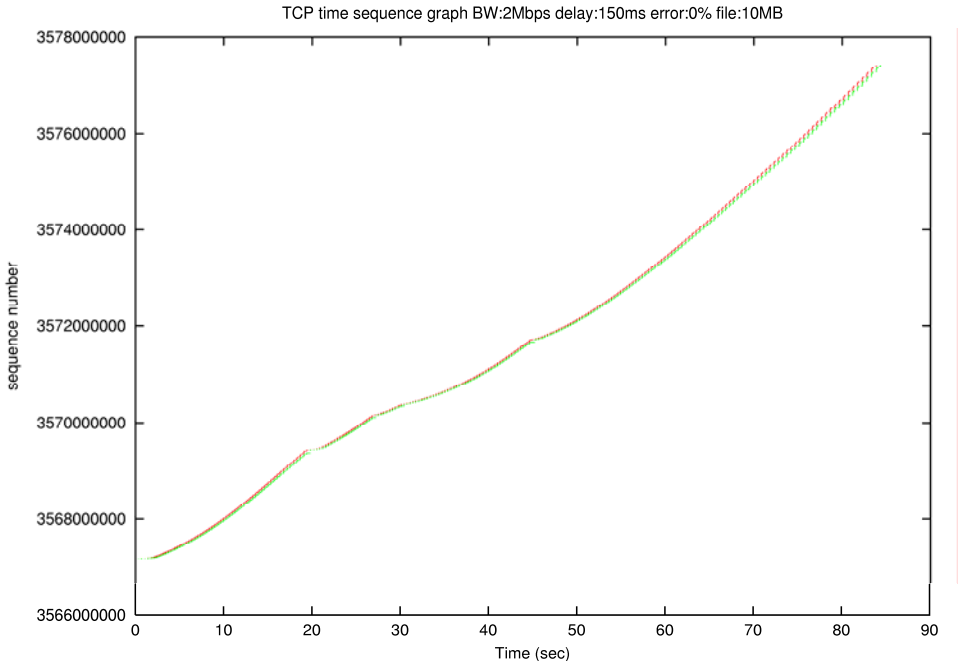
Bandwidth: 2 Mbit/s

Delay: 150ms

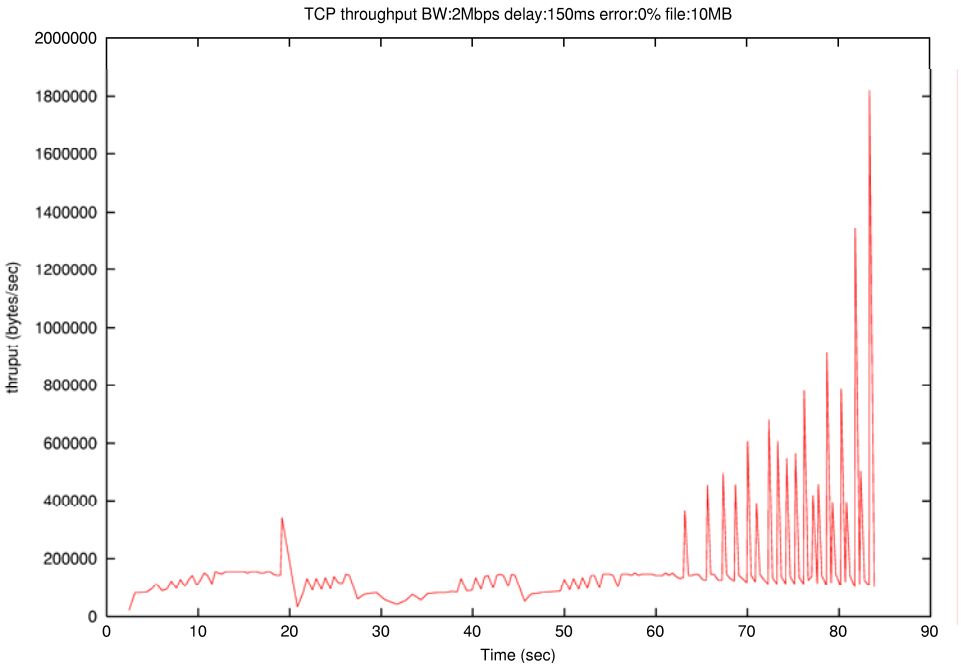
Errors: 0%

Το αρχείο που χρησιμοποιήσαμε για την μεταφορά είναι μεγέθους 10MB.

Με χρήση του πρωτοκόλλου TCP:

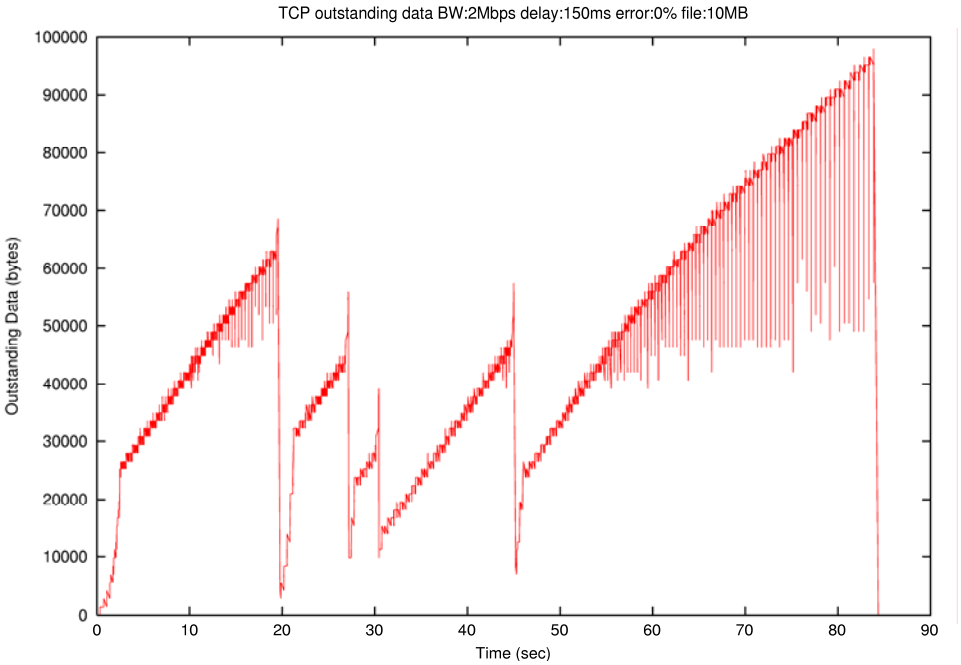


Σχήμα 14: Time sequence graph in TCP(BW:2Mbps,De:150ms,Er:0%,File:10MB)



Σχήμα 15: Ρυθμαπόδοση σύνδεσης TCP (BW:2Mbps,De:150ms,Er:0%,File:10MB)

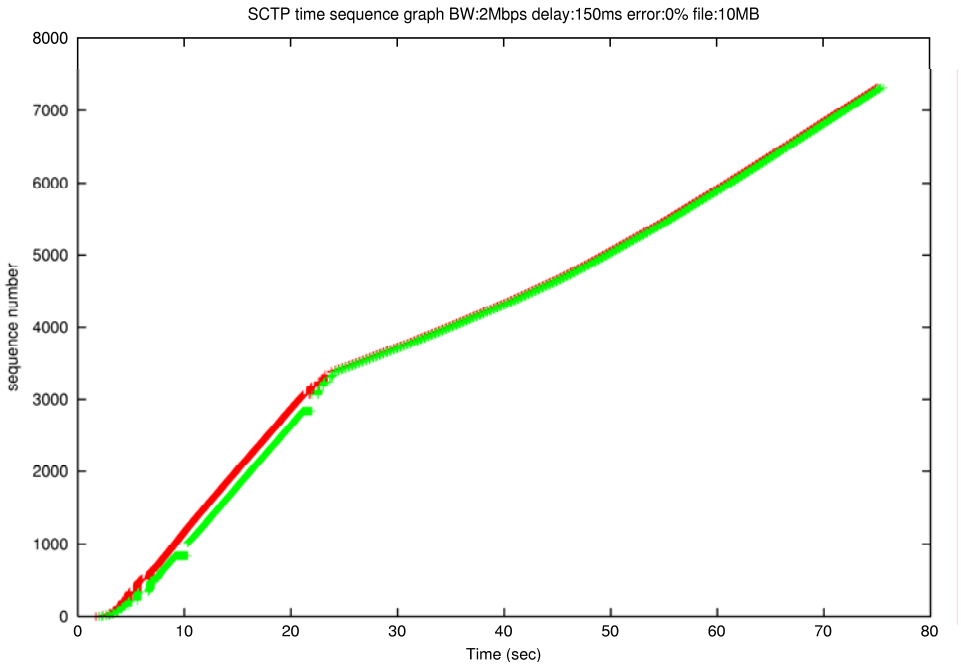
Η αλλαγή του bandwidth σε 2Mbit/s αυξάνει την μέση ρυθμαπόδοση η οποία έχει τώρα μέση τιμή περίπου 250KB/sec.



Σχήμα 16: Ανεπιβεβαίωτα πακέτα TCP (BW:2Mbps,De:150ms,Er:0%,File:10MB)

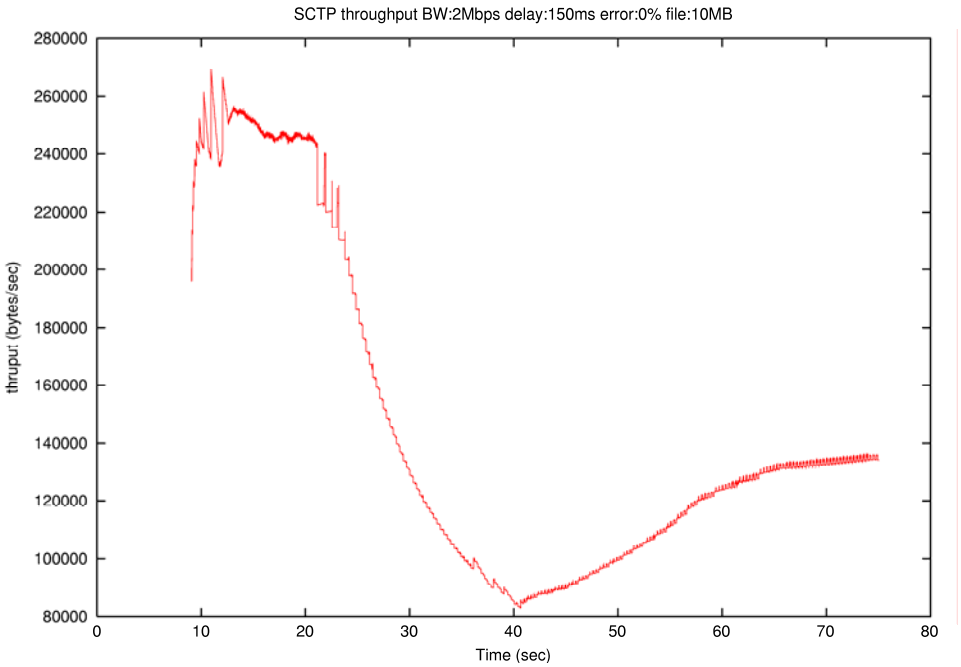
Από το πιο πάνω διάγραμμα φαίνεται ο αλγόριθμος αργής αρχής κατά την έναρξη της σύνδεσης, όπου το παράθυρο ανεβαίνει με εκθετικό ρυθμό. Στο σημείο του κατωφλίου παρατηρούμε την αλλαγή του αλγόριθμου και η αύξηση του παραθύρου είναι πλέων γραμμική. Τα σημεία όπου το παράθυρο πέφτει ίσως να έχουμε απώλειες πακέτων.

Η αντίστοιχη μεταφορά αρχείου με χρήση του πρωτοκόλλου SCTP δίνει:



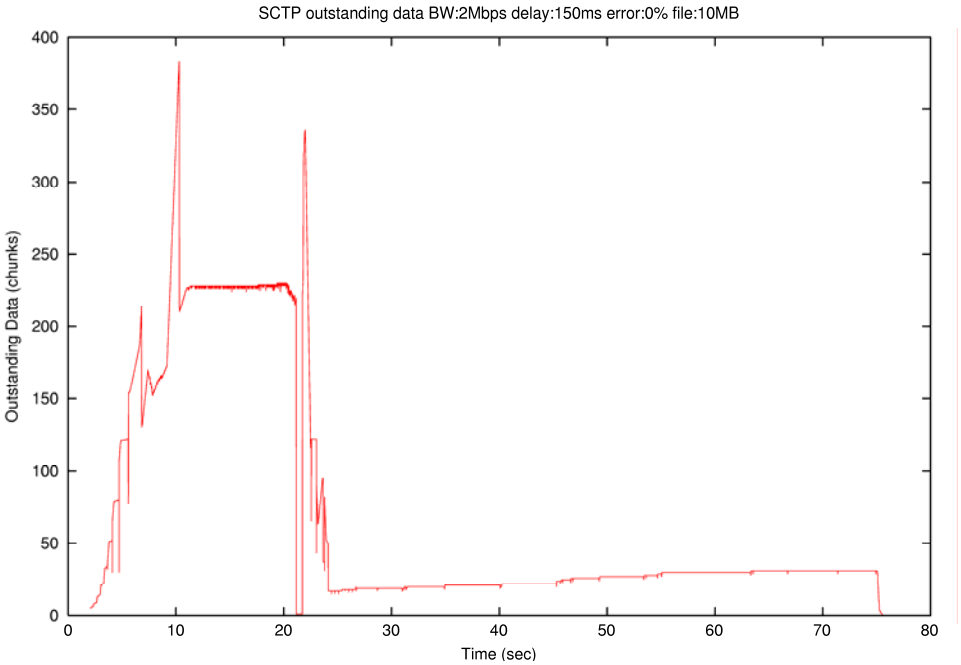
Σχήμα 17: Time sequence graph in SCTP(BW:2Mbps,De:150ms,Er:0%,File:10MB)

Μπορούμε στα σημεία όπου διακόπτεται η συνέχεια των πακέτων αναγνώρισης (με πράσινο χρώμα) να διακρίνουμε κάποια απώλεια στα πακέτα δεδομένων, η οποία γίνεται γνωστή στον αποστολέα με διπλά SACKs τα οποία αναφέρουν οποιοδήποτε κενό στην παραλαβή από τον παραλήπτη.



Σχήμα 18: Ρυθμαπόδοση σύνδεσης SCTP(BW:2Mbps,De:150ms,Er:0%,File:10MB)

Η ρυθμαπόδοση επηρεάζεται από την απώλεια πακέτων και συγκεκριμένα περίπου στο 20 δευτερόλεπτο παρουσιάζεται πτώση της ρυθμαπόδοσης ως συνέπεια χαμένων πακέτων που παρατηρήσαμε στην προηγούμενη γραφική.



Σχήμα 19: Ανεπιβεβαίωτα πακέτα SCTP (BW:2Mbps,De:150ms,Er:0%,File:10MB)

Οι γραφικές στην γενική περίπτωση έχουν πολλά κοινά στην συμπεριφορά αλλά σε καμιά περίπτωση δεν ταυτίζονται και αυτό είναι συνέπεια του τυχαίου παράγοντα λαθών στο δίκτυο.

4.2.3 Παράδειγμα με Bandwidth:2 Mbit/s ,Delay:250ms, Errors:10^{-3}

Στο τρίτο παράδειγμα θα αυξήσουμε την καθυστέρηση στον δίκτυο και θα προσθέσουμε μερικές ακόμα τυχαίες απώλειες πακέτων. Συγκεκριμένα το δίκτυο διαμορφώνεται σε:

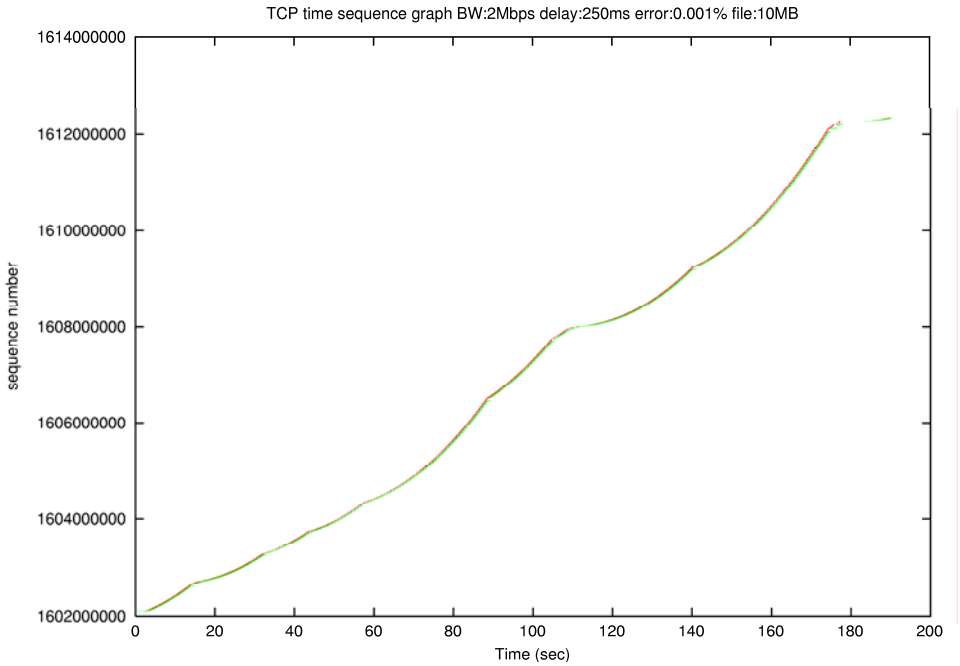
Bandwidth: 2 Mbit/s

Delay: 250ms

Errors: 0.001%

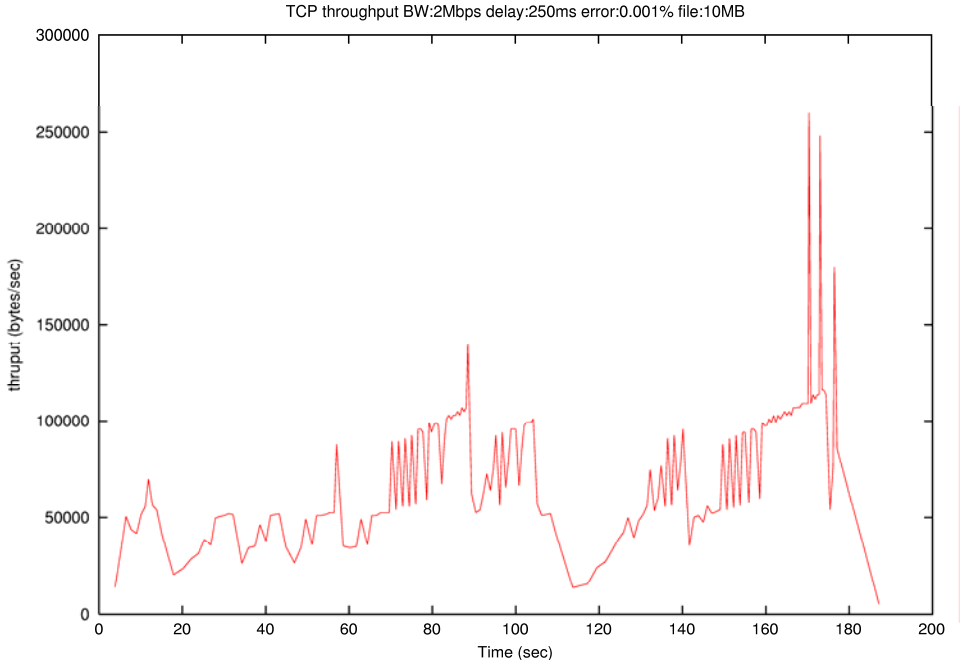
Το αρχείο που χρησιμοποιήσαμε για την μεταφορά είναι μεγέθους 10MB.

Στο TCP παίρνουμε:



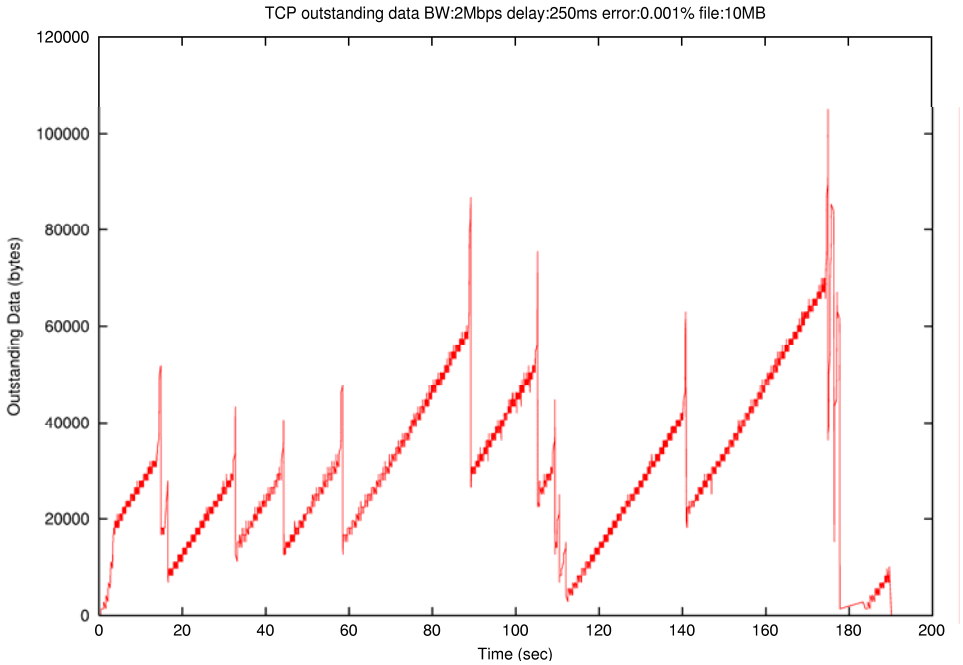
Σχήμα 20: Time sequence graph in TCP(BW:2Mbps,De:250ms,Er:0.001%,File:10MB)

Οι περισσότερες αλλαγές στην κλίση της πιο πάνω γραφικής είναι αποτέλεσμα των επιπρόσθετων λαθών που έχουμε προσθέσει στο δίκτυο.



Σχήμα 21: Ρυθμαπόδοση σύνδεσης TCP (BW:2Mbps,De:250ms,Er:0.001%,File:10MB)

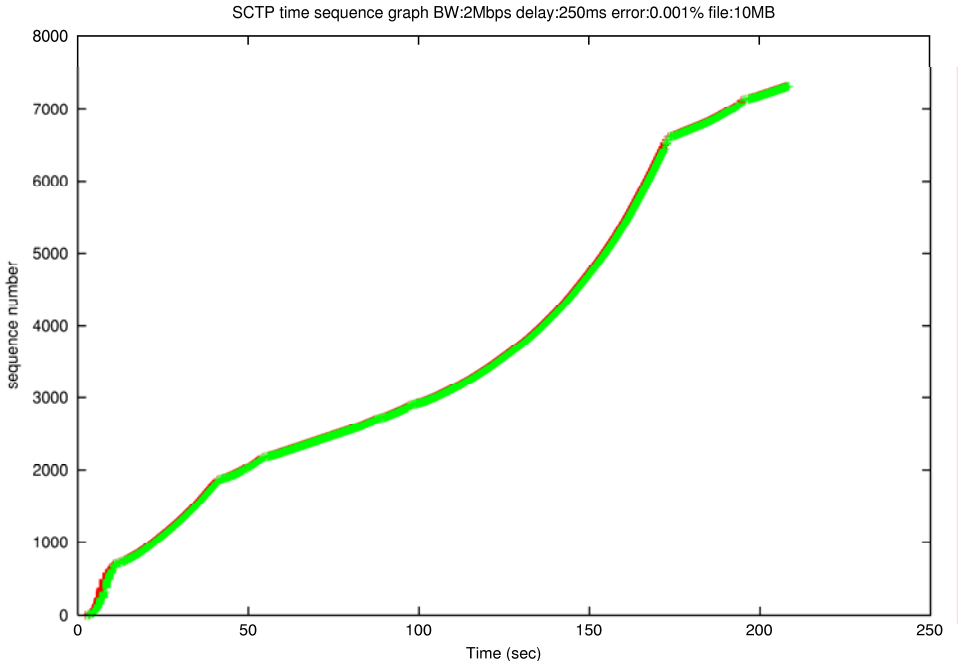
Η αυξημένη καθυστέρηση, καθώς και οι μεγαλύτερες απώλειες προκαλούν μια πτώση της ρυθμαπόδοσης. Φαίνεται ότι πλέων οι συνθήκες του δικτύου είναι τώρα αυτές που καθορίζουν το ρυθμό, ο οποίος δεν μπορεί να φτάσει στην μέγιστη τιμή του που είναι τα 250KB/sec όπως έχουμε σημειώσει στα προηγούμενα παραδείγματα.



Σχήμα 22: Ανεπιβεβαίωτα πακέτα TCP (BW:2Mbps,De:250ms,Er:0.001%,File:10MB)

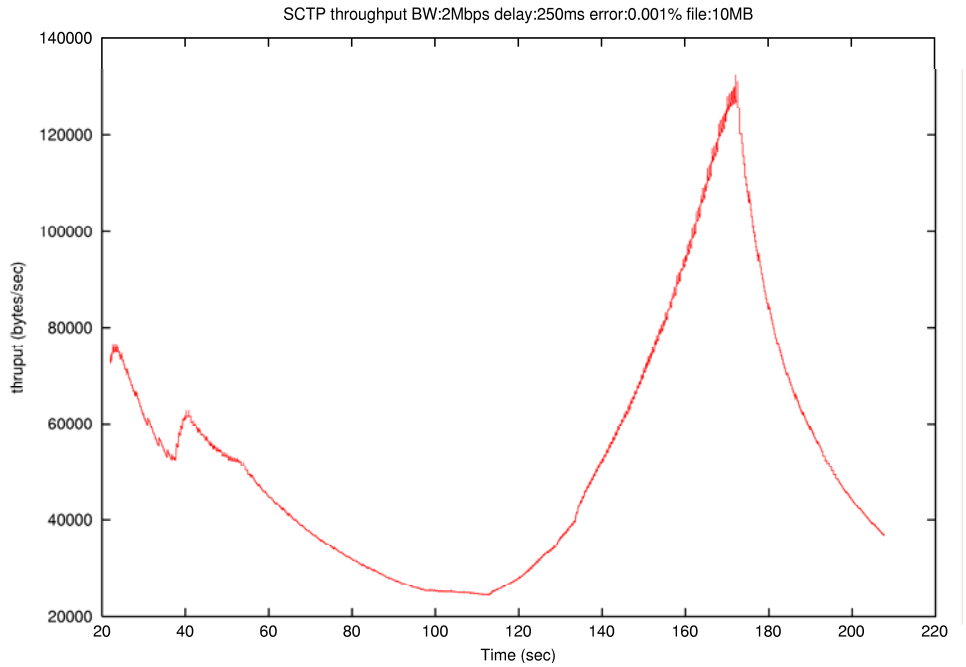
Ο μεγαλύτερος αριθμός απωλειών αποτυπώνεται και στην γραφική των ανεπιβεβαίωτων πακέτων. Πιο πολλές φορές από τις άλλες περιπτώσεις έχουμε πτώση του παραθύρου, το οποίο αναπόφευκτα κινείται σε πιο μικρές τιμές από πριν αφού τα διαδοχικά λάθη δεν του επιτρέπουν να ανέβει, παρόλο που το γινόμενο bandwidth*delay είναι μεγαλύτερο.

Αντίστοιχα στο SCTP:

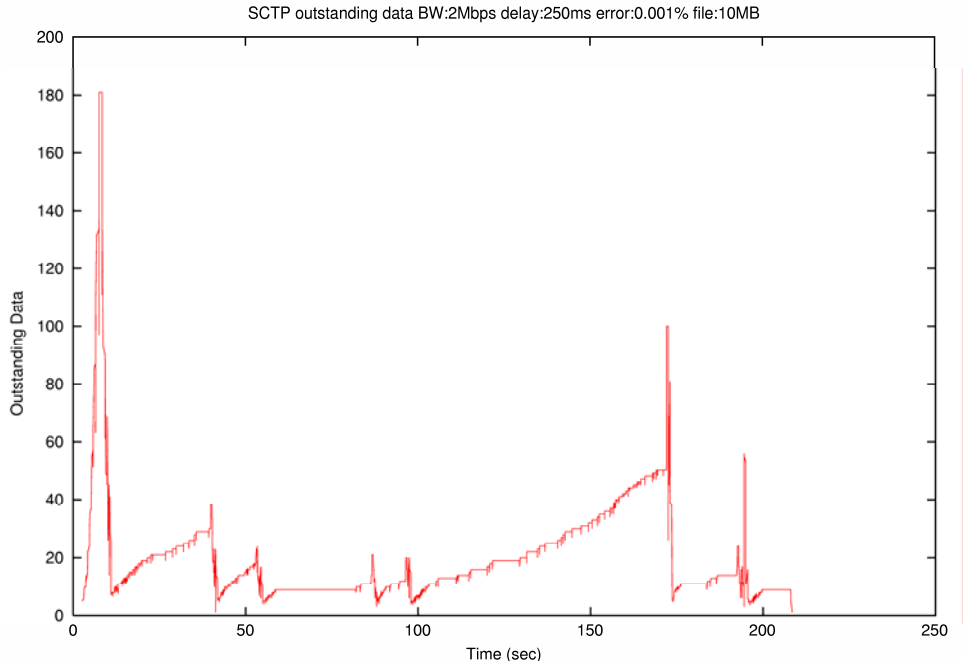


Σχήμα 23: Time sequence graph in SCTP(BW:2Mbps,De:250ms,Er:0.001%,File:10MB)

Παρόμοια συμπεράσματα με τα πιο πάνω ισχύουν και σε αυτό το πρωτόκολλο, αφού οι αλγόριθμοι συμφόρησης του στηρίζονται σε αυτούς του TCP.



Σχήμα 24: Ρυθμαπόδοση σύνδεσης SCTP(BW:2Mbps,De:250ms,Er:0.001%,File:10MB)



Σχήμα 25: Ανεπιβεβαίωτα πακέτα SCTP (BW:2Mbps,De:250ms,Er:0.001%,File:10MB)

Είναι σημαντικό να αναφέρουμε για να μπορεί να γίνει η σύγκριση ότι σε κάθε πακέτο του SCTP επιτρέπονται το μέγιστο 1400 bytes δεδομένων. Στην εφαρμογή

που έχουμε υλοποιήσει και χρησιμοποιούμε σε αυτά τα πειράματα στέλνουμε ένα τεμάχιο δεδομένων σε κάθε πακέτο με μέγιστο φορτίο, δηλαδή 1400 bytes.

Ο περιορισμός αυτός προέρχεται από την υλοποίηση του πρωτοκόλλου που χρησιμοποιούμε και όχι από το ίδιο το πρωτόκολλο. Αν η εφαρμογή γράψει περισσότερα δεδομένα τότε αυτά τεμαχίζονται από το SCTP σε πακέτα των 1400.

Το πρωτόκολλο TCP δημιουργεί πακέτα μεγέθους όχι μεγαλύτερου από την μέγιστη μονάδα μεταφοράς (που καθορίζεται από το πρωτόκολλο μεταφοράς), σύμφωνα με τον αλγόριθμο εύρεσης αυτού του μεγέθους σε μια διαδρομή αν αυτός υπάρχει στην εκάστοτε υλοποίηση του πρωτοκόλλου ή σύμφωνα με την τιμή που ανακοινώνει το άλλο άκρο, δηλαδή ο παραλήπτης. Παρόλα αυτά δίνεται η δυνατότητα στον προγραμματιστή να ρυθμίσει αυτός την μέγιστη τιμή ωφέλιμου φορτίου ανά πακέτο με τον όρο αυτή να μην ξεπερνά την τιμή του μέγιστου μεγέθους τεμαχίου που ανακοίνωσε η άλλη πλευρά. Την δυνατότητα αυτή την χρησιμοποιήσαμε έτσι ώστε να μην επηρεάζεται η σύγκριση μας από τέτοιου είδους “αδικίες”.

Συμπερασματικά και τα δύο πρωτόκολλα σε όλες τις περιπτώσεις που αναφέραμε καθώς και σε αυτές που θα ακολουθήσουν μεταφέρουν σε κάθε πακέτο δεδομένων 1400 bytes ωφέλιμου φορτίου.

Παρατηρούμε ότι το παράθυρο στο SCTP σε αυτή την περίπτωση κινείται περίπου στα 15 τεμάχια δηλαδή 21KB συγκρίσιμο με αυτή του TCP στην αντίστοιχη γραφική παράσταση.

4.2.4 Παράδειγμα με Bandwidth:2 Mbit/s,Delay:150ms,Errors:0.01%

Σαν τελευταίο αναλυτικό παράδειγμα σε αυτή την ενότητα θα παρουσιάσουμε την μεταφορά σε ένα δίκτυο με μεγάλες απώλειες.

Τα χαρακτηριστικά του δικτύου αυτού είναι:

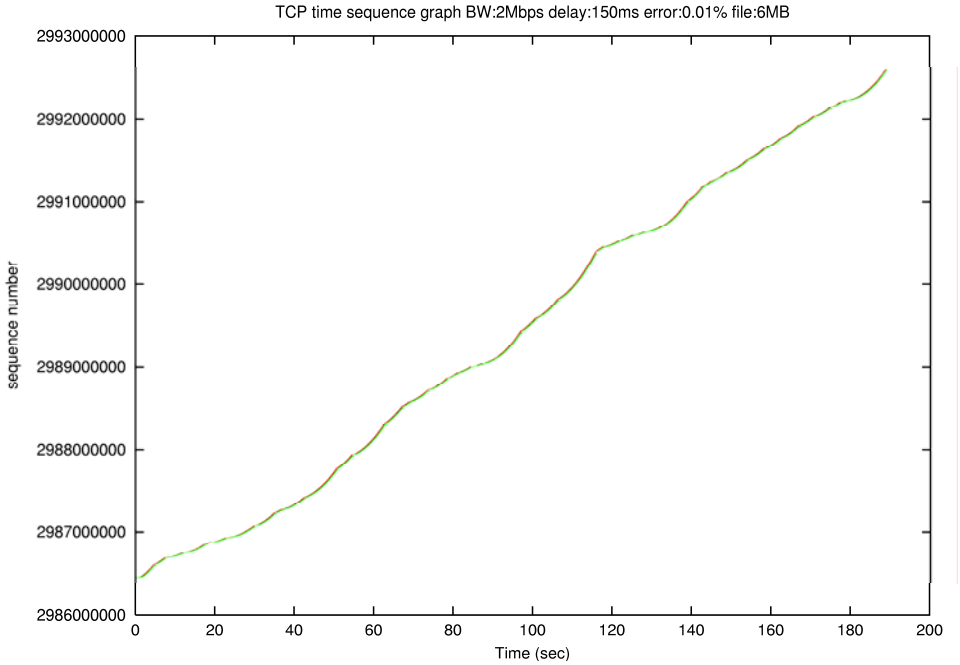
Bandwidth: 2 Mbit/s

Delay: 150ms

Errors: 0.01%

Το αρχείο που χρησιμοποιήσαμε για την μεταφορά είναι μεγέθους 6MB.

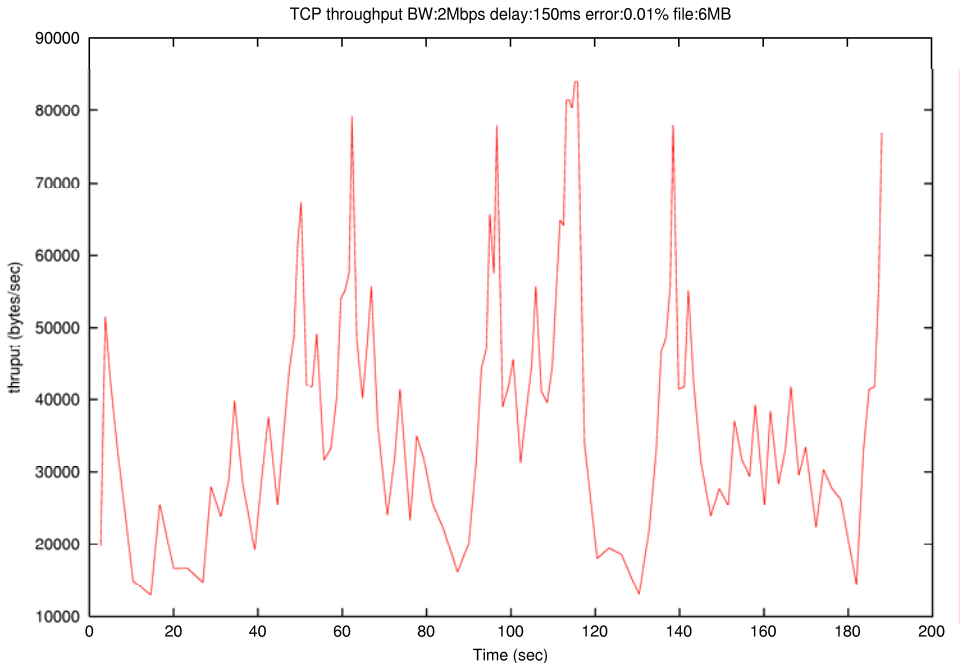
Στη γενική περίπτωση απώλειες τέτοιου βαθμού είναι γνωστό ότι περιορίζουν κατά μεγάλο βαθμό την απόδοση του TCP. Τα αποτελέσματα που πήραμε επιβεβαιώνουν μια τέτοια θεωρία.



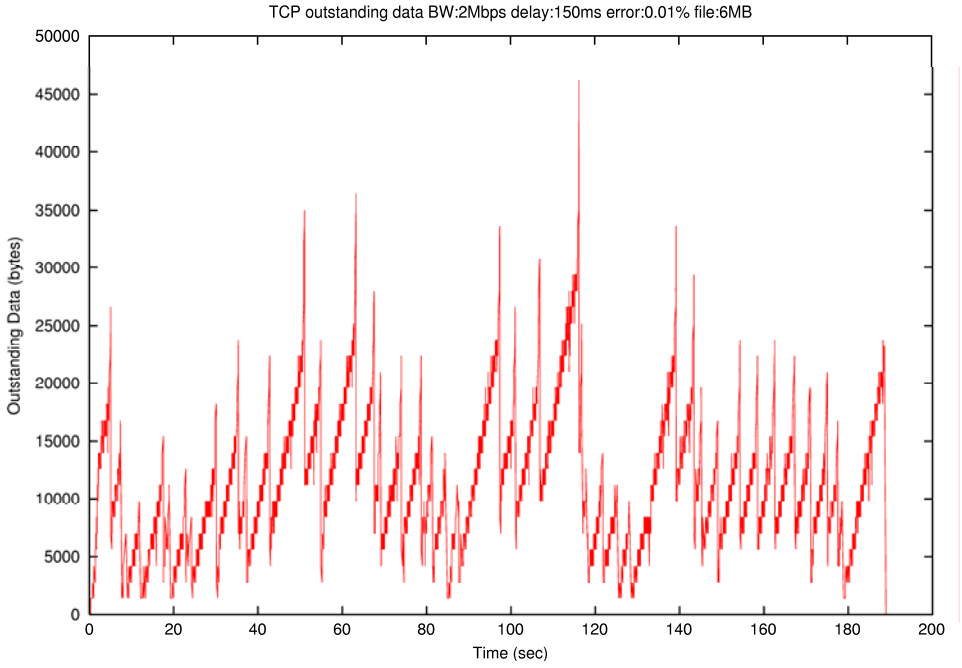
Σχήμα 26: Time sequence graph in TCP(BW:2Mbps,De:150ms,Er:0.01%,File:6MB)

Η κλίση της γραφικής αλλάζει σε πολλά σημεία και είναι μικρότερη από τα άλλα παραδείγματα. Ακόμα ο συνολικός χρόνος μετάδοσης του αρχείου έχει αυξηθεί κατά πολύ στα 120 δευτερόλεπτα. Όλα αυτά είναι ενδείξεις της χαμηλής απόδοσης σε δίκτυα με τέτοια χαρακτηριστικά.

Τα πιο πάνω γίνονται σαφέστερα στη γραφική της ρυθμαπόδοσης.



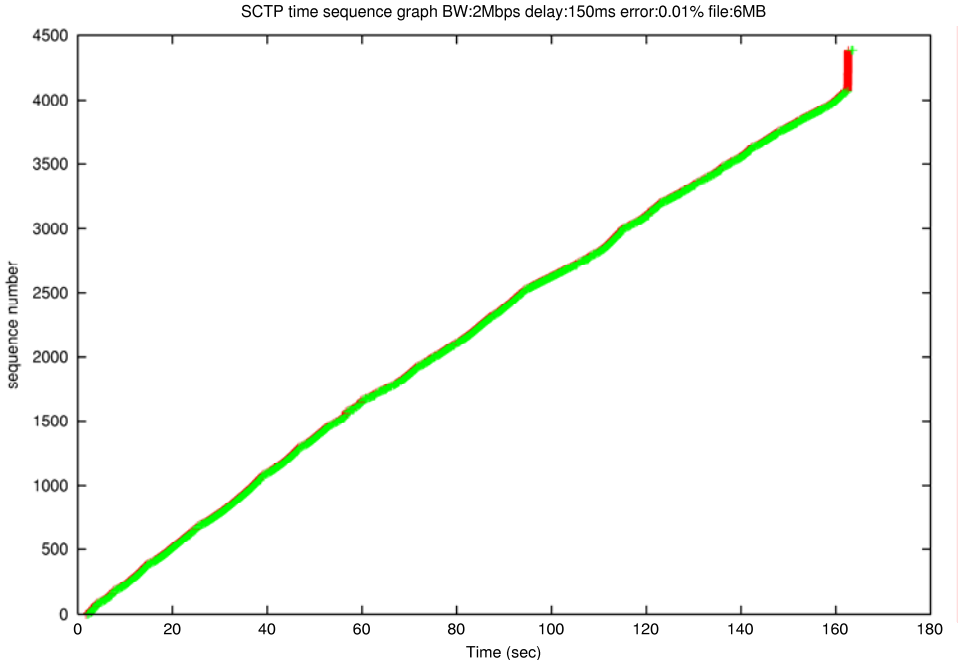
Σχήμα 27: Ρυθμαπόδοση σύνδεσης TCP (BW:2Mbps,De:150ms,Er:0.01%,File:6MB)



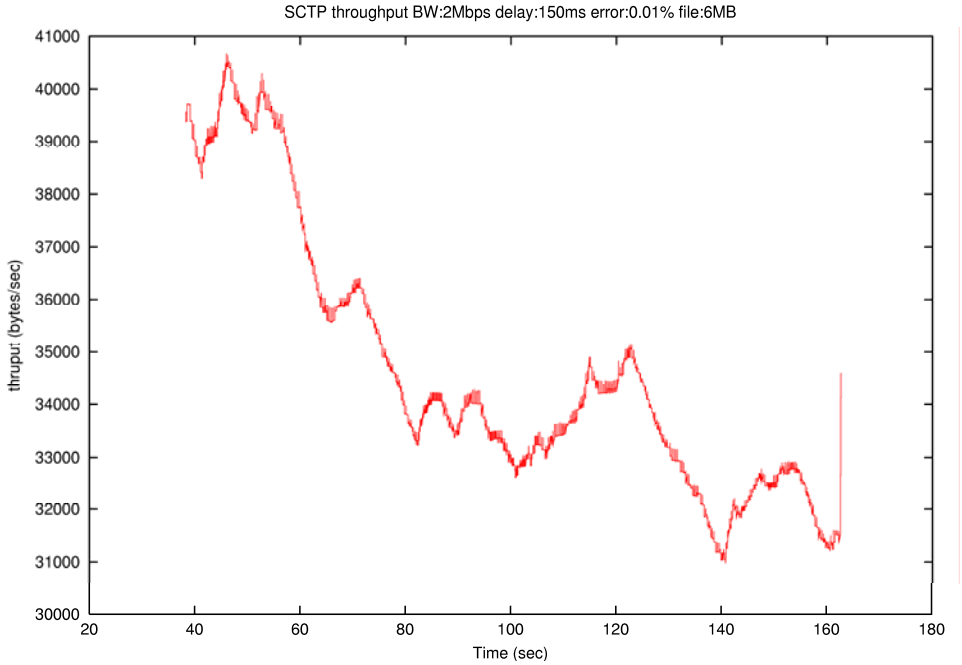
Σχήμα 28: Ανεπιβεβαίωτα πακέτα TCP (BW:2Mbps,De:150ms,Er:0.01%,File:6MB)

Το παράθυρο παραμένει σε πολύ χαμηλές τιμές και παρουσιάζει πολλές πτώσεις από το μεγάλο ποσοστό απωλειών. Σε καμιά περίπτωση δεν μπορεί το πρωτόκολλο μεταφοράς να "γεμίσει" το δίκτυο.

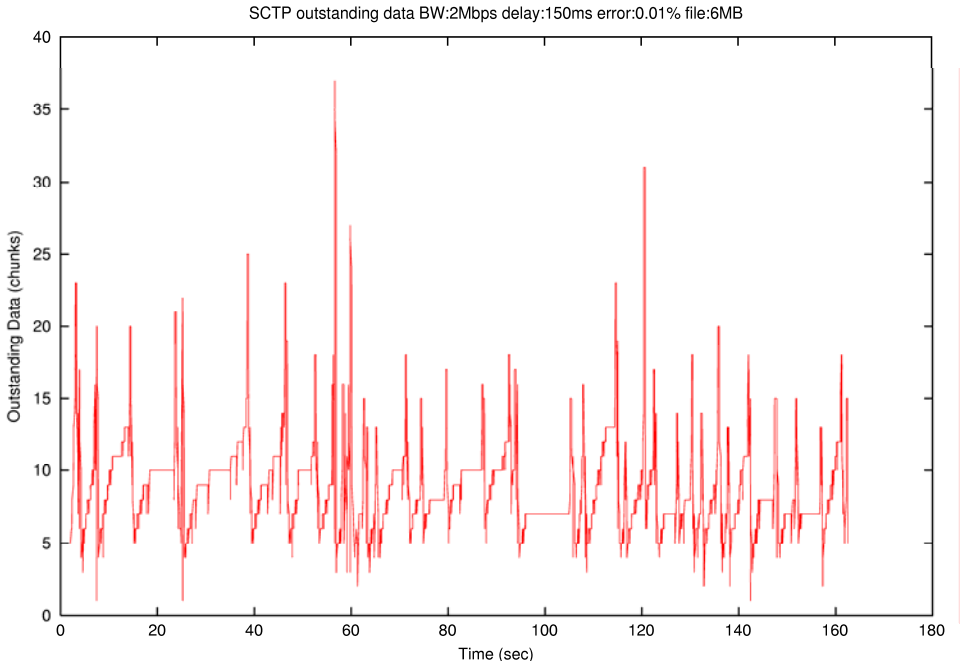
Τα ίδια φαινόμενα παρατηρούμε και στο SCTP:



Σχήμα 29: Time sequence graph in SCTP(BW:2Mbps,De:150ms,Er:0.01%,File:6MB)



Σχήμα 30: Ρυθμαπόδοση σύνδεσης SCTP(BW:2Mbps,De:150ms,Er:0.01%,File:6MB)



Σχήμα 31: Ανεπιβεβαίωτα πακέτα SCTP (BW:2Mbps,De:150ms,Er:0.01%,File:6MB)

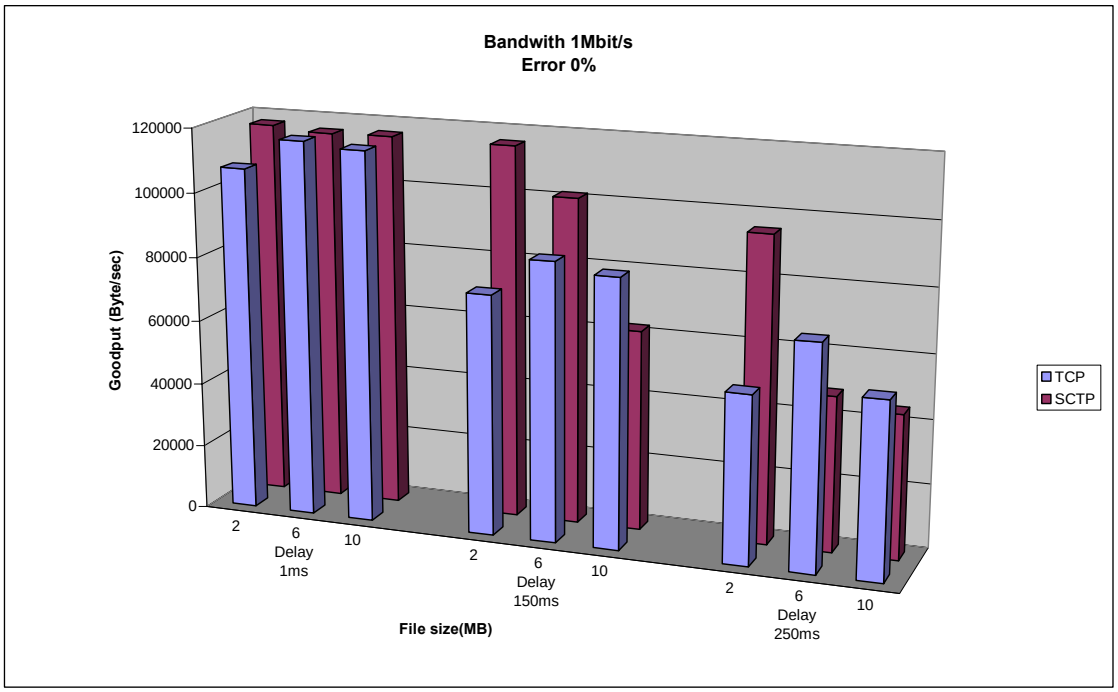
4.3 Συνολικά αποτελέσματα μεταφοράς ενός αρχείου

Το σημαντικότερο μέτρο σύγκρισης για μεταφορά δεδομένων είναι η ρυθμαπόδοση. Όταν έχουμε μεταφορά συγκεκριμένου αρχείου με χρήση και των δύο πρωτοκόλλων

μπορούμε να συγκρίνουμε την μέση ρυθμαπόδοση και όχι τις στιγμιαίες τιμές ρυθμαπόδοσης. Η τιμή αυτή ισούται με το συνολικό μέγεθος του αρχείου που μεταφέρεται δια του συνολικού χρόνου διάρκειας της μετάδοσης.

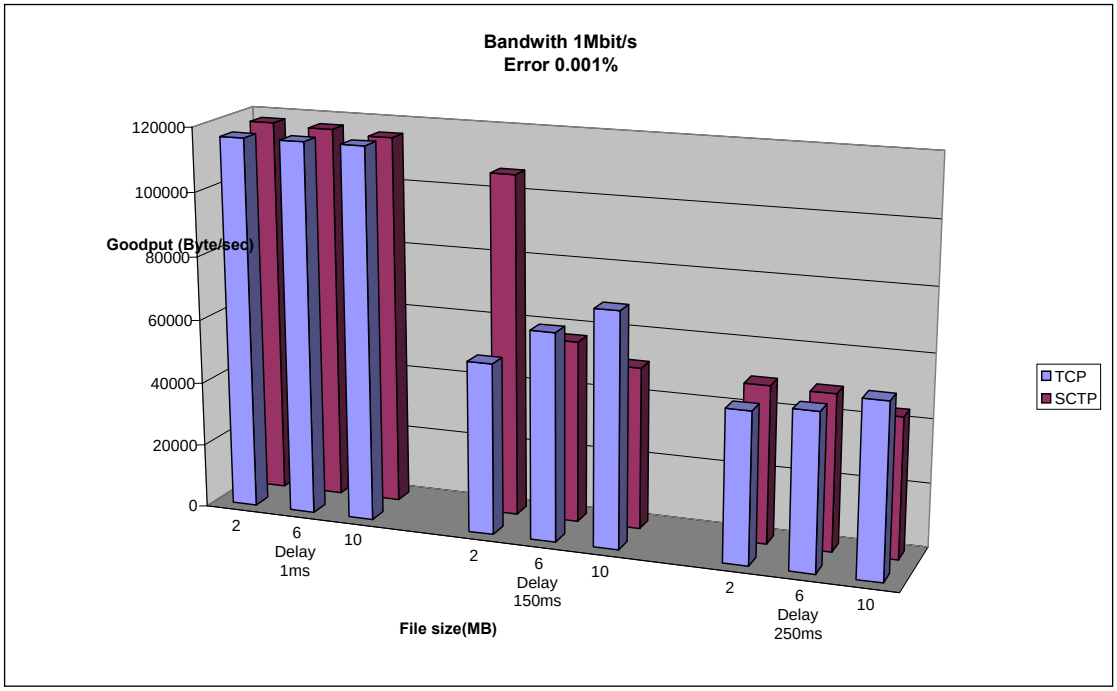
Θα παρουσιάσουμε συνολικά τις τιμές της μέσης ρυθμαπόδοσης (goodput) για διάφορα μεγέθη αρχείων σε δίκτυα με διαφορετικό εύρος ζώνης, καθυστέρηση και ποσοστό απωλειών.

4.3.1 Μεταβολή της καθυστέρησης



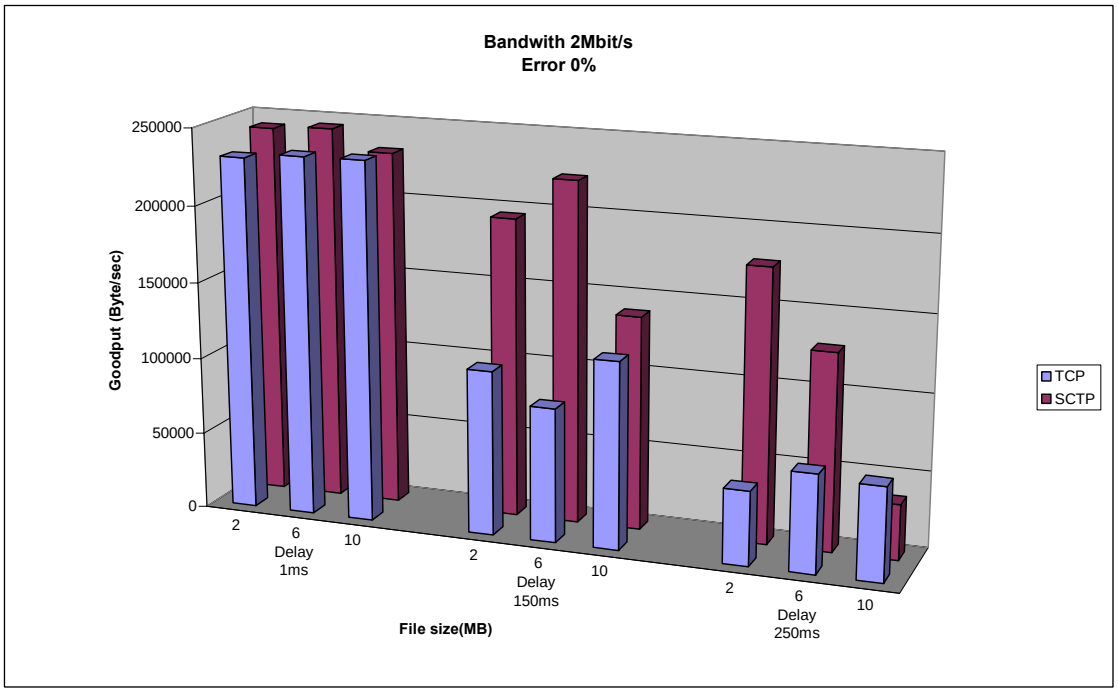
Σχήμα 32: Συνοπτικά αποτελέσματα(BW:1Mbps,De:1-150-250ms,Er:0%,File:2-6-10MB)

Σε δίκτυο με μικρή καθυστέρηση και τα δύο πρωτόκολλα εκμεταλλεύονται όλο το εύρος ζώνης που προσφέρει η κάρτα δικτύου, καθώς η καθυστέρηση αυξάνει η μέση ρυθμαπόδοση πέφτει. Πρέπει να τονίσουμε ότι τα πειράματα γίνονται σε φυσικό δίκτυο υπολογιστών και ότι ένα ποσοστό λαθών υπάρχει και είναι αδύνατο να αποφευχθεί.



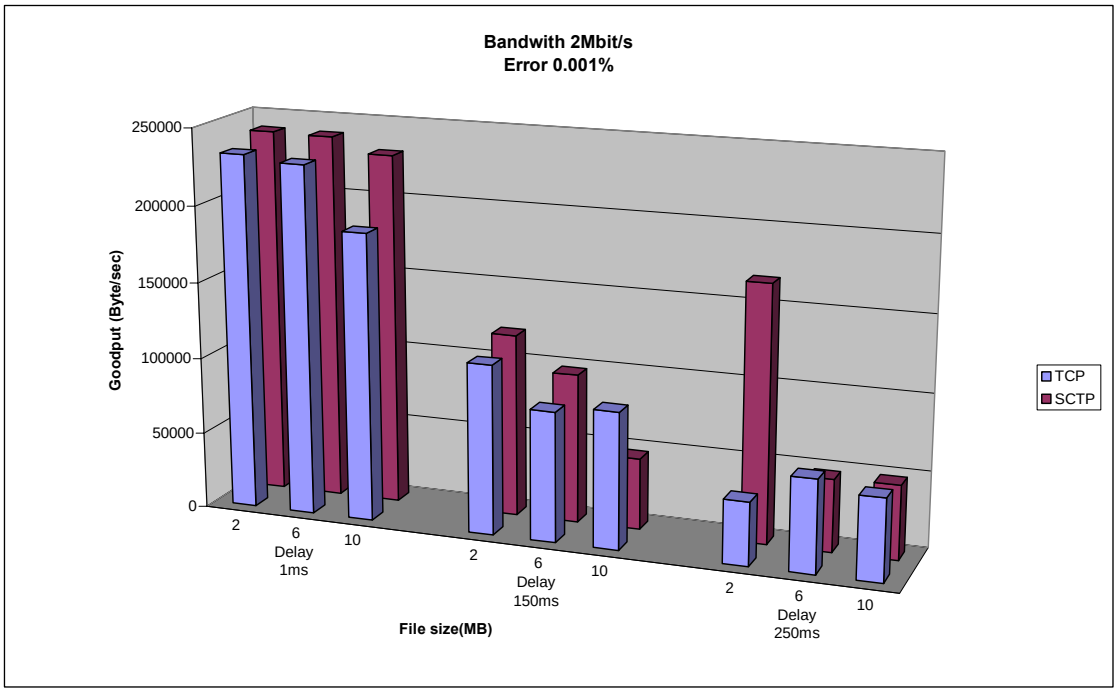
Σχήμα 33: Συνοπτικά αποτελέσματα(BW:1Mbps,De:1-150-250ms,Er:0.001%,File:2-6-10MB)

Εκτός από μεμονωμένες περιπτώσεις τα δύο πρωτόκολλα είναι ισοδύναμα και δεν μπορούμε να ξεχωρίσουμε γενικό κανόνα που να υπερτερεί κάποιο από αυτά. Αυτό εξάλλου είναι και το αναμενόμενο, καθώς ο σχεδιασμός των αλγορίθμων συμφόρησης του είναι πανομοιότυπος.



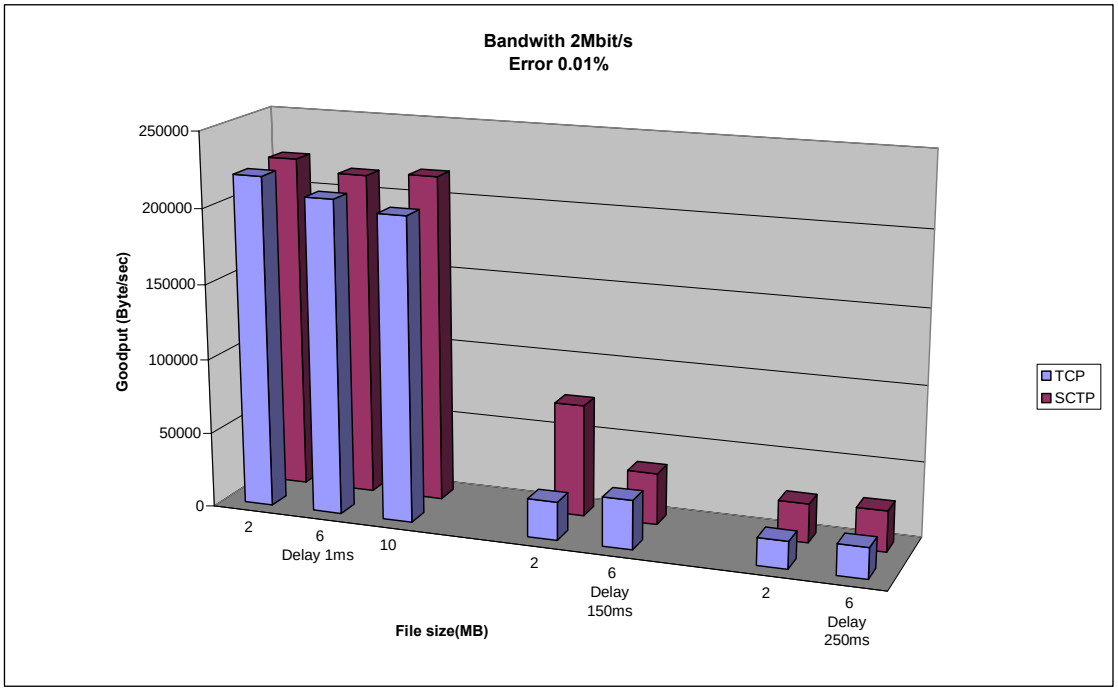
Σχήμα 34: Συνοπτικά αποτελέσματα(BW:2Mbps,De:1-150-250ms,Er:0%,File:2-6-10MB)

Κατά την μεταφορά μικρών αρχείων βλέπουμε ότι υπάρχουν περιπτώσεις όπου οι τιμές αποκλίνουν από το ένα πρωτόκολλο στο άλλο. Αυτό δεν έχει να κάνει με καλύτερη συμπεριφορά ενός από τα δύο αλλά με την τυχαία απώλεια πακέτων. Επειδή στα μικρά αρχεία υπάρχει πιθανότητα σε μια μεταφορά να γίνει κάποιο λάθος ενώ σε μια άλλη να μην παρουσιαστεί απώλεια σε συνδυασμό με τον μικρό συνολικό χρόνο μεταφοράς οι διάφορες μετρήσεις μπορούν να διαφέρουν κατά μεγάλο ποσοστό ακόμα και αν χρησιμοποιηθεί το ίδιο πρωτόκολλο. Στην περίπτωση του μεγαλύτερου αρχείου των 10MB όπου τα πακέτα που ανταλλάσσονται είναι πάρα πολλά μια μικρή διαφορά στις απώλειες δεν επηρεάζει την μέτρηση της μέσης ρυθμαπόδοσης.



Σχήμα 35: Συνοπτικά αποτελέσματα(BW:2Mbps,De:1-150-250ms,Er:0.001%,File:2-6-10MB)

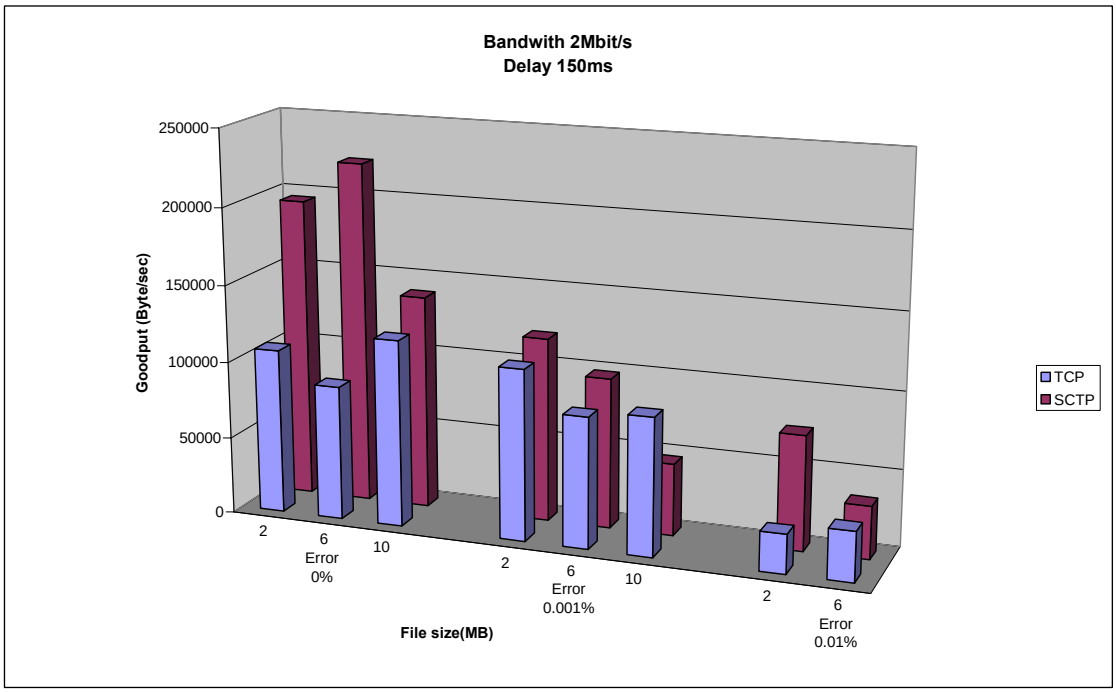
Ενδιαφέρων παρουσιάζει η περίπτωση που έχουμε μεγάλο ποσοστό απωλειών. Ο συνδυασμός τους με μεγάλες καθυστερήσεις παρατηρούμε ότι ρίχνει πάρα πολύ την ρυθμαπόδοση σε σχέση με όλα τα άλλα παραδείγματα.



Σχήμα 36: Συνοπτικά αποτελέσματα(BW:2Mbps,De:1-150-250ms,Er:0.01%,File:2-6-10MB)

4.3.2 Μεταβολή του ποσοστού λαθών

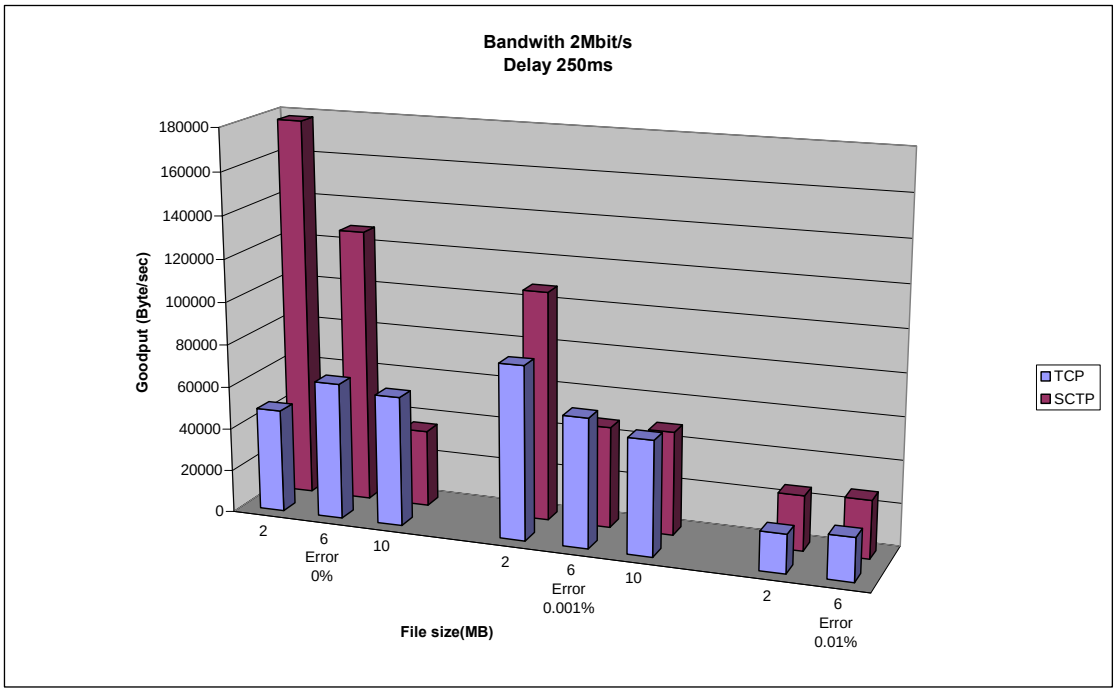
Θα παρουσιάσουμε και μερικές συγκεντρωτικές παραστάσεις όπου η καθυστέρηση διατηρείται σταθερή για να μπορούμε με ευκολία να παρατηρήσουμε την επίδραση των απωλειών στη μετάδοση.



Σχήμα 37: Συνοπτικά αποτελέσματα(BW:2Mbps,De:150,Er:0-0.001-0.01%,File:2-6-10MB)

Η προσθήκη 0.001% απωλειών στις υπάρχουσες του δικτύου δεν φαίνεται να επηρεάζουν σε μεγάλο βαθμό την μετάδοση. Ο λόγος είναι ότι οι φυσικές απώλειες του δικτύου είναι της τάξεως του 10^{-3} και συνεπώς με την πρόσθεση απωλειών της ίδιας τάξης δεν κάνουμε σημαντική διαφορά. Αντίθετα όμως ένα ποσοστό απώλειας της τάξεως του 10^{-2} αλλάζει την κατάσταση και προκαλεί μια αισθητή πτώση της ρυθμαπόδοσης.

Αυτό μπορούμε να το επιβεβαιώσουμε και με καθυστέρηση 250ms όπως παρουσιάζεται πιο κάτω.



Σχήμα 38: Συνοπτικά αποτελέσματα(BW:2Mbps,De:250ms,Er:0-0.001-0.01%,File:2-6-10MB)

5. ΜΕΤΑΦΟΡΑ ΠΟΛΛΩΝ ΑΡΧΕΙΩΝ (ΣΕΝΑΡΙΟ Α)

Η δεύτερη ενότητα μετρήσεων περιλαμβάνει μεταφορά πολλών αρχείων σε διάφορους συνδυασμούς. Η γενική ιδέα είναι να συγκρίνουμε την συμπεριφορά των πρωτοκόλλων όταν ένα αρχείο μπαίνει στη διαδικασία μεταφοράς όταν ήδη στο δίκτυο μεταφέρονται άλλα αρχεία που ουσιαστικά αποτελούν «παρασκηνιακή» κίνηση. Είδαμε στην προηγούμενη ενότητα ότι στη μεταφορά ενός αρχείου τα δύο πρωτόκολλα έχουν παρόμοιες συμπεριφορές και αποδώσεις. Η λειτουργία του SCTP που θέλουμε να εκμεταλλευτούμε είναι αυτή των πολλαπλών ροών σε κάθε σύνδεση. Στα παραδείγματα που ακολουθούν στο πρωτόκολλο TCP κάθε αρχείο μεταφέρεται σε ξεχωριστή σύνδεση με δικό του αυτόνομο έλεγχο συμφόρησης ενώ στο SCTP όλα τα αρχεία μεταφέρονται από την ίδια σύνδεση αλλά σε διαφορετική ροή.

Ο αριθμός των ροών που μπορεί να υποστηρίξει το SCTP σε μια σύνδεση ορίζεται κατά την αρχικοποίηση από το κάθε άκρο και χωρίζεται σε εισερχόμενες και εξερχόμενες ροές. Ο αριθμός που μπορεί να τεθεί είναι των 16 bit εκτός και αν το ίδιο το λειτουργικό θέτει κάποιο ανώτατο όριο. Για τις μετρήσεις μας κάνουμε ταυτόχρονη μεταφορά δέκα αρχείων και για αυτό το λόγο η εφαρμογή ορίζει δέκα ροές κατά την αρχικοποίηση με τον ακόλουθο τρόπο [6].

```
struct sctp_initmsg initm;
bzero(&initm,sizeof(initm));
initm.sinit_num_ostreams = 10;
initm.sinit_max_instreams = 10;
ext_setsockopt(   sock_fd,´
a                 IPPROTO_SCTP,
a                 SCTP_INITMSG,
a                 &initm,sizeof(initm))

ext_connect(sock_fd,(SA *)&servaddr,sizeof(servaddr));
```

Κώδικας καθορισμού αριθμού ροών

Θα πραγματοποιήσουμε δύο σενάρια για την διεξαγωγή μετρήσεων κατά τα οποία μεταφέρουμε εννέα αρχεία ταυτόχρονα, που αποτελούν μια κίνηση δικτύου που στη

πραγματικότητα θα μπορούσε να είναι η υπάρχουσα κίνηση την στιγμή που θέλουμε να πραγματοποιήσουμε μια μεταφορά.

Στο πρώτο σενάριο το αρχείο που ζητάμε να μεταφερθεί μπαίνει εμβόλιμα κατά την διάρκεια που μεταφέρονται τα υπόλοιπα και τελειώνει η μεταφορά του πριν κάποιο από τα άλλα αρχεία τελειώσει.

Στο δεύτερο σενάριο το αρχείο πάλι ξεκινά όταν ήδη στο δίκτυο μεταφέρονται τα άλλα αρχεία αλλά η μεταφορά διαρκεί έτσι ώστε να τελειώνει μετά από όλα τα υπόλοιπα εννέα αρχεία.

Για την διεξαγωγή των πειραμάτων αυτών δεν χρησιμοποιήσαμε το δίκτυο που περιγράψαμε κατά την μεταφορά ενός αρχείου, αλλά τρέξαμε τα ίδια προγράμματα σε ένα υπολογιστή δίνοντας ψευδοδιευθύνσεις στην διεπαφή που χρησιμοποιούμε (alias network address). Με τον τρόπο αυτό πετυχαίνουμε πλήρη έλεγχο στο ποσοστό λαθών που εισάγουμε, γιατί δεν υπάρχουν φυσικές απώλειες πακέτων κατά την μεταφορά.

Ακόμα χρειάστηκε να κάνουμε κάποιες ρυθμίσεις σε μεταβλητές του πυρήνα για να μπορούμε να έχουμε μέγιστη απόδοση μεταφοράς κάτω από συνθήκες LFN δικτύων. Γενικά δίκτυα με μεγάλο γινόμενο bandwidth*delay απαιτούν στις συνδέσεις από τους κόμβους να έχουν μεγάλους αποταμιευτές (buffers). Το TCP μιας και τρέχει στον πυρήνα του λειτουργικού κάνει χρήση των εξής δύο μεταβλητών πυρήνα:

net.inet.tcp.recvspace

net.inet.tcp.sendspace

Οι μεταβλητές αυτές ρυθμίζουν τους χώρους αυτούς. Ο διαθέσιμος χώρος στον receiver buffer περιορίζει τον παράθυρο που θα διαφημίσει το TCP στο άλλο άκρο. Αν και η τιμή αυτή δεν θα μπορούσε να ξεπεράσει ένα αριθμό με 16 bit χρησιμοποιούμε την κλιμάκωση παραθύρου που αναφέραμε στις νέες τεχνολογίες για να επιτρέπουμε πιο μεγάλες τιμές. Από την άλλη πλευρά και ο buffer του αποστολέα πρέπει να έχει αρκετό χώρο γιατί τα δεδομένα που αποστέλλονται διατηρούνται εκεί μέχρι να επιβεβαιωθούν και μόνο τότε διαγράφονται για να υπάρξει χώρος για να αποσταλούν νέα δεδομένα. Μια τιμή για τους buffer αυτούς που καλύπτει αυτές τις

απαιτήσεις είναι το γινωμένο bandwidth*delay όπου delay είναι ο χρόνος μετ' επιστροφής.

Η υλοποίηση του πρωτοκόλλου SCTP που χρησιμοποιούμε τρέχει στο χώρο του χρήστη (userland) και δεν έχει δικές του μεταβλητές πυρήνα. Παρόλα αυτά επειδή χρησιμοποιεί raw sockets πάνω από το πρωτόκολλο IP για να λειτουργήσει παίρνει τις τιμές για τους buffers από την μεταβλητή net.inet.raw.recvspace.

Για να μπορούμε να δίνουμε τις μεγάλες τιμές που απαιτούνται σε αυτούς τους αποταμιευτές αλλάζουμε και την μεταβλητή kern.ipc.maxsockbuf έτσι ώστε να υπάρχει διαθέσιμη μνήμη για κάθε σύνδεση.

Στην αλλαγή των μεταβλητών για το SCTP χρειάζεται μεγάλη προσοχή γιατί υπάρχει περίπτωση ο αποταμιευτής να υπερχειλίσει όταν είναι πολύ μεγάλος σε μέγεθος και να παρουσιαστεί αδικαιολόγητα αυξημένο ποσοστό απωλειών.

5.1 Αναλυτική παρουσίαση μεταφοράς πολλών αρχείων με ένα ενδιάμεσο

Θα παρουσιάσουμε ενδεικτικά αναλυτικές γραφικές για μερικές από τις περιπτώσεις του πρώτου σεναρίου. Πειραματιστήκαμε με διάφορες παραμέτρους για το δίκτυο όπως φαίνεται στον πίνακα που ακολουθεί.

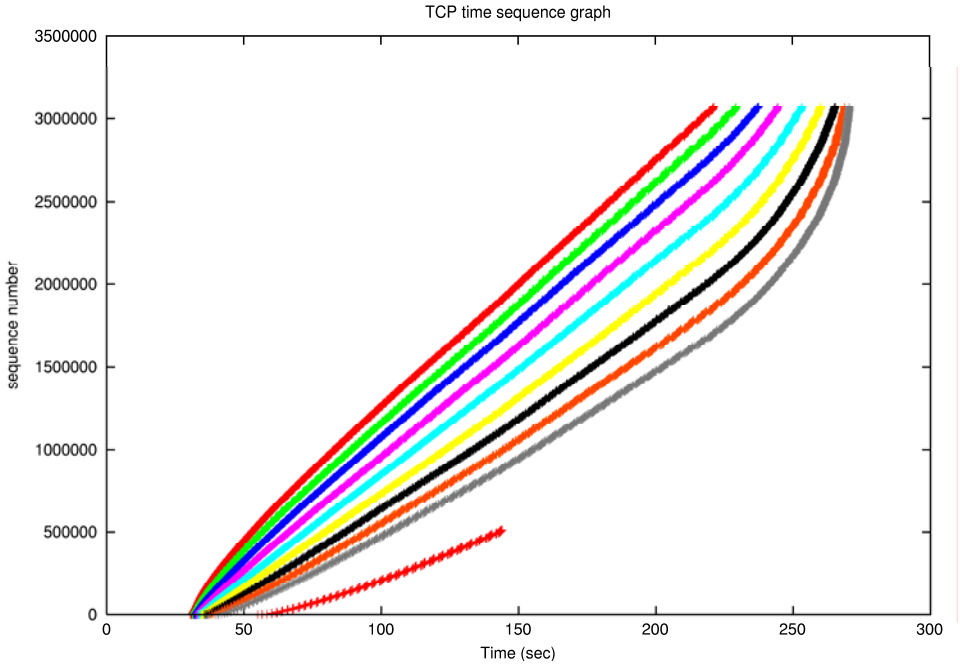
Πίνακας 4: Παράμετροι πειραμάτων

Bandwidth Mbps	Delay(RTT/2) msec	Error %
2-6-10	50-150-250	0-0.0001

5.1.1 Παράδειγμα με Bandwidth:2 Mbit/s ,Delay:50ms, Errors:0%

Η πρώτη περίπτωση που θα παρουσιάσουμε είναι σε δίκτυο με μικρό bandwidth(2Mbps) μικρή σχετικά καθυστέρηση (50 msec) και χωρίς λάθη.

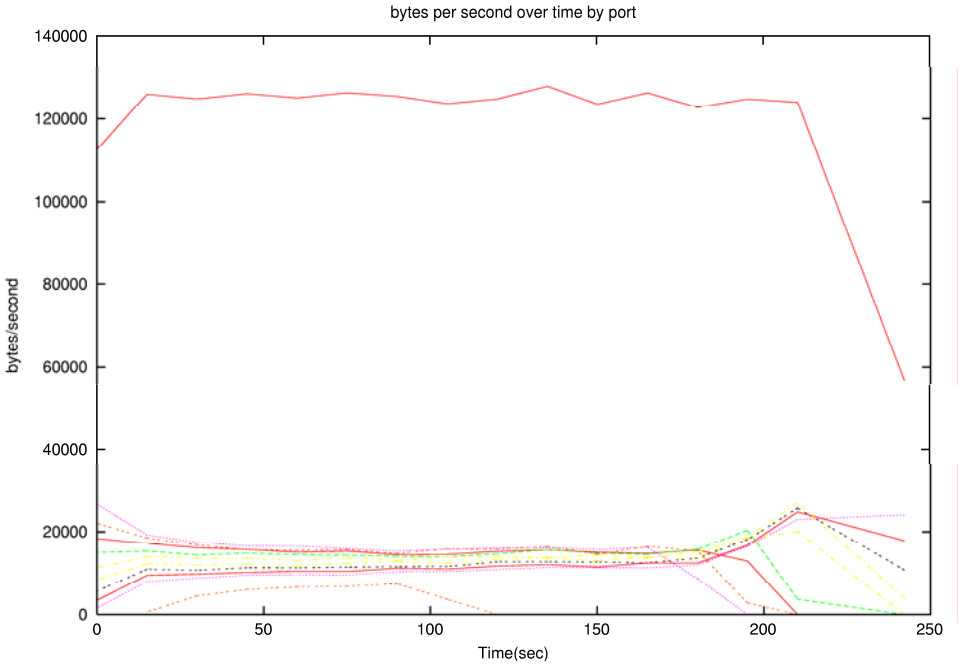
Για το πρωτόκολλο TCP παίρνουμε:



Σχήμα 39: Time sequence graph in TCP(BW:2Mbps,De:50ms,Er:0%)

Τα διάφορα χρώματα αποτελούν κάθε μια από τις συνδέσεις μεταφοράς αρχείου. Είναι όλες κανονικοποιημένες προς τον άξονα ψ και παρουσιάζονται όλες μαζί για ευκολία σύγκρισης. Η πιο μικρή με κόκκινο χρώμα αποτελεί το ενδιάμεσο αρχείο. Μπορούμε να παρατηρήσουμε την αλλαγή της κλίσης των διαφόρων ευθειών που αντιπροσωπεύει την ρυθμαπόδοση της κάθε σύνδεσης. Συμπερασματικά μπορούμε να πούμε ότι το αρχείο που μπαίνει στο δίκτυο ενδιάμεσα βρίσκει ένα δίκτυο «γεμάτο» δηλαδή ένα δίκτυο που όλοι οι πόροι του είναι κατειλημμένοι. Για το λόγο αυτό ξεκινά σιγά σιγά να στέλλει δεδομένα αυτή η σύνδεση μέχρι να πάρει το μερίδιο που της αντιστοιχεί στο δίκτυο. Το TCP είναι σχεδιασμένο ώστε να παρέχει δικαιοσύνη ανάμεσα στις ανεξάρτητες συνδέσεις του και δεν συμπεριφέρεται ανταγωνιστικά, όμως απαιτείται κάποιος χρόνος μέχρι να επέλθει αυτή η δίκαιη κατανομή.

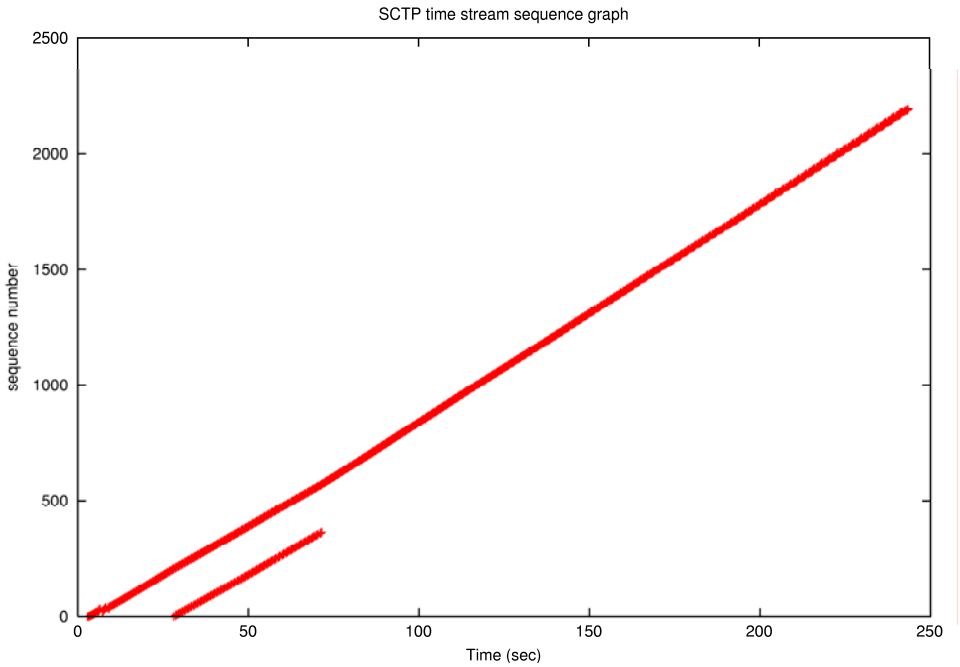
Αναλυτικότερα μπορούμε να παρατηρήσουμε το διάγραμμα της κίνησης σε κάθε σύνδεση:



Σχήμα 40: Κίνηση ανά σύνδεση(BW:2Mbps,De:50ms,Er:0%)

Στο διάγραμμα αυτό φαίνεται ξεκάθαρα η λογική του σεναρίου. Πολλά αρχεία μεταφέρονται ταυτόχρονα και περίπου στο εικοστό δευτερόλεπτο ξεκινά να μεταδίδεται το ενδιάμεσο αρχείο, όταν πλέων όλοι οι πόροι του δικτύου είναι κατειλημμένοι. Το αρχείο αυτό ξεκινά με χαμηλούς ρυθμούς μετάδοσης και ανεβαίνει σταδιακά μέχρι να πάρει το μερίδιο που του αντιστοιχεί. Η πάνω γραμμή παρουσιάζει την συνολική χρησιμοποίηση του δικτύου.

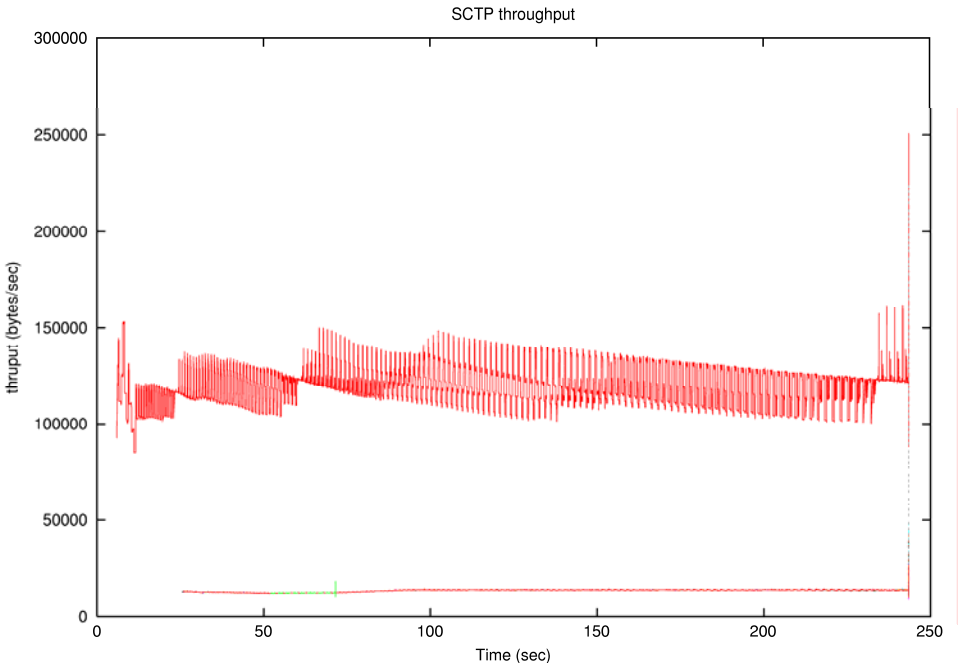
Στο πρωτόκολλο SCTP τα πράγματα είναι κάπως διαφορετικά.



Σχήμα 41: Time sequence graph in SCTP(BW:2Mbps,De:50ms,Er:0%)

Υπενθυμίζουμε για ακόμη μια φορά ότι η μεταφορά των αρχείων αυτών γίνεται σε μια σύνδεση αλλά σε ανεξάρτητες ροές. Αυτό σημαίνει ότι έχουμε ένα ενιαίο έλεγχο ροής και συμφόρησης αλλά ανεξάρτητη σειρά παράδοσης σε κάθε ροή.

Στο πιο πάνω διάγραμμα η μεγάλη γραμμή αποτελεί τα πακέτα που προέρχονται από τα πολλά αρχεία. Αυτά ξεκινούν ταυτόχρονα και αποστέλλονται με τον ίδιο ακριβώς ρυθμό. Ο αριθμός σειράς που παρουσιάζουμε είναι ανά ροή. Επειδή τα αρχεία αυτά στέλνονται ταυτόχρονα και με τον ίδιο ρυθμό δεν μπορούμε να διακρίνουμε πολλές ευθείες όπως την περίπτωση του TCP. Ακόμα μπορούμε να δούμε το ενδιάμεσο αρχείο ότι δεν έχει πιο μικρή κλίση αλλά ξεκινά και αυτό με όση ταχύτητα του ισοδυναμεί. Μια λεπτομέρεια είναι ότι η κλίση τις πάνω γραμμής πέφτει ελαφρά όταν ξεκινά το ενδιάμεσο αρχείο και επανέρχεται όταν αυτό τελειώσει την μετάδοση αφού το ενδιάμεσο αρχείο καταλαμβάνει και αυτό πόρους από το δίκτυο. Το σημαντικό είναι ότι η αυτή η μεταβολή γίνεται ακαριαία και όχι σταδιακά όπως παρατηρήσαμε στο TCP.



Σχήμα 42: Ρυθμαπόδοση σύνδεσης SCTP(BW:2Mbps,De:50ms,Er:0%)

Στο διάγραμμα ρυθμαπόδοσης διακρίνεται ξεκάθαρα το ενδιάμεσο αρχείο να καταλαμβάνει αμέσως την ταχύτητα που του αναλογεί χωρίς να περνά την διαδικασία αργής αρχής και σταδιακής αύξησης της ρυθμαπόδοσης του, γι'αυτό και οι καμπύλες ταυτίζονται.

Για να δούμε πως επηρεάζει αυτή η διαφορετική συμπεριφορά την μεταφορά των αρχείων μπορούμε να δούμε την μέση ρυθμαπόδοση για κάθε ένα από αυτά.

Πίνακας 5: Goodput ανά αρχείο (BW2Mbps,Dl:50ms,Er:0%)

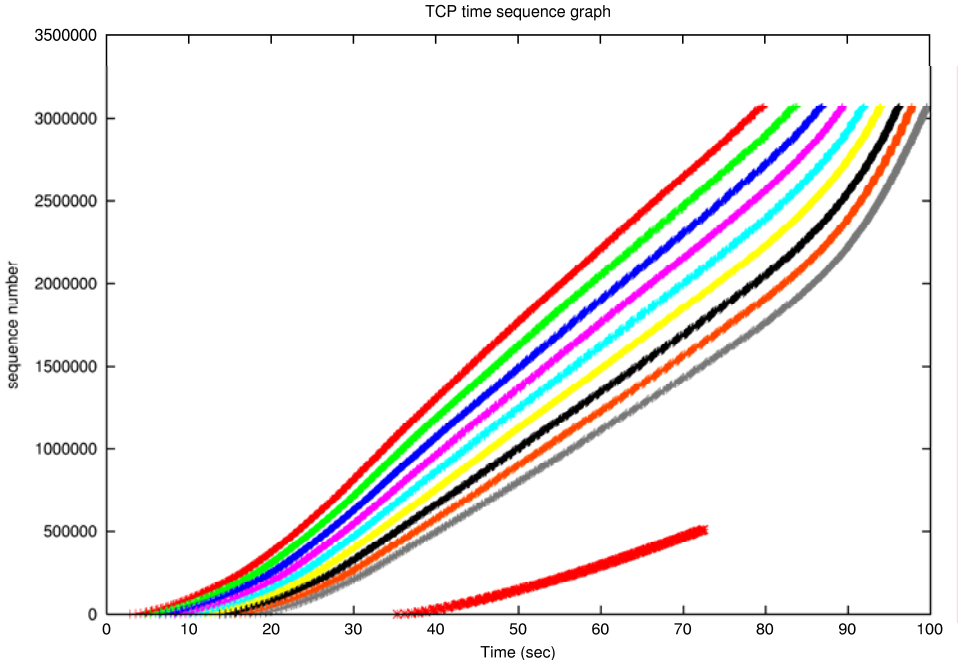
A/A	SCTP (Bps)	TCP (Bps)
1	12845	15948
2	12838.25	15359
3	12840.12	14812
4	12850.63	14400
5	12860.42	13896
6	12858.76	13543
7	12847.16	13333
8	12847.97	13248
9	12854.81	13286

10	11745.22	5610

Βλέπουμε ότι το ενδιάμεσο αρχείο που είναι το τελευταίο στον πίνακα έχει μεγάλη διαφορά από το ένα πρωτόκολλο στο άλλο, και ότι η αργή εισαγωγή του στο TCP στοιχίζει πολύ στη ρυθμαπόδοση του.

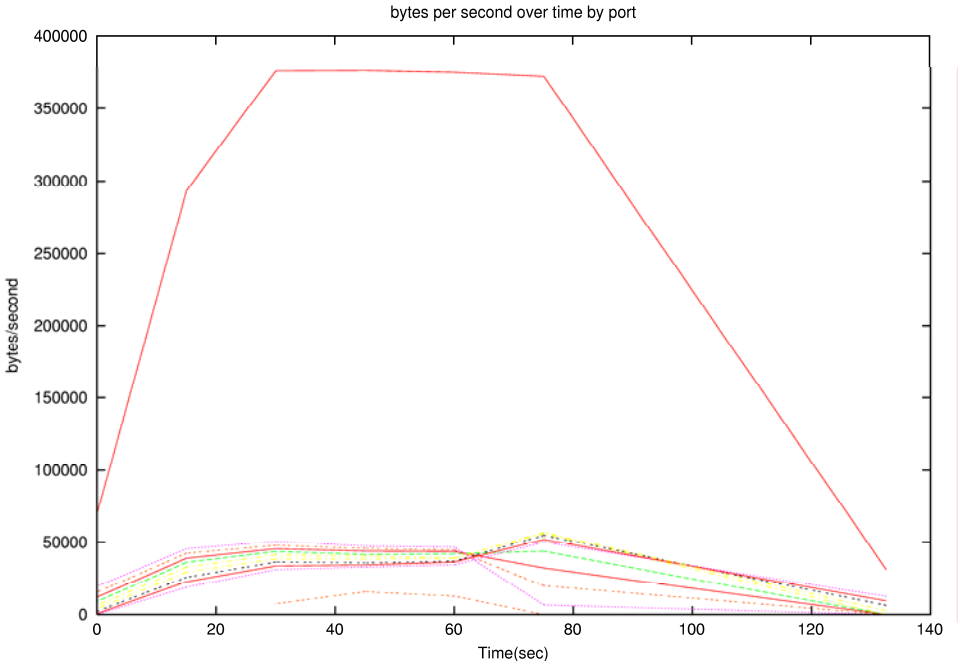
5.1.2 Παράδειγμα με Bandwidth:6 Mbit/s ,Delay:150ms, Errors:0%

Όσο αυξάνεται η καθυστέρηση και το bandwidth του δικτύου τόσο αυξάνεται και η διαφορά στην απόδοση. Στο επόμενο παράδειγμα παρουσιάζουμε ένα δίκτυο 6Mbps με καθυστέρηση 150msec και πάλι χωρίς απώλειες.



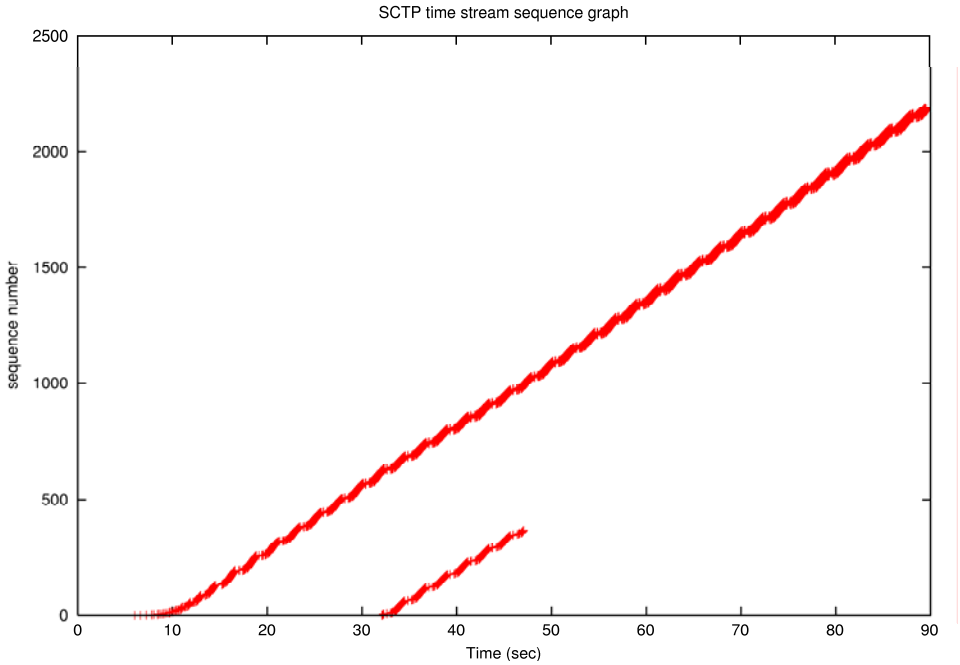
Σχήμα 43: Time sequence graph in TCP(BW:6Mbps,De:150ms,Er:0%)

Η συμπεριφορά είναι η ίδια όμως η κλίση του ενδιάμεσου αρχείου είναι ακόμα πιο μικρή. Τα αρχεία παρασκηνίου δεν ξεκινούν ταυτόχρονα ακριβώς γιατί αυτό είναι πρακτικά αδύνατο. Επειδή κάθε αρχείο πρέπει να δημιουργήσει δική του σύνδεση η διαδικασία αυτή χρειάζεται κάποιο χρόνο. Στο SCTP τα πράγματα δεν είναι τόσο πολύπλοκα μιας και η σύνδεση είναι κοινή, έτσι φαίνονται να ξεκινούν ακριβώς ταυτόχρονα.

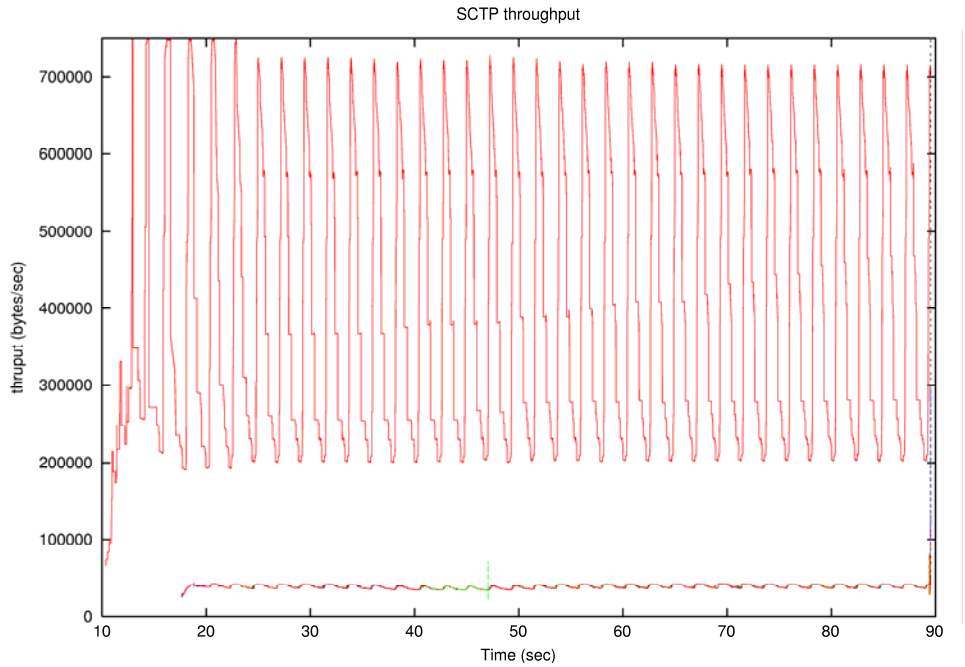


Σχήμα 44: Κίνηση ανά σύνδεση(BW:6Mbps,De:150ms,Er:0%)

Οι αντίστοιχες γραφικές στο SCTP:



Σχήμα 45: Time sequence graph in SCTP(BW:6Mbps,De:150ms,Er:0%)



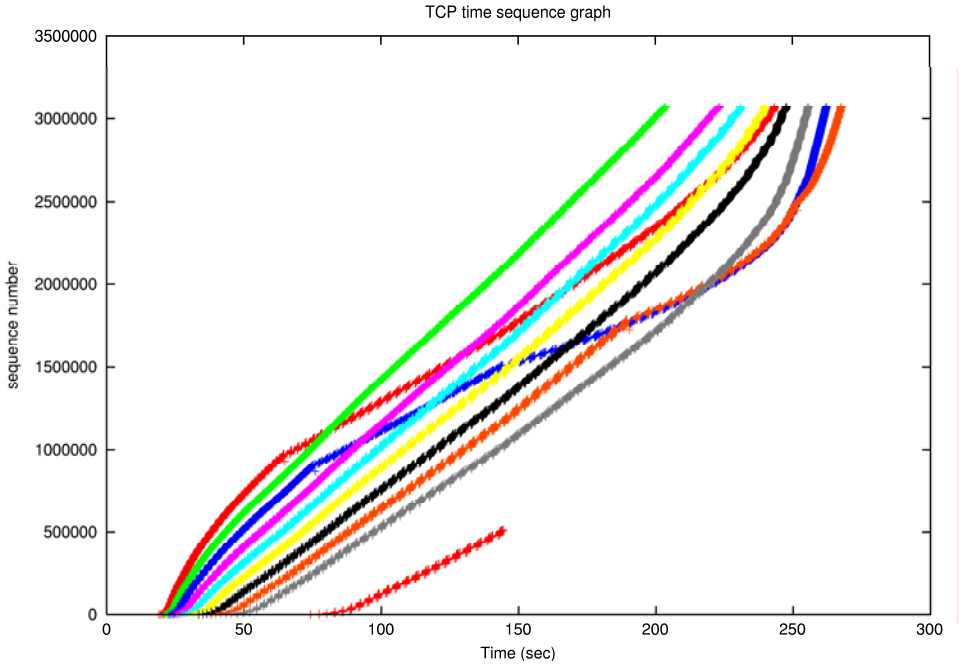
Σχήμα 46: Ρυθμαπόδοση σύνδεσης SCTP(BW:6Mbps,De:150ms,Er:0%)

Αν συγκρίνουμε και πάλι τη μέση ρυθμαπόδοση για το ενδιάμεσο αρχείο έχουμε 12349Bps στο TCP ενώ 34252Bps SCTP και είναι μεγαλύτερη η διαφορά τους από το προηγούμενο παράδειγμα.

Σε όλα τα ποιό πάνω αφήσαμε εκτός τον παράγοντα των απωλειών μιας και τα λάθη ήταν 0% και όπως έχουμε ήδη αναφέρει δεν υπάρχουν μη ελεγχόμενα λάθη λόγο της χρήσης ενός και μόνο υπολογιστή και όχι φυσικού δικτύου. Τα λάθη σε όλες τις περιπτώσεις λειτουργούν αρνητικά στην απόδοση και τον δύο δικτύων, όμως το γεγονός ότι κάνουμε χρήση των ροών του SCTP και χρησιμοποιούμε ένα ενιαίο έλεγχο συμφόρησης έχει κάποιο κόστος. Όταν παρουσιάζονται απώλειες στη μετάδοση, σε όποια ροή και να είναι αυτές επηρεάζουν το ρυθμό μετάδοσης σε όλες τις ροές αφού το παράθυρο είναι κοινό. Στο TCP όμως αφού κάθε σύνδεση έχει δικό της έλεγχο συμφόρησης, επηρεάζεται μόνο από λάθη που εμφανίζονται σε αυτή και μόνο.

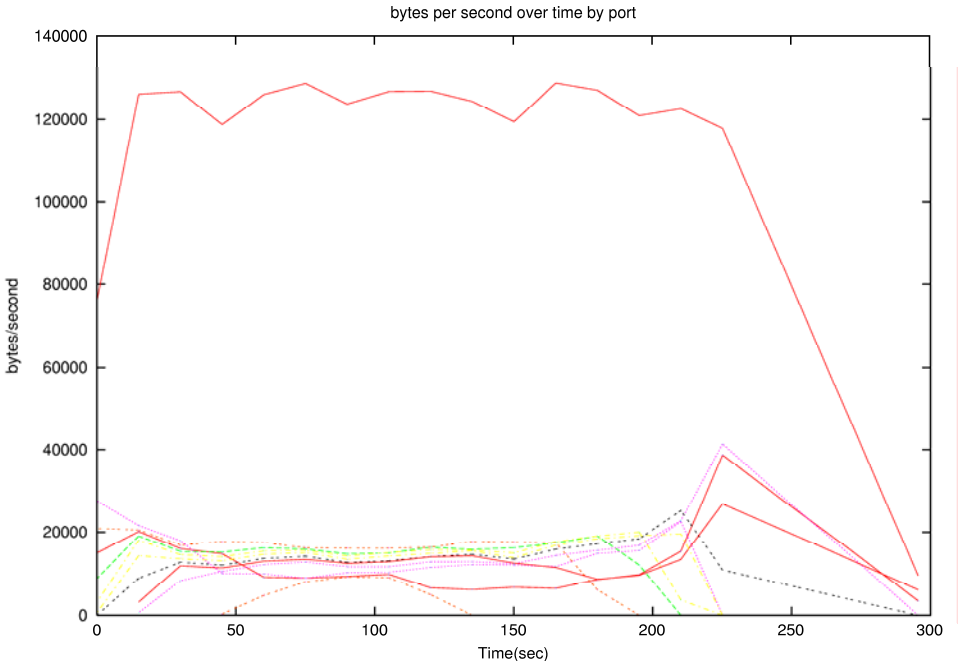
5.1.3 Παράδειγμα με Bandwidth:2 Mbit/s ,Delay:150ms, Errors:10^{-4}

Το επόμενο παράδειγμα είναι σε δίκτυο με bandwidth 2Mbps καθυστέρηση 150ms και error rate της τάξης 10^{-4}.



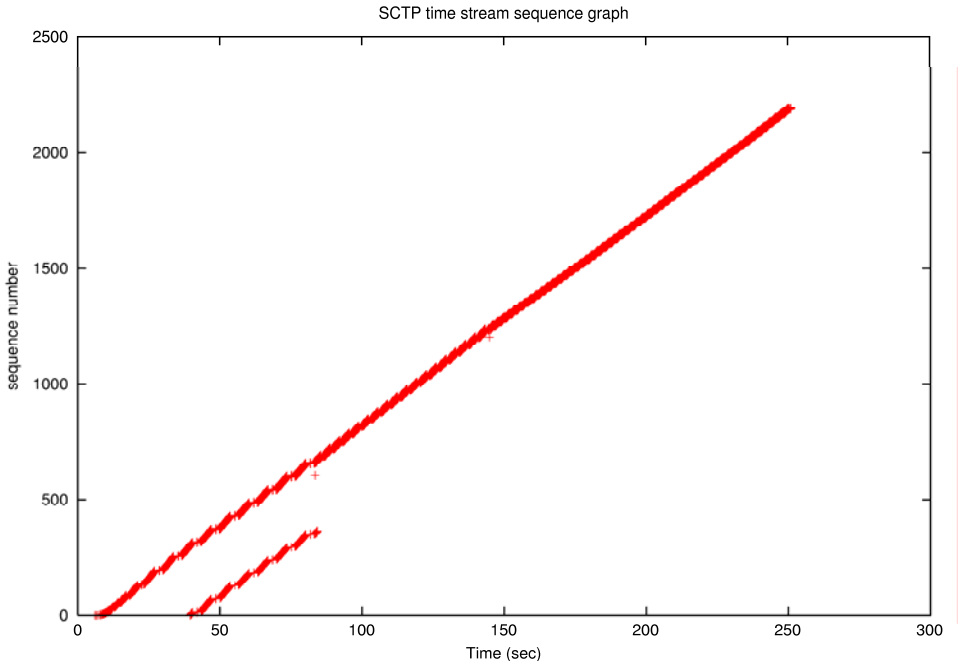
Σχήμα 47: Time sequence graph in TCP(BW:2Mbps,De:150ms,Er:0.0001%)

Μπορούμε να διακρίνουμε τις συνδέσεις στις οποίες έχει παρουσιαστεί κάποια απώλεια, όπως η κόκκινη και η μπλε σύνδεση στις οποίες φαίνεται μάλιστα και η αποστολή του χαμένου πακέτου (εκτός ευθείας σημείο). Από το διάγραμμα κίνησης ανά θύρα μπορούμε να δούμε ότι η κίνηση σε σύνδεση που δεν έχει γίνει κάποιο λάθος δεν επηρεάζεται.



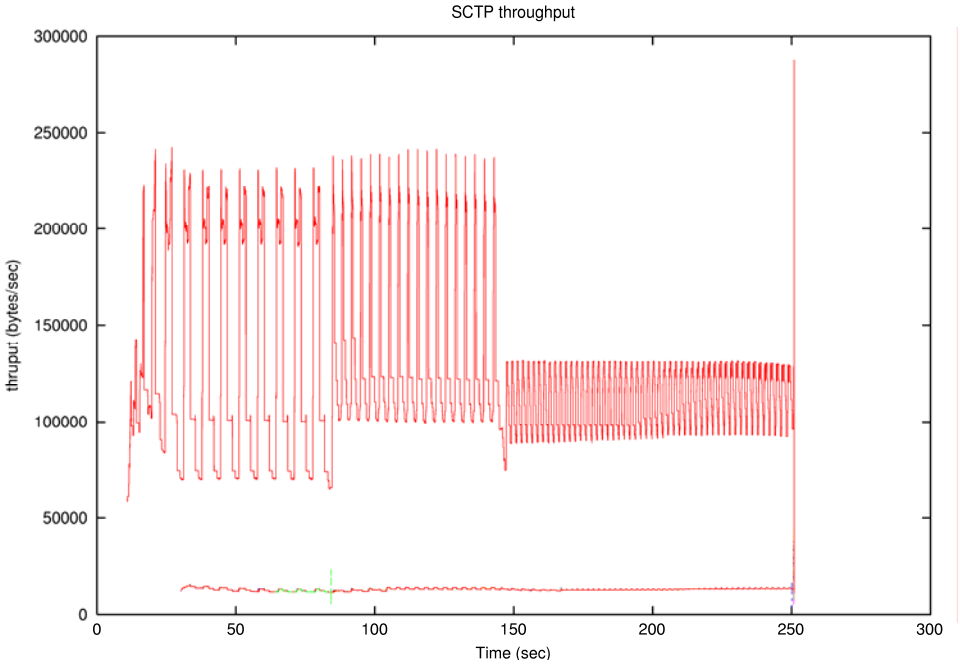
Σχήμα 48: Κίνηση ανά σύνδεση(BW:2Mbps,De:150ms,Er:0.0001%)

Στο πρωτόκολλο SCTP μπορούμε να δούμε την επιρροή του ενιαίου ελέγχου.



Σχήμα 49: Time sequence gragh in SCTP(BW:2Mbps,De:150ms,Er:0.0001%)

Η κλίση αλλάζει συνολικά σε μια απώλεια χωρίς να κάνει διακρίσεις ανά ροή.



Σχήμα 50: Ρυθμαπόδοση σύνδεσης SCTP(BW:2Mbps,De:150ms,Er:0.0001%)

Πίνακας 6: Goodput ανά αρχείο (BW:2Mbps,Dl:150ms,Er:0.0001%)

A/A	SCTP (Bps)	TCP (Bps)
1	12552.53	13553
2	12590.13	16432
3	12554.12	12771
4	12578.19	15169
5	12580.75	14792
6	12583.18	14405
7	12586.74	14179
8	12583.34	13253
9	12582.87	14292
10	11445.96	6685

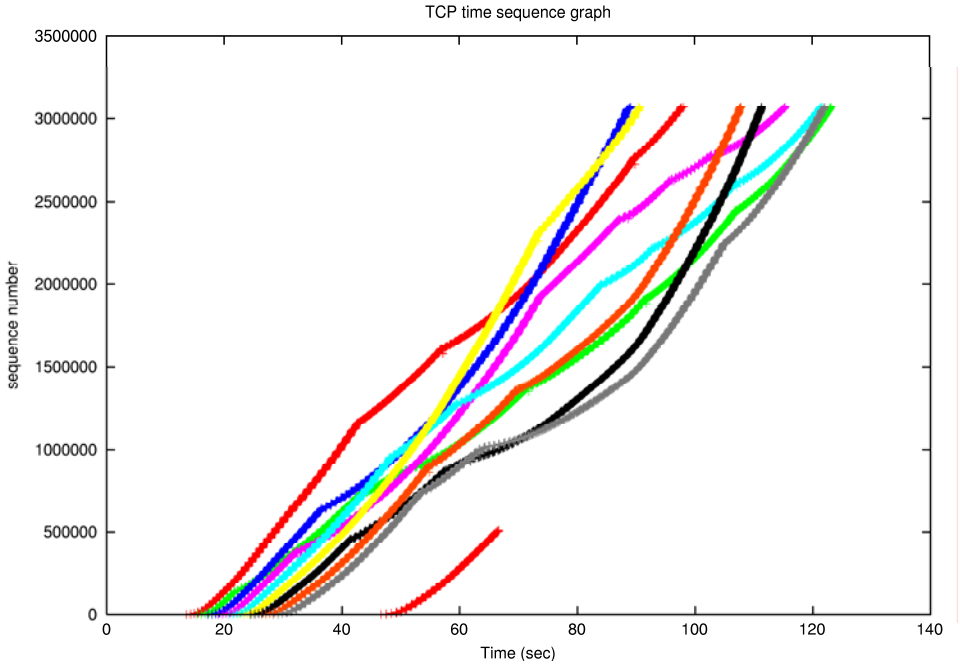
Παρατηρώντας αναλυτικά τον πίνακα μέσης ρυθμαπόδοσης, όλες οι συνδέσεις στο SCTP έχουν παραπλήσιο goodput ενώ στο TCP υπάρχει διακύμανση. Παρόλα αυτά το ενδιάμεσο αρχείο συνεχίζει να έχει σημαντικό προβάδισμα αλλά το ποσοστό απωλειών είναι σε χαμηλά επίπεδα.

5.1.4 Παράδειγμα με Bandwidth:6 Mbit/s ,Delay:150ms, Errors:10^{-3}

Μετρήσεις με πιο μεγάλες απώλειες τις τάξεως του 10^{-3} και περισσότερα έδειξαν ότι το SCTP δεν είναι πλέων ανταγωνιστικό σε μεταφορές με πολλές ροές αντί πολλές συνδέσεις και η απόδοση πέφτει κατά μεγάλο ποσοστό.

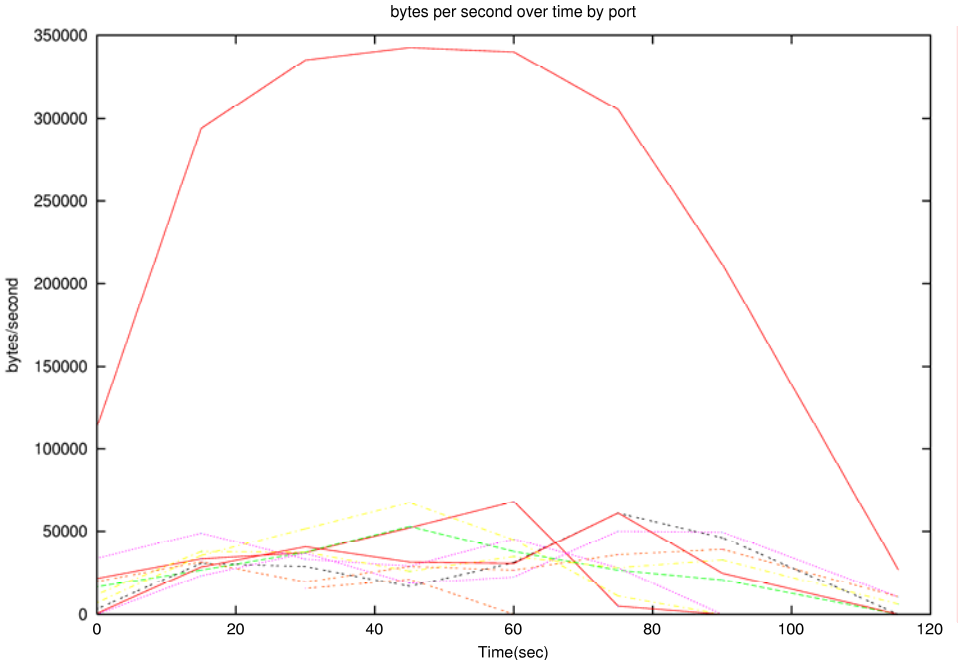
Θα παρουσιάσουμε αναλυτικά ένα παράδειγμα με απώλειες 0.001% για να δείξουμε αυτή την πτώση του SCTP αλλά δεν θα εμβαθύνουμε περισσότερο στο ζήτημα αυτό. Δεδομένου ότι το API για το πρωτόκολλο που χρησιμοποιούμε είναι ακόμα υπό συνεχή ανάπτυξη και βελτίωση και το γεγονός ότι η βιβλιοθήκη τρέχει στο χώρο του χρήστη και όχι στο χώρο του πυρήνα, δίνουν περιθώρια σε προγράμματα που χρησιμοποιούν το πρωτόκολλο αυτό να γίνουν ακόμα πιο γρήγορα από αυτά που δείχνουν οι δικές μας μετρήσεις.

Για δίκτυο με χαρακτηριστικά BW:6Mbps DELAY:150msec ERROR:0.001%



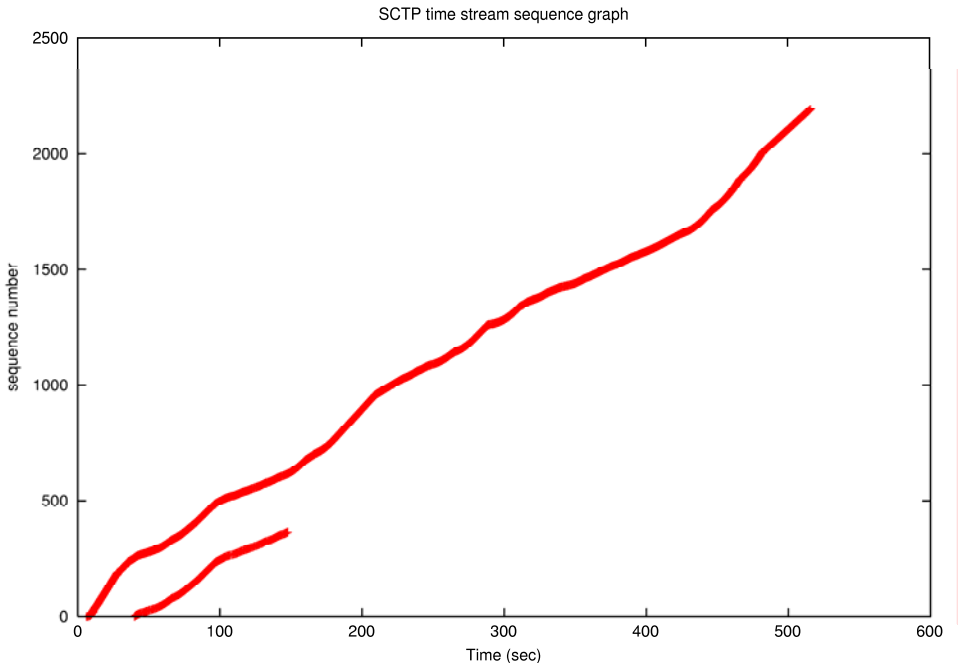
Σχήμα 51: Time sequence graph in TCP(BW:6Mbps,De:150ms,Er:0.001%)

Οι ανωμαλίες στην μετάδοση δείχνουν τις απώλειες στο δίκτυο.

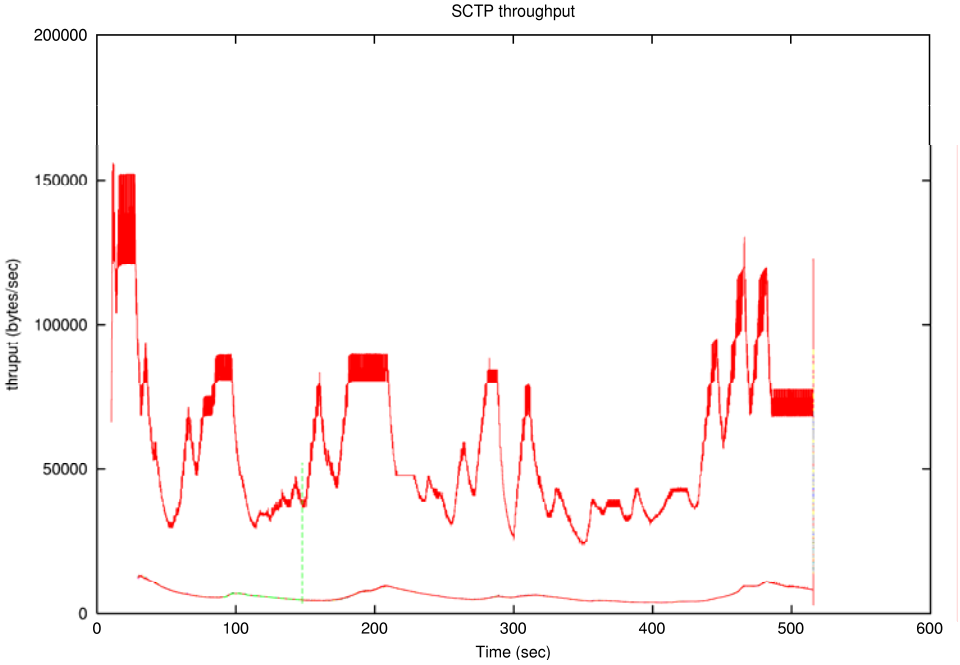


Σχήμα 52: Κίνηση ανά σύνδεση(BW:6Mbps,De:150ms,Er:0.001%)

Στο SCTP οι αντίστοιχες γραφικές διαμορφώνονται ως εξής:



Σχήμα 53: Time sequence graph in SCTP(BW:6Mbps,De:150ms,Er:0.001%)

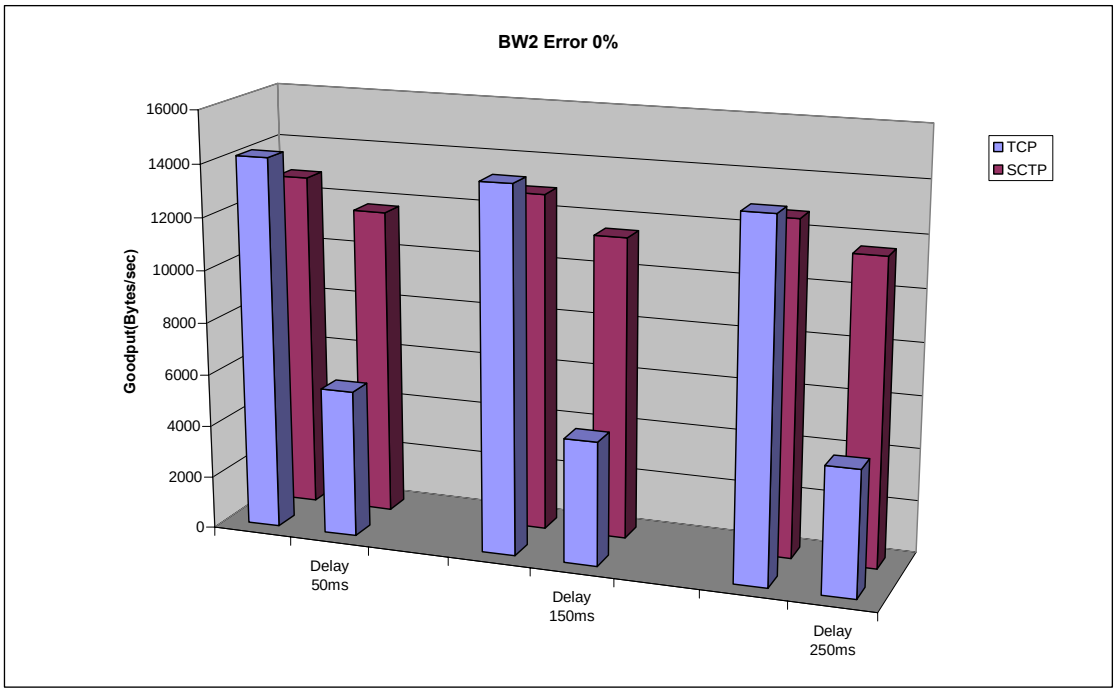


Σχήμα 54: Ρυθμαπόδοση σύνδεσης SCTP(BW:6Mbps,De:150ms,Er:0.001%)

Από τον συνολικό χρόνο που χρειάστηκε ολόκληρη η μεταφορά είναι εμφανής η καθυστέρηση που παρουσιάζεται στο SCTP, γεγονός που δείχνει ότι η χρήση ενιαίου ελέγχου συμφόρησης δεν είναι πάντα κερδοφόρα.

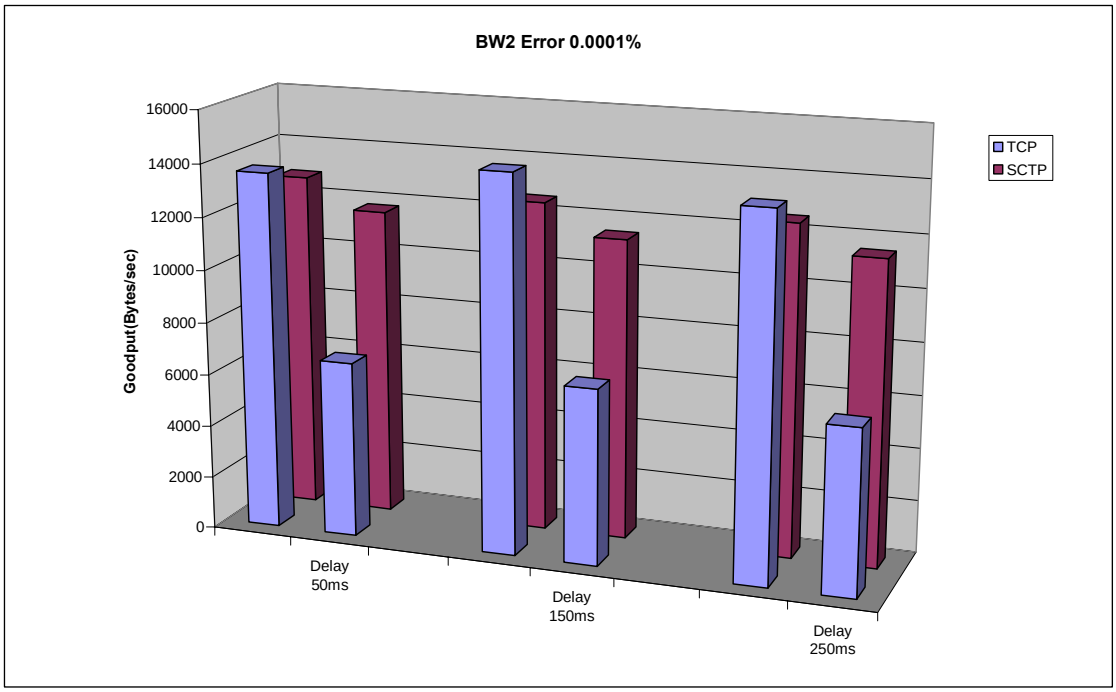
5.2 Συνολικά αποτελέσματα μεταφοράς

Στις γραφικές που ακολουθούν παρουσιάζουμε τον μέσο όρο ρυθμαπόδοσης για τις εννέα συνδέσεις και ξεχωριστά την μέση ρυθμαπόδοση για την ενδιάμεση σύνδεση.



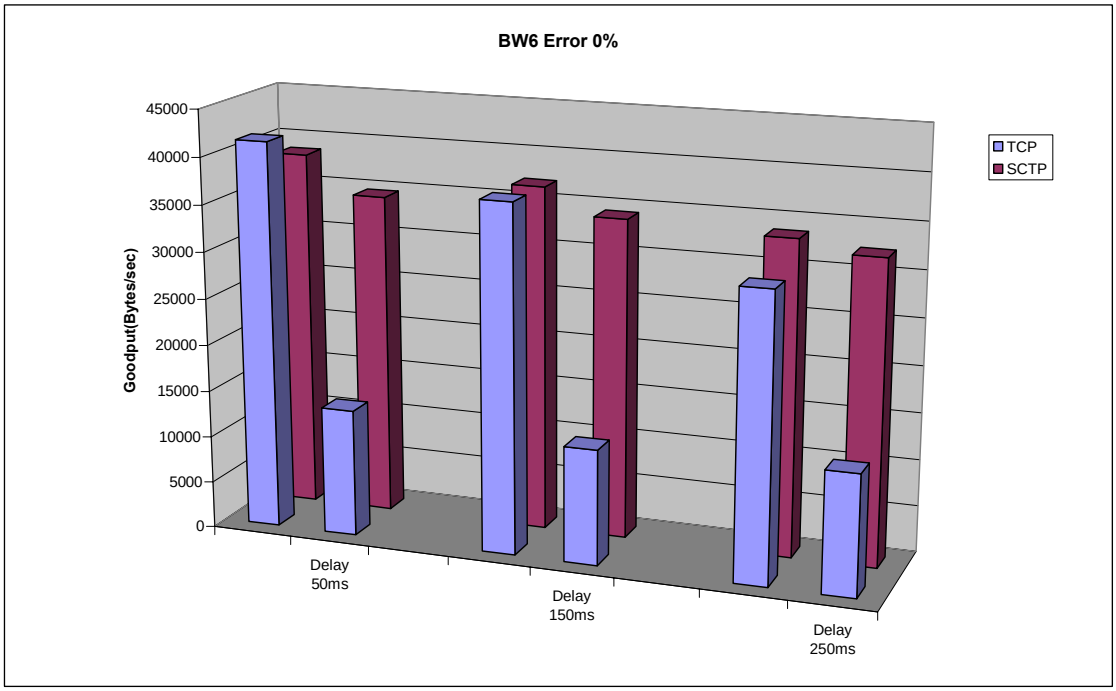
Σχήμα 55: Συνοπτικά αποτελέσματα(BW:2Mbps,Er:0%,De:50-150-250ms)

Η πρώτη στήλη κάθε δυάδας παρουσιάζει τον μέσο όρο ρυθμαπόδοσης για τις μεταφορές των αρχείων παρασκηνίου και η δεύτερη για το ενδιάμεσο αρχείο. Παρατηρούμε ότι στο SCTP η απόδοση δεν επηρεάζεται από την καθυστέρηση στο μοναδικό αρχείο γιατί όταν ξεκινά χρησιμοποιεί ήδη το παράθυρο που έχει διαμορφωθεί. Αντίθετα στο TCP υπάρχει μείωση γιατί χρειάζεται περισσότερο χρόνο να ανέβει στο ποσοστό που δικαιούται.



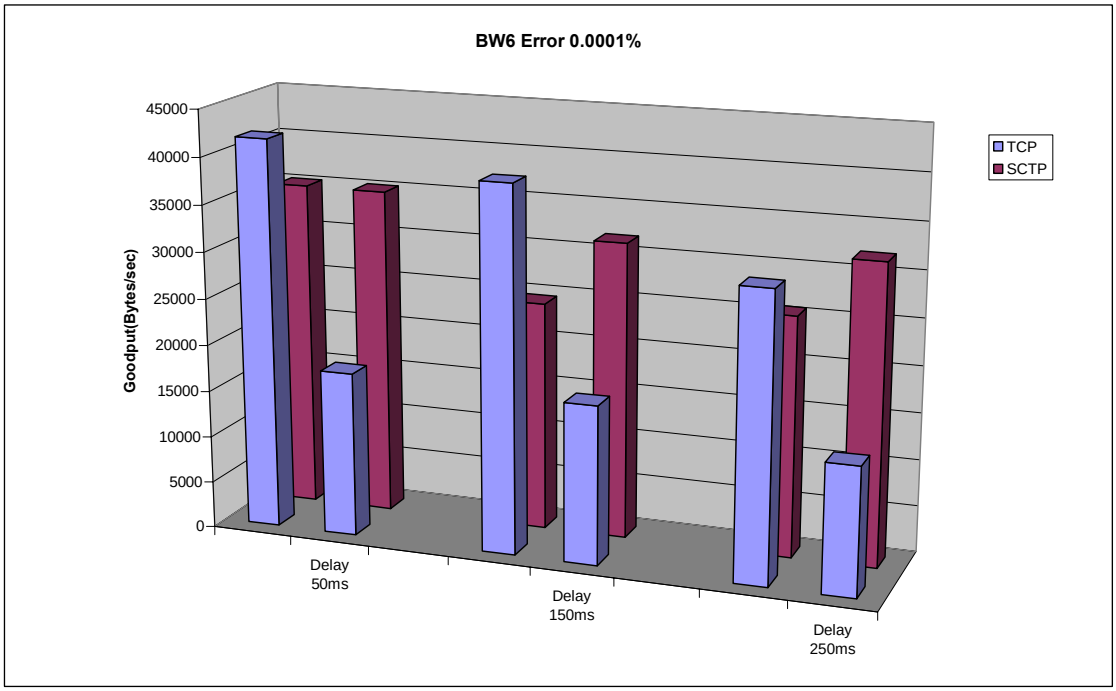
Σχήμα 56: Συνοπτικά αποτελέσματα(BW:2Mbps,Er:0.0001%,De:50-150-250ms)

Τα λάθη όταν αυτά γίνονται με μικρό ποσοστό δεν επηρεάζουν ιδιαίτερα την απόδοση αν και αυξάνουν ελαφρός τη διαφορά στο μέσο όρο των πολλών αρχείων.



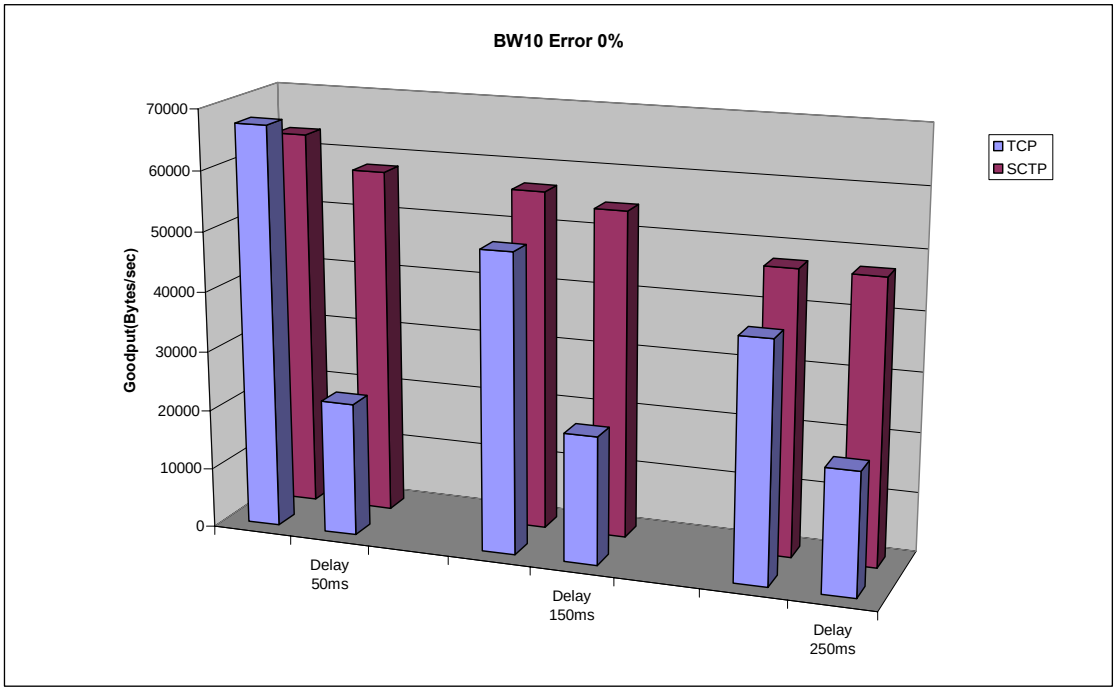
Σχήμα 57: Συνοπτικά αποτελέσματα(BW:6Mbps,Er:0%,De:50-150-250ms)

Η διαφορά αυξάνεται ακόμα περισσότερο όσο αυξάνεται και η χωρητικότητα του δικτύου, και παρατηρούμε ότι με μεγάλη καθυστέρηση το SCTP ξεπερνά σε ταχύτητα το TCP ακόμα και στα πολλά αρχεία, δηλαδή όχι μόνο στο χρόνο μεταφοράς του ενδιάμεσου αλλά και στον συνολικό χρόνο μετάδοσης και των υπολοίπων αρχείων. Εξηγήσαμε ότι το ενδιάμεσο αρχείο καθυστερεί στην προσπάθεια του να πάρει το ποσοστό που δικαιούται στο δίκτυο και αυτό εξηγεί την εμφανή διαφορά του στις δύο περιπτώσεις. Τα πολλά αρχεία με την ίδια λογική όταν τελειώσει την μετάδοση του το ενδιάμεσο αρχείο απαιτείται κάποιος χρόνος για να καλύψουν τους πόρους που αφήνει ελεύθερους το αρχείο που έχει τελειώσει. Επειδή τα αρχεία είναι πολλά η διαφορά αυτή δεν είναι εμφανής σε δίκτυα με μικρή χωρητικότητα. Στην περίπτωση του SCTP η επιστροφή των πόρων του δικτύου από το τέλος του ενδιάμεσου αρχείου γίνεται άμεσα και αυτό εξηγεί το κέρδος που παρουσιάζεται στην τελευταία δυάδα και που φαίνεται ακόμα πιο έντονα σε δίκτυο με ακόμη μεγαλύτερη χωρητικότητα.



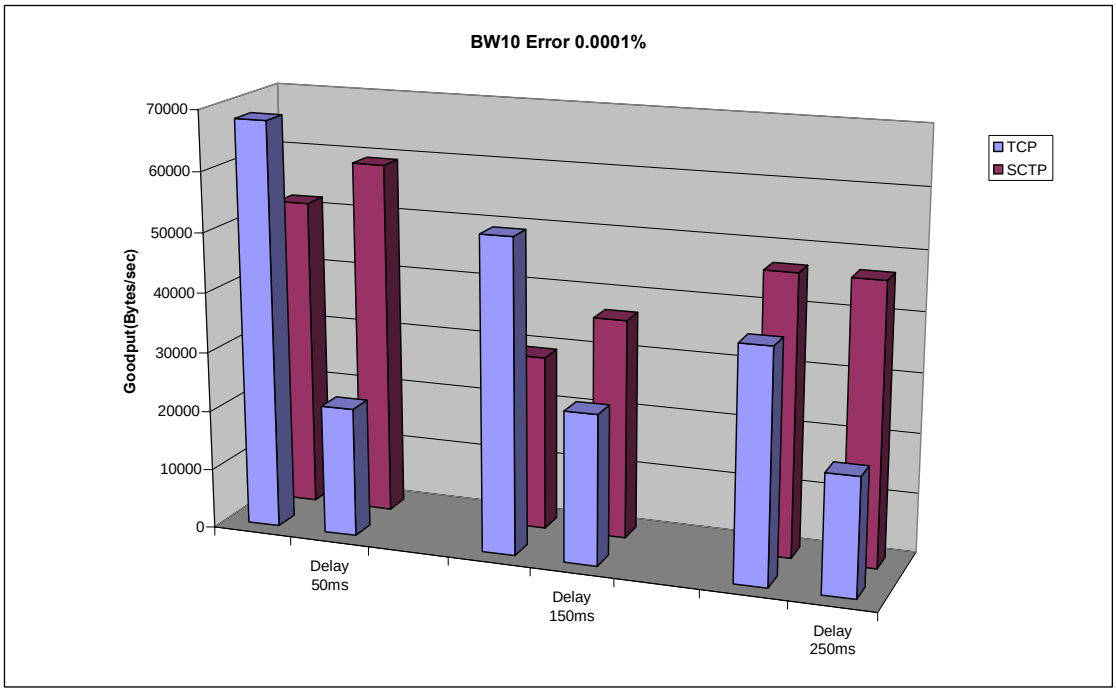
Σχήμα 58: Συνοπτικά αποτελέσματα(BW:6Mbps,Er:0.0001%,De:50-150-250ms)

Αυτό δεν συμβαίνει με λάθη στη μετάδοση γιατί το κέρδος των πολλαπλών ροών περιορίζεται από την μείωση του μοναδικού παραθύρου από τα λάθη.



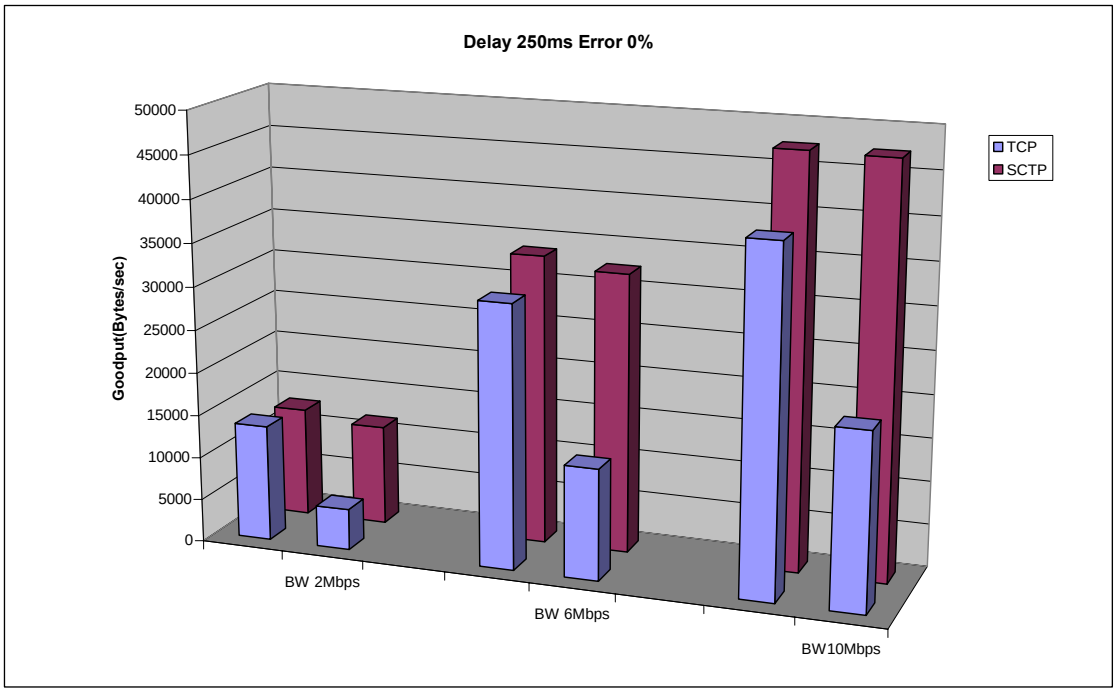
Σχήμα 59: Συνοπτικά αποτελέσματα(BW:10Mbps,Er:0%,De:50-150-250ms)

Σε αυτή την περίπτωση έχουμε την πιο έντονη επικράτηση της περίπτωσης με το SCTP το οποίο με καθυστέρηση 150 και 250 χιλιοστά του δευτερολέπτου έχει μεγάλο κέρδος στην απόδοση και για το ενδιάμεσο αλλά και για τα άλλα αρχεία.



Σχήμα 60: Συνοπτικά αποτελέσματα(BW:10Mbps,Er:0.0001%,De:50-150-250ms)

Θα παρουσιάσουμε μαζί την μεταφορά χωρίς απώλειες και με καθυστέρηση 250ms ανά διαδρομή για να δούμε πως αυξάνεται το όφελος στην περίπτωση των πολλαπλών ροών.



Σχήμα 61: Συνοπτικά αποτελέσματα(BW:2-6-10Mbps,Er:0%,De:250ms)

6. ΜΕΤΑΦΟΡΑ ΠΟΛΛΩΝ ΑΡΧΕΙΩΝ (ΣΕΝΑΡΙΟ Β)

Για το δεύτερο σενάριο θέλουμε ένα αρχείο να ξεκινά όταν τα υπόλοιπα εννέα έχουν ήδη καταλάβει τους πόρους του δικτύου αλλά να μην τελειώνει πριν από αυτά αλλά ένα τυχαίο διάστημα μετά. Λογική αυτού του σεναρίου είναι να παρατηρήσουμε πόσος χρόνος χάνεται μέχρι το μοναδικό αρχείο που συνεχίζει να μεταδίδεται να «καταλάβει» ότι τα άλλα αρχεία έχουν τελειώσει και να μπορέσει να χρησιμοποιήσει το μέρος του δικτύου που τώρα είναι διαθέσιμο. Αυτά φυσικά συμβαίνουν στο TCP με τις πολλές συνδέσεις, μιας και στο SCTP όπως και προηγουμένως αυτό γίνεται στιγμιαία.

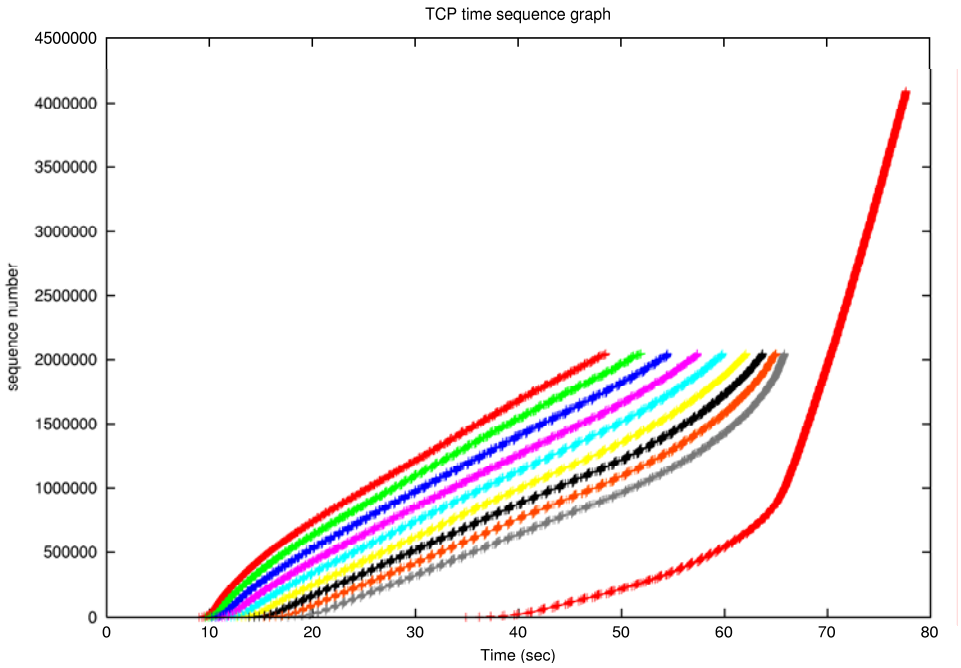
6.1 Αναλυτικά αποτελέσματα μετρήσεων

Όπως και στο προηγούμενο κεφάλαιο θα παρουσιάσουμε μερικά χαρακτηριστικά αποτελέσματα με αναλυτικές γραφικές που να δείχνουν τι ακριβώς συμβαίνει κατά την μετάδοση των αρχείων στο σενάριο αυτό.

6.1.1 Παράδειγμα με Bandwidth:6Mbit/s ,Delay:50ms, Errors:0%

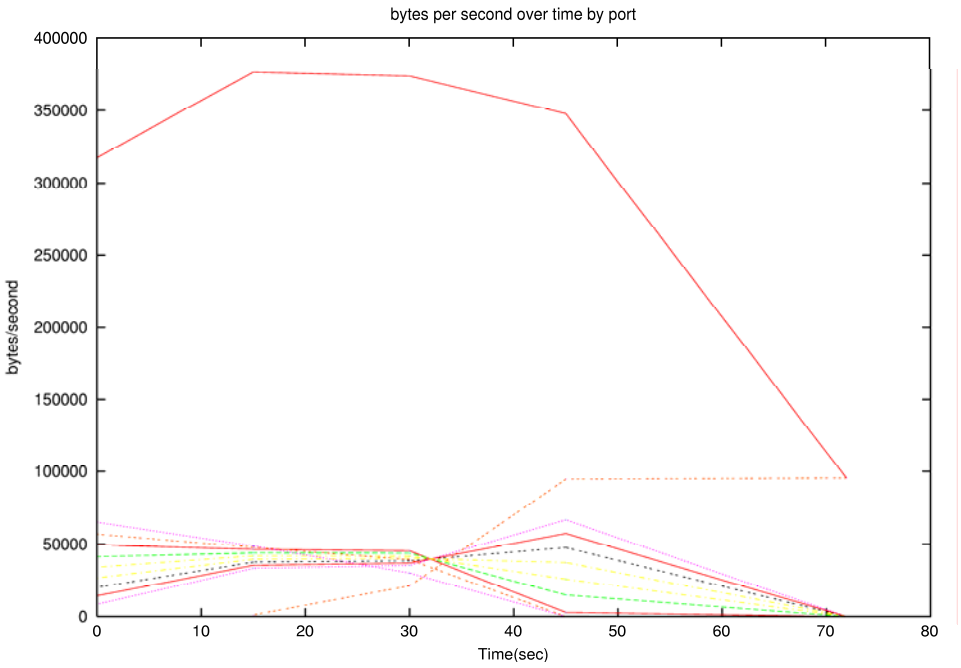
Στο πρώτο αναλυτικό παράδειγμα η μετάδοση γίνεται σε δίκτυο με bandwidth 6Mbps με καθυστέρηση 50 msec ανά κατεύθυνση και χωρίς λάθη.

Παρουσιάζουμε τις ίδιες γραφικές όπως και πριν, έτσι στο TCP παίρνουμε:



Σχήμα 62: Time sequence graph in TCP(BW:6Mbps,De:50ms,Er:0%)

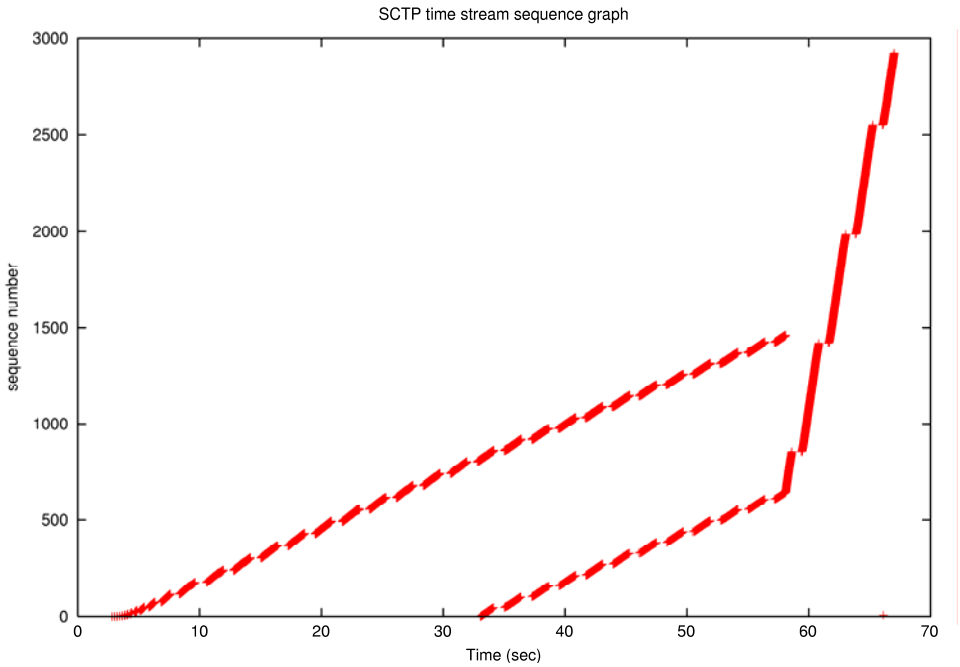
Το αρχείο που έχει το πιο μεγάλο ενδιαφέρον είναι το δεξιά το οποίο ξεκινά την μετάδοση του περίπου στο 35 δευτερόλεπτο. Όταν τα υπόλοιπα αρχεία έχουν τελειώσει δηλαδή περίπου στο 65 δευτερόλεπτο παρατηρείται μια αύξηση της κλίσης του τελευταίου αρχείου που αντιστοιχεί σε αύξηση της ταχύτητας μετάδοσης. Όλο το ενδιαφέρον επικεντρώνεται στο πόσο γρήγορα μπορεί να εκμεταλλευτεί το διαθέσιμο εύρος ζώνης που προκύπτει.



Σχήμα 63: Κίνηση ανά σύνδεση(BW:6Mbps,De:50ms,Er:0%)

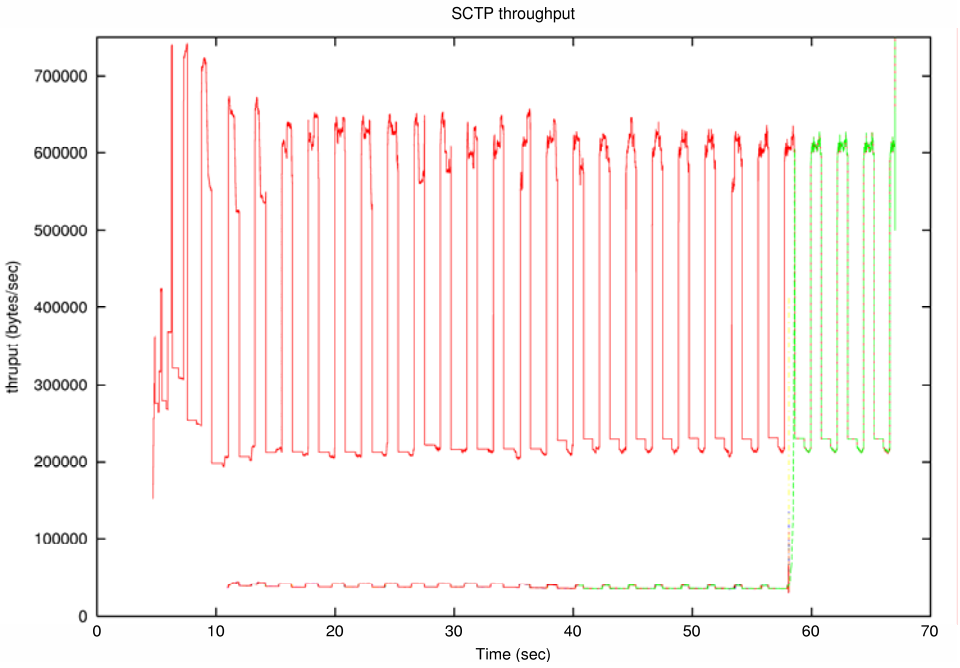
Από το διάγραμμα κίνησης μπορούμε να δούμε ότι η χρησιμοποίηση του δικτύου από το αρχείο είναι αυξητική άλλά ο ρυθμός αύξησης δεν είναι ιδιαίτερα γρήγορος, και η συνολική χρησιμοποίηση ξεκινά να πέφτει από την στιγμή που τελειώνουν τα πολλά αρχεία.

Μπορούμε να δούμε την αντίθεση στις γραφικές από το SCTP:



Σχήμα 64: Time sequence graph in SCTP(BW:6Mbps,De:50ms,Er:0%)

Για το δεξιά αρχείο η αλλαγή της κλίσης δεν είναι σταδιακή όπως πριν για να σχηματίζει καμπύλη αλλά είναι στιγμιαία και για αυτό σχηματίζει γωνία.

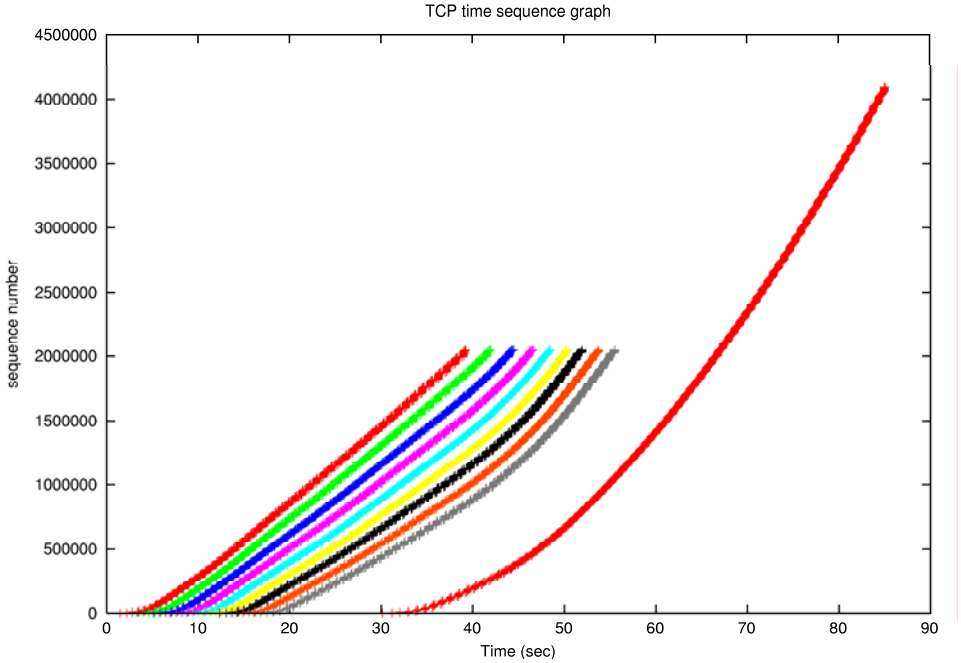


Σχήμα 65: Ρυθμαπόδοση σύνδεσης SCTP(BW:6Mbps,De:50ms,Er:0%)

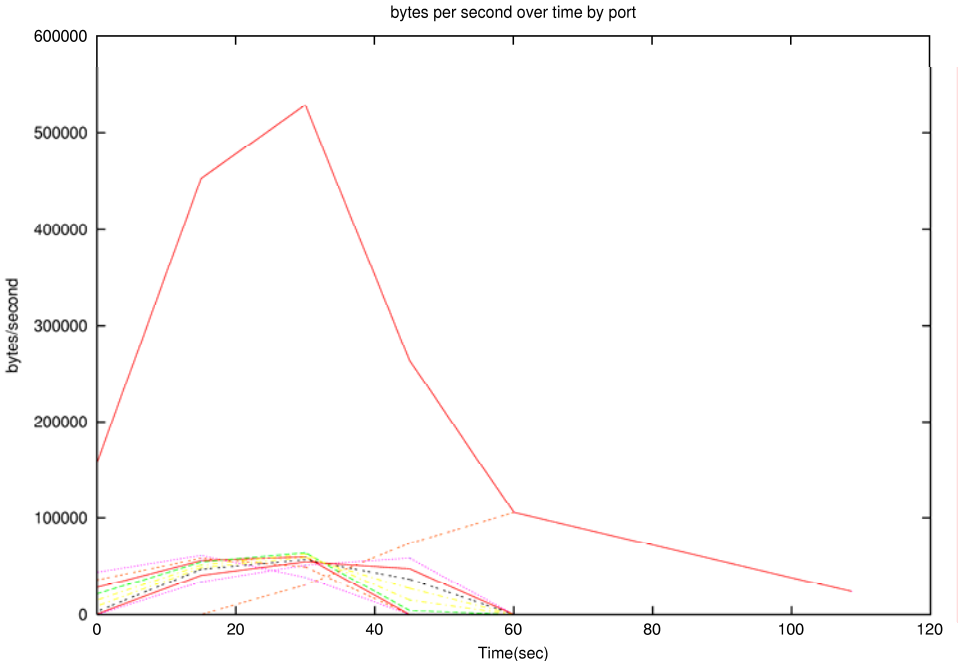
Σε αυτό το διάγραμμα φαίνεται η αυτόματη προσαρμογή του εναπομείναντος αρχείου στη διαθέσιμη ρυθμαπόδοση. Το αρχείο παίρνει αμέσως το ποσοστό που δικαιούται από το δίκτυο όταν ξεκινά να μεταδίδεται όπως παρατηρήσαμε και από το πρώτο σενάριο. Όταν τα άλλα αρχεία τελειώνουν δεν έχουμε πτώση της συνολικής χρησιμοποίησης αλλά εκμετάλλευση της άμεσα από το αρχείο που ακόμα μεταδίδεται.

6.1.2 Παράδειγμα με Bandwidth:10 Mbit/s ,Delay:150ms, Errors:0%

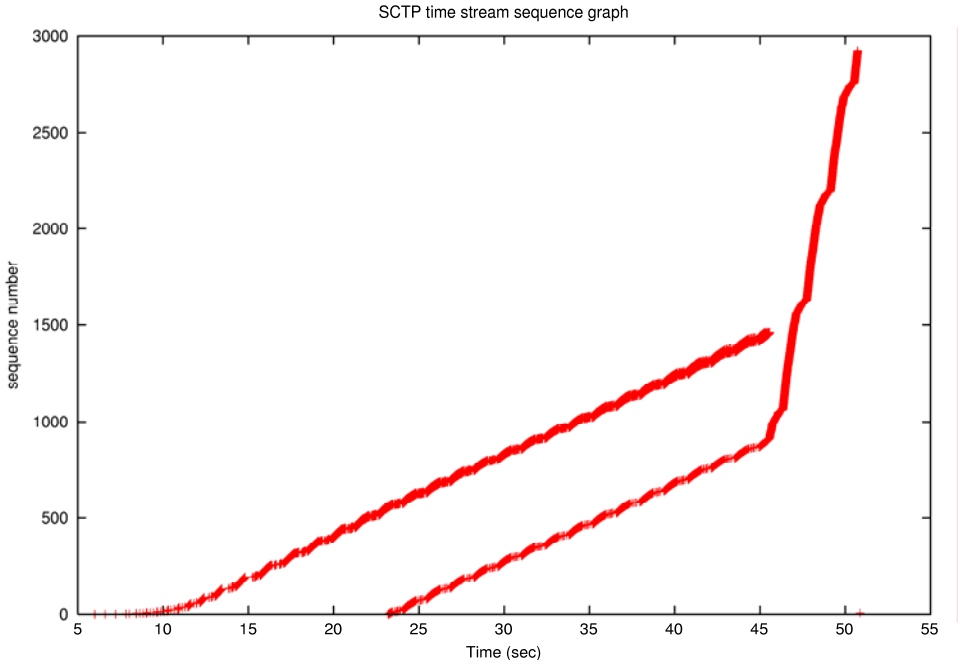
Στο επόμενο παράδειγμα έχουμε bandwidth 10Mbps delay 150ms και χωρίς λάθη.



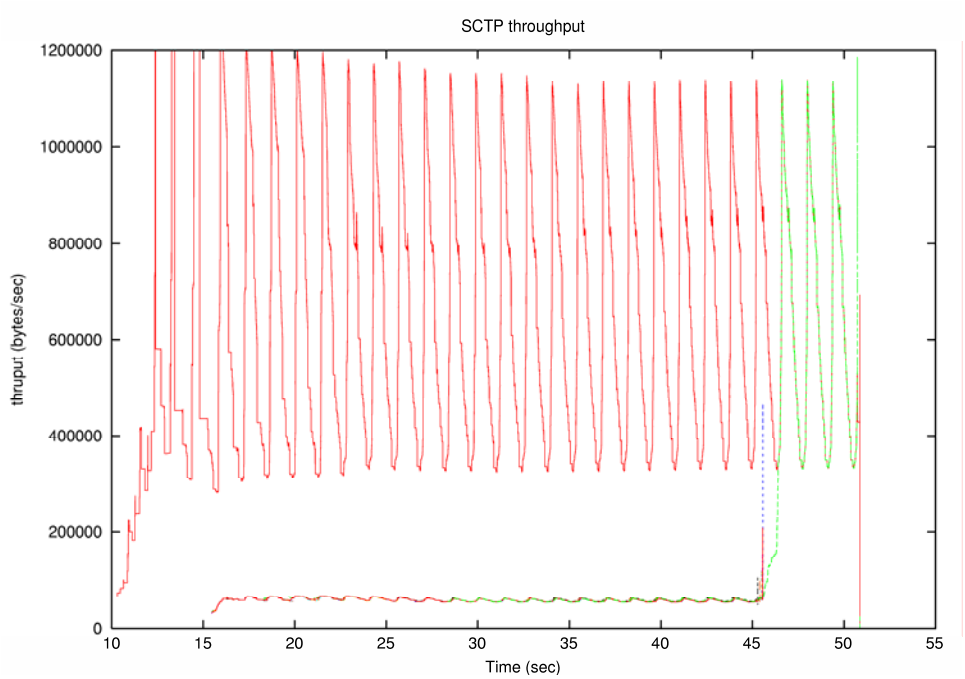
Σχήμα 66: Time sequence graph in TCP(BW:10Mbps,De:150ms,Er:0%)



Σχήμα 67: Κίνηση ανά σύνδεση(BW:10Mbps,De:50ms,Er:0%)



Σχήμα 68: Time sequence graph in SCTP(BW:10Mbps,De:50ms,Er:0%)



Σχήμα 69: Ρυθμαπόδοση σύνδεσης SCTP(BW:10Mbps,De:50ms,Er:0%)

Για να μπορέσουμε να δούμε τι στοιχίζει στη απόδοση της μεταφοράς η αυτόματη προσαρμογή που έχουμε με την χρήση πολλών ροών μπορούμε να δούμε τη μέση ρυθμαπόδοση κάθε αρχείου.

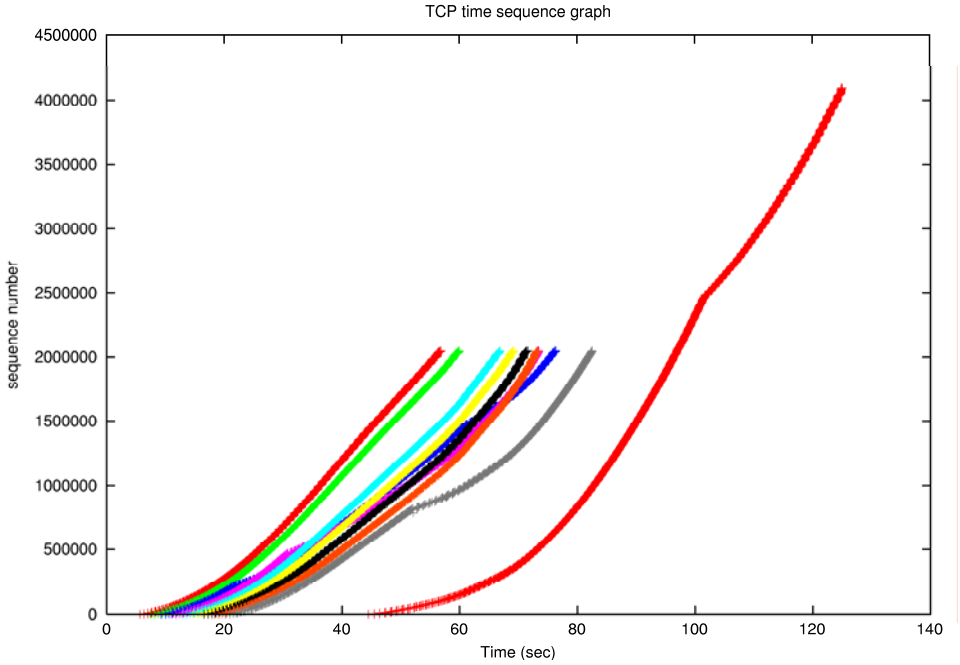
Πίνακας 7: Goodput ανά αρχείο (BW:10Mbps,De:150ms,Er:0%)

A/A	SCTP	TCP
1	37184	36817
2	41830	36787
3	41159	36776
4	41948	36779
5	42010	36776
6	42005	36875
7	42307	36862
8	41997	36847
9	42033	36817
10	106605	48065

Στη περίπτωση του SCTP η μέση ρυθμαπόδοση είναι καλύτερη για όλα τα αρχεία αλλά η μεγάλη διαφορά παρουσιάζεται για το τελευταίο αρχείο.

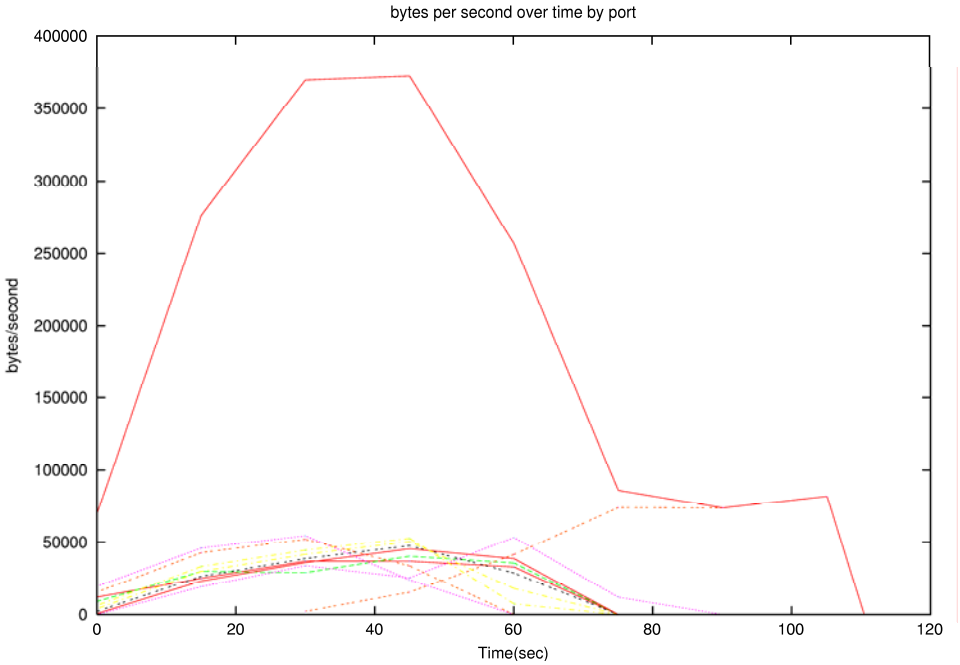
6.1.3 Παράδειγμα με Bandwidth:6 Mbit/s ,Delay:150ms, Errors:10^{-4}

Θα τελειώσουμε την αναλυτική παρουσίαση αποτελεσμάτων με ένα παράδειγμα το οποίο περιέχει και απώλειες. Συγκεκριμένα έχουμε bandwidth 6Mbps delay 150ms errors 10^{-4}%, για να μπορέσουμε να δούμε πως τα λάθη επηρεάζουν την μετάδοση και εντονότερα το SCTP άσχετα από το σενάριο.



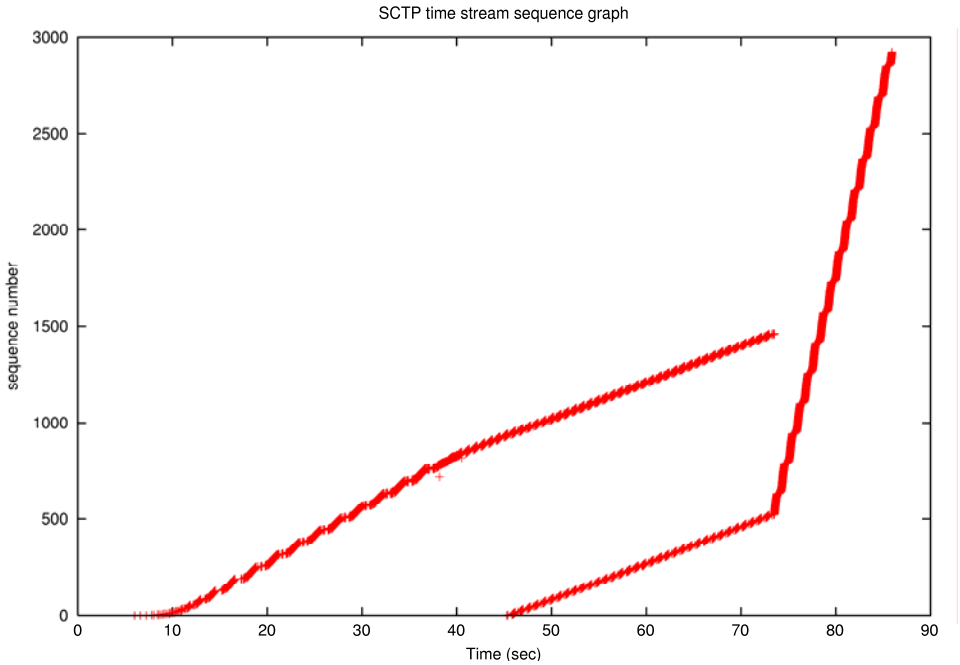
Σχήμα 70: Time sequence graph in TCP(BW:6Mbps,De:150ms,Er:0.0001%)

Θυμίζουμε ότι τα λάθη επηρεάζουν μόνο την σύνδεση στην οποία συμβαίνουν και αυτό φαίνεται από την αλλαγή τις κλίσης σε μεμονωμένα σημεία σε κάποιες από τις γραμμές.

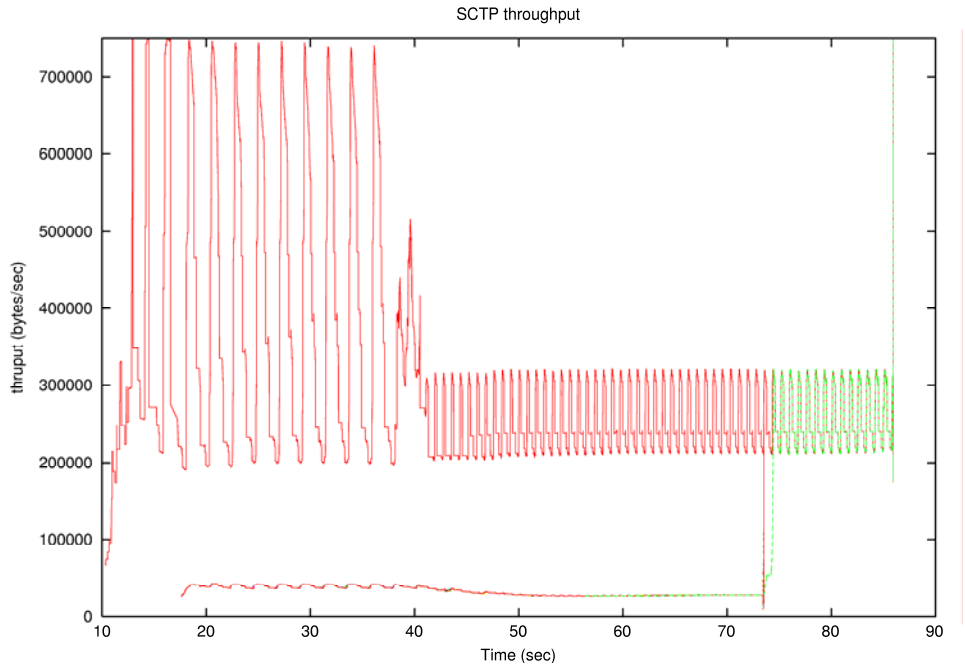


Σχήμα 71: Κίνηση ανά σύνδεση(BW:6Mbps,De:150ms,Er:0.0001%)

Αντίστοιχα στο SCTP έχουμε:



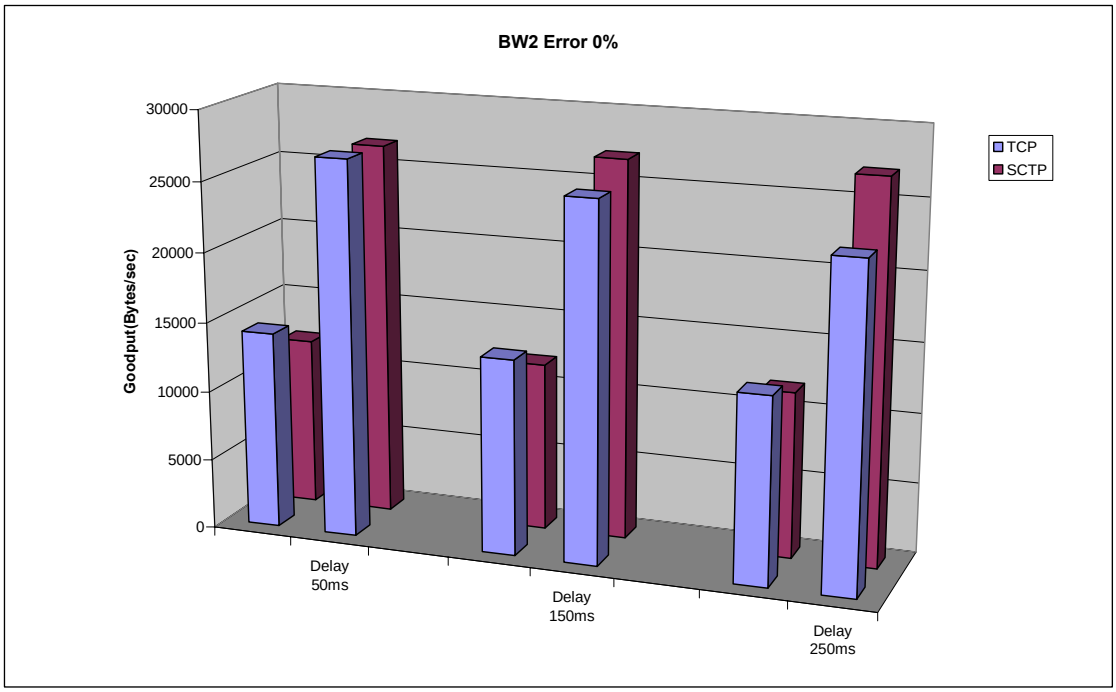
Σχήμα 72: Time sequence graph in SCTP(BW:6Mbps,De:150ms,Er:0.0001%)



Σχήμα 73: Ρυθμαπόδοση σύνδεσης SCTP(BW:6Mbps,De:150ms,Er:0.0001%)

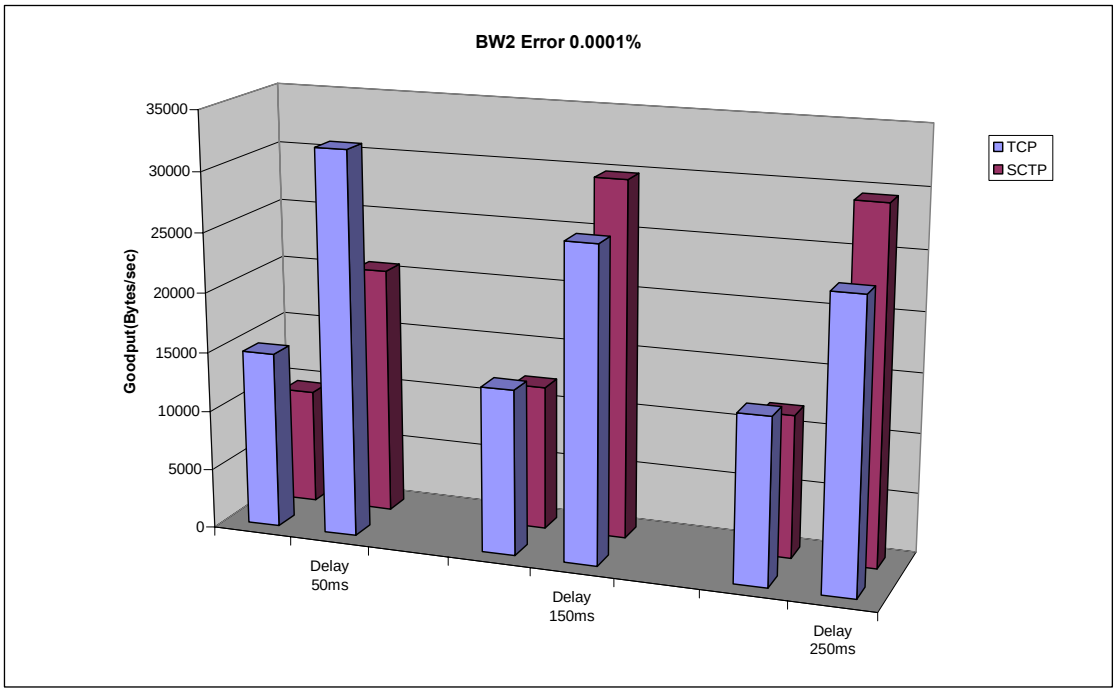
6.2 Παρουσίαση συνολικών αποτελεσμάτων

Παρουσιάζουμε τις συνολικές γραφικές με όλα τα αποτελέσματα από τις μετρήσεις στο σενάριο αυτό.

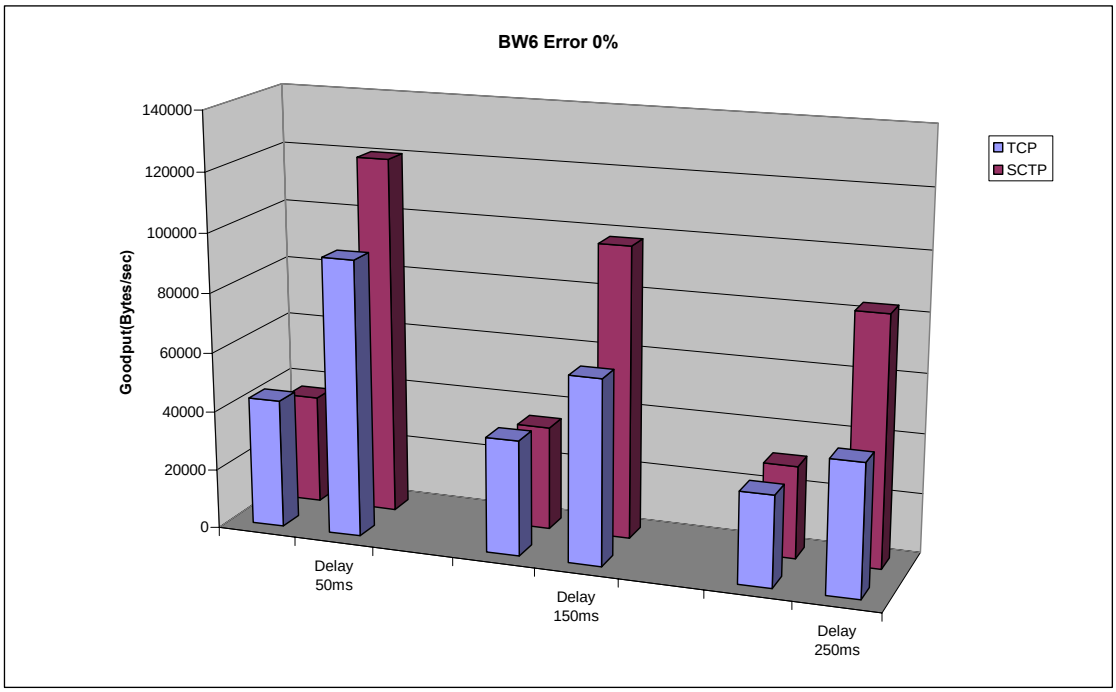


Σχήμα 74: Συνοπτικά αποτελέσματα(BW:2Mbps,Er:0%,De:50-150-250ms)

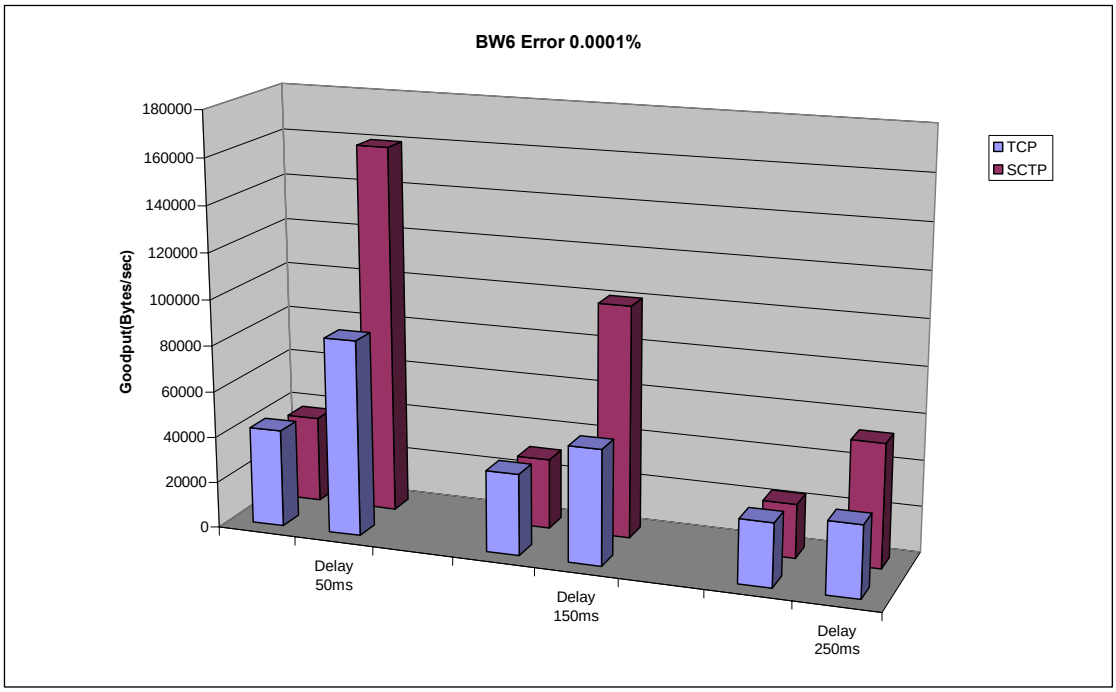
Σε αντίθεση με το προηγούμενο σενάριο η μέση ρυθμαπόδοση του τελευταίου αρχείου είναι πιο μεγάλη από τον μέσο όρο των πολλών και αυτό γιατί στη τελική φάση έχει όλο το δίκτυο στη διάθεση του χωρίς να γίνεται κάποια άλλη μεταφορά ταυτόχρονα. Με μικρές καθυστερήσεις δεν υπάρχει σημαντική διαφορά, αλλά όσο αυξάνει η καθυστέρηση αρχίζει να φαίνεται το πλεονέκτημα της χρήσης πολλαπλών ροών.



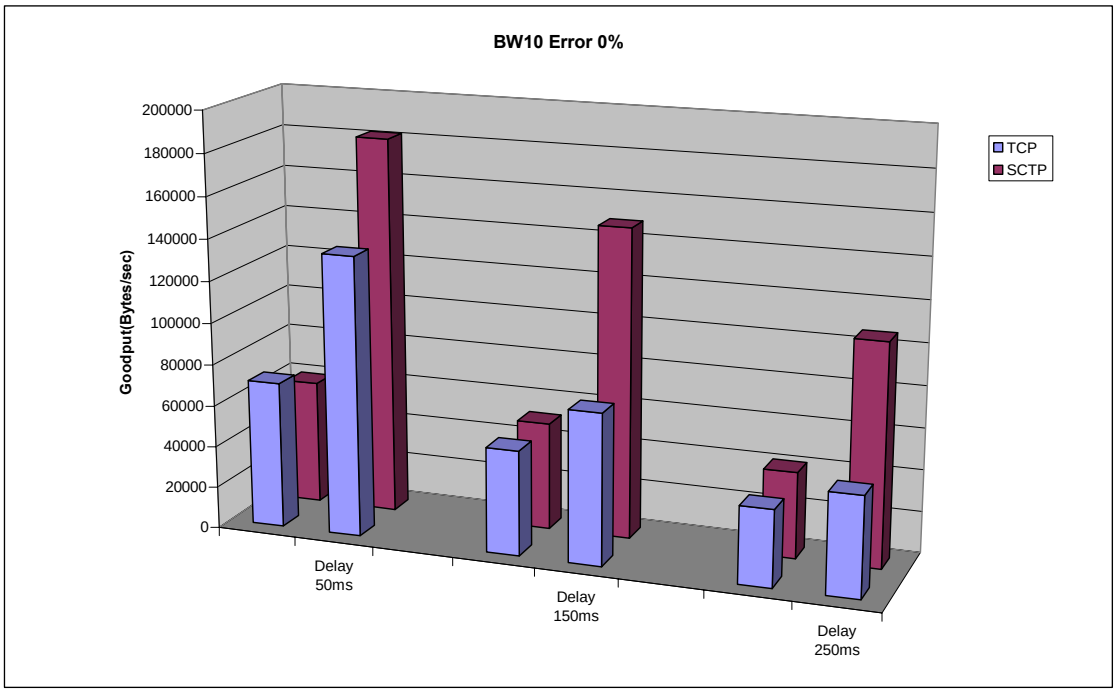
Σχήμα 75: Συνοπτικά αποτελέσματα(BW:2Mbps,Er:0.0001%,De:50-150-250ms)



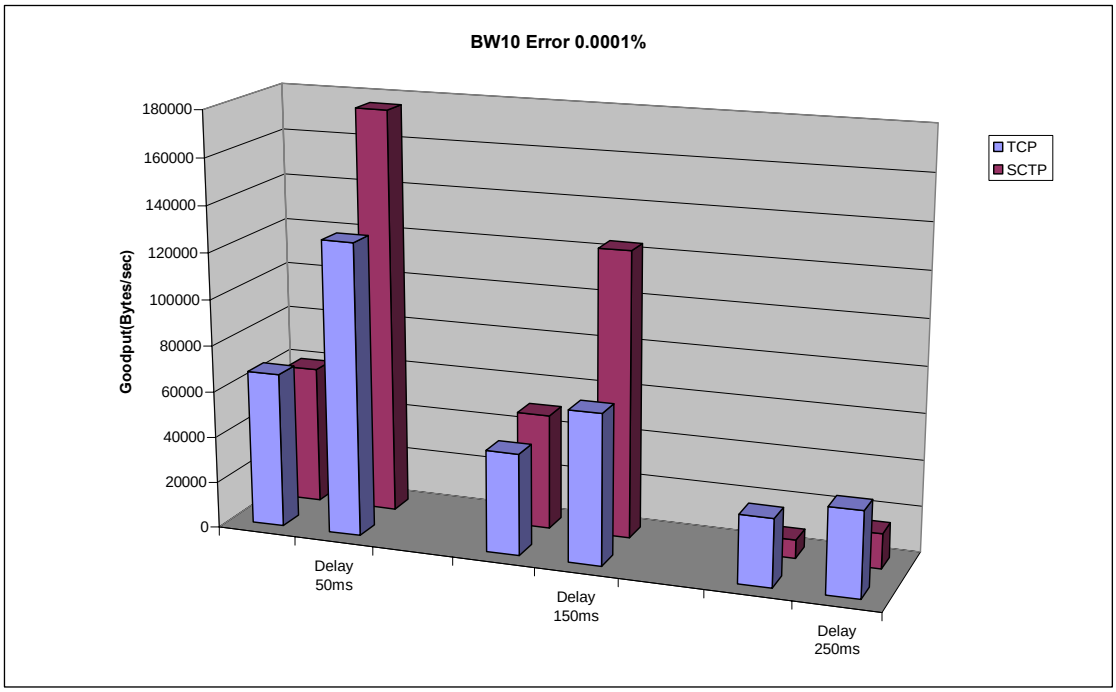
Σχήμα 76: Συνοπτικά αποτελέσματα(BW:6Mbps,Er:0%,De:50-150-250ms)



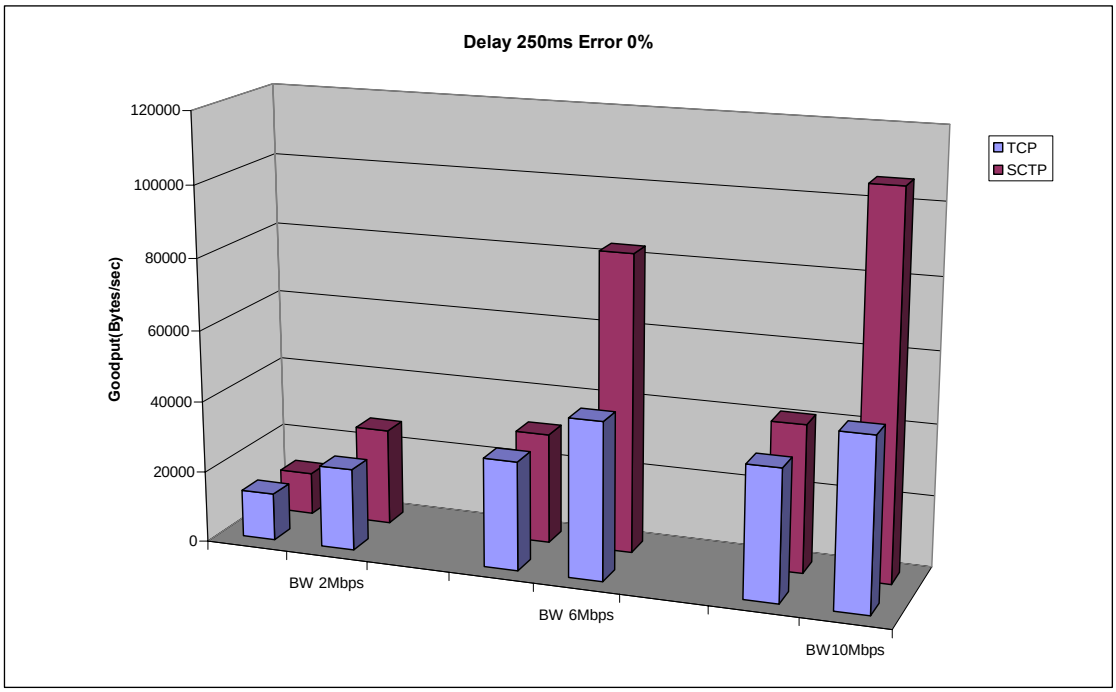
Σχήμα 77: Συνοπτικά αποτελέσματα(BW:6Mbps,Er:0.0001%,De:50-150-250ms)



Σχήμα 78: Συνοπτικά αποτελέσματα(BW:10Mbps,Er:0%,De:50-150-250ms)



Σχήμα 79: Συνοπτικά αποτελέσματα(BW:10Mbps,Er:0.0001%,De:50-150-250ms)



Σχήμα 80: Συνοπτικά αποτελέσματα(BW:2-6-10Mbps,Er:0%,De:250ms)

Συμπεράσματα

Σε αυτή τη τελευταία ενότητα θα θέλαμε να αποτυπώσουμε τα συμπεράσματα και τις γενικές παρατηρήσεις που κάναμε κατά τη διάρκεια της εκπόνησης αυτής της διπλωματικής εργασίας.

Τα πρωτόκολλα που χρησιμοποιούμε σήμερα στο διαδίκτυο έχουν γραφτεί εδώ και πολλά χρόνια, όμως παρέχουν καλή λειτουργικότητα και σταθερότητα και παραμένουν μια σταθερή αξία στα δίκτυα υπολογιστών. Οι ανάγκες όμως αυξάνονται και οι απαιτήσεις είναι όλο ένα και πιο πολλές. Φυσικά πρωτόκολλα όπως το TCP εξελίχθηκαν και νέες τεχνολογίες έρχονται να τα προσαρμόσουν στη σημερινή πραγματικότητα. Το SCTP είναι ένα πρωτόκολλο που έχει όλες τις δυνατότητες να αντικαταστήσει το TCP σε πολλές εφαρμογές μιας και εκμεταλλεύεται την πείρα που αποκτήθηκε από την χρήση του TCP και είναι εμπλουτισμένο με νέες δυνατότητες που το κάνουν πιο αποτελεσματικό σε πολλές περιπτώσεις.

Αν και αρχικά σχεδιάστηκε για να παρέχει υπηρεσίες σημαντικές σε μεταφορές τηλεφωνικής σηματοδοσίας μπορεί να χρησιμοποιηθεί σε πολλές εφαρμογές με αυξημένες απαιτήσεις. Αυτό κάναμε πράξη σε αυτή την εργασία και υλοποιήσαμε πρόγραμμα ανταλλαγής αρχείων που να χρησιμοποιεί το πρωτόκολλο αυτό. Οι αποδώσεις του δείχνουν να είναι παρόμοιες με το TCP όταν λειτουργούν με τα ίδια χαρακτηριστικά.

Η προσπάθεια μας να πάμε ένα βήμα πάρα πέρα και να χρησιμοποιήσουμε τις πολλαπλές ροές που προσφέρει το SCTP για να αυξήσουμε την απόδοση των μεταφορών μας πάνω στο δίκτυο ήταν επιτυχημένη σε πολλές περιπτώσεις όπως αναλύσαμε με ακρίβεια στα κεφάλαια που προηγήθηκαν.

Θα αναφέρουμε μερικά γενικά συμπεράσματα από την όλη προσπάθεια.

- Το πρωτόκολλο SCTP μπορεί να χρησιμοποιηθεί για υλοποίηση εφαρμογών που απαιτούν με σειρά και χωρίς λάθη μετάδοση σαν αντικατάστατο του πρωτοκόλλου TCP.

- Ο προγραμματισμός στο πρωτόκολλο αυτό έχει πολλά χαρακτηριστικά από τον προγραμματισμό σε TCP και η μετάβαση από το ένα στο άλλο έχει σχεδιαστεί να είναι αρκετά ομαλή.
- Σε δίκτυα με μεγάλο γινόμενο bandwidth*delay είναι κερδοφόρο και αποτελεσματικό να χρησιμοποιούμε πολλές ροές και όχι πολλές συνδέσεις για μεταφορά πολλών αρχείων, ένα προνόμιο που δεν υπάρχει στο TCP.
- Αν υπάρχουν πολλά λάθη στο δίκτυο η πιο πάνω υπόθεση δεν ισχύει και είναι προτιμότερη η χρήση ξεχωριστών ελέγχων συμφόρησης ανά μεταφορά.
- Όταν στο δίκτυο υπάρχει κάποια κίνηση τότε το όφελος στο SCTP προκύπτει γιατί μια νέα μεταφορά παίρνει αμέσως το μερίδιο που της αντιστοιχεί χωρίς να ξεκινά από μηδενική βάση και να προσπαθεί με τους αλγόριθμους αργής αρχής και ελέγχου συμφόρησης να καταλάβει αυτό που της αντιστοιχεί.
- Όταν στο δίκτυο αποδεσμεύονται κάποιοι πόροι, το όφελος στο SCTP προκύπτει γιατί χρησιμοποιείται αμέσως το διαθέσιμο δίκτυο χρησιμοποιώντας την γνώση που υπάρχει για το παράθυρο συμφόρησης, ενώ στο TCP η μετάδοση αυξάνει το ρυθμό μετάδοσης της σταδιακά και συνήθως με αργό ρυθμό σύμφωνα με τον αλγόριθμο αποφυγής συμφόρησης.

ΠΑΡΑΡΤΗΜΑ Α – ΚΩΔΙΚΑΣ FTP SERVER ΣΕ SCTP

Ο πλήρης κώδικας των αρχείων του FTP server που δημιουργήθηκαν και χρησιμοποιήθηκαν κατά τη διάρκεια αυτής της διπλωματικής εργασίας παρατίθεται με εκτενή σχόλια ανά αρχείο, στο παράρτημα αυτό.

Αρχείο servmain.c

```
#include    "unp.h"
#include    "global.h"


void precess_function(int connfd);              /* process to handle incomming commands*/



// we must call err_msg instead of printf because we are running a deamon!

int main(int argc, char **argv)
{
        int                                     listenfd,connfd,clilen;
        pid_t                                   childpid;
        void                                    sig_chld(int);
        struct sockaddr_in cliaddr;

        if (argc < 2 )
                err_quit("usage: ftpserver <port>");

        //daemon_init(argv[0], 0);  //to make the program run as deamon


        listenfd = mysctp_listen(atoi(argv[1]));
        printf("listening port %d\n",atoi(argv[1]));


        Signal(SIGCHLD, sig_chld);


        err_msg("FTP SERVER STARTED");

        clilen=sizeof(cliaddr);

        struct sctp_sndrcvinfo sri;
        bzero(&sri,sizeof(sri));

        for ( ; ; ) {
                err_msg("ready to accept connection");
                connfd = ext_accept(listenfd, (SA *) &cliaddr,&clilen);
                        sctp_get_no_strms(connfd,&sri);

                /*to have a concurrent server*/
                //if ( (childpid = Fork()) == 0) {   /* child process */
                //              ext_close(listenfd);                    /* close listening socket */
                                precess_function(connfd);               /* process request to handle incoming commands*/
                //              exit(0);
                //}
                err_msg("closing connected socket");
                ext_close(connfd);                              /* parent closes connected socket */
        }
}
```

Αρχείο process_func.c

```
#include <dirent.h>
#include "unp.h"
```

```
#include "global.h"
#include <pthread.h>

#define MAX_FILE_NO 10
#define MAXLINE1400 1400
int delay=0;

//diathesimen entoles
struct kwtab yykey[] = {                        /* In order defined in RFC 765 */
                "USER",  user_cmd ,
                "PASS",  pass_cmd ,
                "PORT",  port_cmd ,
                "RETR",  retr_cmd ,
                "CWD",   cwd_cmd ,
                "LIST",  list_cmd,
                "PWD",   pwd_cmd ,
                "DELAY", delay_cmd,
                "0"
};


char sentline[MAXLINE]="\n";

                char a[16];  //for the data connection
                char p[6];
int gotolisten=0;
int notifcame=0;

int datasockfd;

pthread_cond_t start_cond=PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
int threadsok=0;

struct fileinfo {
                char * f_name;
                int f_stream;
                float startdelay;
                }fileinfo[MAX_FILE_NO];

void * threadget(void * arg);
void Pthread_cond_wait(pthread_cond_t *cptr, pthread_mutex_t *mptr);


static void handle_event(void *buf)
{
  struct sctp_assoc_change *sac;
  struct sctp_send_failed  *ssf;
  struct sctp_paddr_change *spc;
  struct sctp_remote_error *sre;
  union sctp_notification  *snp;
  char addrbuf[INET6_ADDRSTRLEN];
  const char *ap;
  struct sockaddr_in  *sin;
  struct sockaddr_in6 *sin6;

  snp = buf;

  switch(snp->sn_header.sn_type) {
    case SCTP_ASSOC_CHANGE:
      sac = &snp->sn_assoc_change;
      printf("^^^ assoc_change: state=%hu, error=%hu, instr=%hu, outstr=%hu\n",
             sac->sac_state, sac->sac_error, sac->sac_inbound_streams, sac->sac_outbound_streams);
     break;
    case SCTP_SEND_FAILED:
      ssf = &snp->sn_send_failed;
      printf("^^^ sendfailed: len=%hu err=%d\n", ssf->ssf_length, ssf->ssf_error);
     break;
    case SCTP_PEER_ADDR_CHANGE:
      spc = &snp->sn_paddr_change;
      if(((struct sockaddr_in *)&spc->spc_aaddr)->sin_family == AF_INET) {
        sin = (struct sockaddr_in *)&spc->spc_aaddr;
        ap = inet_ntop(AF_INET, &sin->sin_addr, addrbuf, INET6_ADDRSTRLEN);
      } else {
        sin6 = (struct sockaddr_in6 *)&spc->spc_aaddr;
```

```
          ap = inet_ntop(AF_INET6, &sin6->sin6_addr, addrbuf, INET6_ADDRSTRLEN);
        }
        printf("^^^ intf_change: %s state=%d, error=%d\n", ap, spc->spc_state, spc->spc_error);
       break;
      case SCTP_REMOTE_ERROR:
        sre = &snp->sn_remote_error;
        printf("^^^ remote_error: err=%hu\n", ntohs(sre->sre_error));
       break;
      case SCTP_SHUTDOWN_EVENT:
        printf("^^^ shutdown event\n");
       break;
      default:
        printf("unknown type: %hu\n", snp->sn_header.sn_type);
       break;
   }
}



void precess_function(int connfd)               /* process to handle incoming commands*/
{
            struct sctp_sndrcvinfo sri;
            bzero(&sri,sizeof(sri));

            snprintf(sentline,sizeof(sentline),"220 Connected to Florides FTP server.\n");

            ext_write(connfd, sentline, strlen(sentline));

            strcpy(sentline,"\n");

for( ; ; )                                  //main loop
    {


            initialize();
            getline(connfd);
            if (notifcame==1) {notifcame=0;continue;}
            if (gotolisten==1) {gotolisten=0;return;}
            process(lexical(),connfd);
    }

}

void initialize()
{

/* memory allocation for shell data */
if ( ( shglp = (struct shglobals *)malloc(sizeof(struct shglobals)) )
                == NULL )
   {
   err_msg("Cannot allocate memory for shell global variables.\n");
   exit(1);
   }
}

void getline(int connfd)
{
            union sctp_notification *snp;
            struct sctp_sndrcvinfo sri;
            int msg_flags;
            int rd_sz;
            struct sctp_event_subscribe events;

            /* Enable ancillary data and notifications */
            //memset(&events,1,sizeof(events));
            events.sctp_data_io_event = 1;
            events.sctp_association_event=1;
    if(ext_setsockopt(connfd, IPPROTO_SCTP, SCTP_EVENTS, &events, sizeof(events)) < 0) {
       perror("setsockopt SCTP_EVENTS");
       exit(1);
    }


            bzero(&sri,sizeof(sri));
```

```
                printf("waiting at sctp_recvmsg\n");
                msg_flags=0;
                rd_sz=Sctp_recvmsg(connfd, Linebuf, sizeof(Linebuf),
                                   NULL, 0,
                                   &sri,&msg_flags);

                /* Intercept notifications here */
        if(msg_flags & MSG_NOTIFICATION) {
          snp = (union sctp_notification *)Linebuf;
                 printf("event notification\n");
          handle_event(Linebuf);
                 notifcame=1;
                 return;
        }

                if(rd_sz==0) {gotolisten=1;return;}


                linep = Linebuf;
                printf("leaving getline\n");

}



/* process -- process the input line */

void process(struct lex *head,int connfd)
{
struct lex *lptr;
void (*func)();

lptr = head;
if ( lptr == (struct lex *)NULL)
                return;
/* every line mast start with a command */
if ( lptr->class != COMMAND)
                {
                err_msg("%s : No such command - ",lptr->text) ;
                release(head);
                }
else
                {
                func = lptr->fnaddr;
                lptr = lptr->next;

                                /* execute the proper function for the specified command*/
                                (*func)(head,connfd);
                }
}


/* release -- free the list memory */

void release(struct lex *head)
{
struct lex *curr, *prev;
                curr = head;
                while( curr != (struct lex *)NULL )
                                {
                                prev = curr;
                                curr = curr->next;
                                free(prev);
                                }
}


/* lexical – lexical analysis  */

struct lex *lexical()
{
                struct lex *head, *currtok, *prevtok;

                head = currtok = prevtok = (struct lex *)NULL;
```

```
        while( yylex() != EOL )
                {
                currtok = (struct lex *)Malloc(sizeof(struct lex));
                if ( head == (struct lex *)NULL )
                        head = currtok;
                if ( prevtok != (struct lex *)NULL )
                        prevtok->next = currtok;
                *currtok = yyval;
                prevtok = currtok;
                }
        if( prevtok != (struct lex *)NULL )
                prevtok->next = (struct lex *)NULL;
        return( head );
}


/* yylex -- identify next token */

int yylex()
{
    int c,d,i;
    int dev;
    char *tp;

    /* ignore spaces ... */
    while ( (c = *linep++) == ' ');
    /* check end of line */
    if (c =='\n') return(EOL);
    yyval.class = IDENTF;
    for (tp=token,i=1 ; ; i++) {
/* find next token */
        if (i<MAXTOKEN) *tp++=c;
/* "cat big tokens */
        c=*linep;
        if (c=='\n'||c==' ') break;
        linep++;
    }
    *tp=(char)NULL;
    strcpy(yyval.text,token);
                {
                /* check if the first token is a regognized command*/
                for( i = 0 ; yykey[i].name != "0" ; i++)
                                if ( strcmp(yykey[i].name,token) == 0)
                                {
                                yyval.fnaddr = yykey[i].fnaddr;
                                yyval.class = COMMAND;
                                return(COMMAND);
                                }
                /* else is a simpe argument */
                return(IDENTF);
                }
}



//implementation of command functions

void user_cmd(struct lex *head,int connfd)
{
        err_msg("user_cmd\n");(void) fflush(stdout);

        struct lex *argum1;
        argum1=head->next;

        if (strcmp(argum1->text, "ftp") == 0 || strcmp(argum1->text, "anonymous") == 0) {

        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"331 Guest login ok, type your name as password.\n");
                ext_write(connfd,sentline,strlen(sentline));
                strcpy(sentline,"");
                                                                                        }
                else        {
        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"530 User %s unknown.\n", argum1->text);
                ext_write(connfd,sentline,strlen(sentline));
```

```
                        strcpy(sentline,"");
                                }


        release(head);
        return;
}


void pass_cmd(struct lex *head,int connfd)
{
        err_msg("pass_cmd\n");(void) fflush(stdout);
        //allow all
        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"230 Guest login ok, access restrictions apply.\n");
                ext_write(connfd,sentline,strlen(sentline));
                strcpy(sentline,"");

        release(head);
        return;
}

void cwd_cmd(struct lex *head,int connfd)
{
        err_msg("cwd_cmd\n");(void) fflush(stdout);
        struct lex *argum1;
        argum1=head->next;

if (chdir(argum1->text) < 0){
        bzero(&sentline,MAXLINE);
        snprintf(sentline,sizeof(sentline),"550 can't cwd to %s.\n",argum1->text);
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");
                                        }

else {
        bzero(&sentline,MAXLINE);
        snprintf(sentline,sizeof(sentline),"250 Requested file action okay, completed\n");
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");
        }


        release(head);
        return;
}

void port_cmd(struct lex *head,int connfd)
{
        err_msg("port_cmd\n");(void) fflush(stdout);
        int a1,a2,a3,a4,p1,p2;
        struct lex *argum1;
        argum1=head->next;
        int pint;
                /*
                 * What we've got at this point is a string of comma separated
                 * one-byte unsigned integer values, separated by commas.
                 * The first four are the an IP address. The fifth is the MSB
                 * of the port number, the sixth is the LSB. From that we'll
                 * prepare a sockaddr_in.
                 */

                if (sscanf(argum1->text,"%d,%d,%d,%d,%d,%d",
                                &a1,&a2,&a3,&a4,&p1,&p2)
                   != 6)

                {

        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"500 PORT command faild.\n");
                ext_write(connfd,sentline,strlen(sentline));
                strcpy(sentline,"");
                        return;
                }
```

```
                        pint=(p1*256)+p2;
                        snprintf(p,sizeof(p),"%d\0",pint);
err_msg("port=%s\n",p);(void) fflush(stdout);

                        snprintf(a,sizeof(a),"%d.%d.%d.%d\0",a1,a2,a3,a4);
err_msg("address=%s\n",a);(void) fflush(stdout);


        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"200 PORT command successful.\n");
                ext_write(connfd,sentline,strlen(sentline));
                strcpy(sentline,"");


        release(head);
        return;
}

void retr_cmd(struct lex *head,int connfd)
{
        err_msg("retr_cmd\n");(void) fflush(stdout);
        struct lex * argum[MAX_FILE_NO];
        struct lex * tocount;
        pthread_t tid[MAX_FILE_NO];
        int realnooffiles,i;       // for the number of files on the RETR command
        threadsok=0;

        start_cond=PTHREAD_COND_INITIALIZER;
        mutex=PTHREAD_MUTEX_INITIALIZER;
        /* count no of files in RETR command*/
        realnooffiles=0;
        tocount=head;
        for (i=0;i<MAX_FILE_NO;i++)
        {
        if ((tocount->next)!=NULL)
                {
                tocount=tocount->next;
                realnooffiles++;
                }
        }

        printf("No of files requested %d\n",realnooffiles);

        /* fill in file info*/
        argum[0]=head->next;
        for (i=1;i<realnooffiles;i++)
        argum[i]=argum[i-1]->next;

        for (i=0;i<realnooffiles;i++)
        {
        fileinfo[i].f_name=argum[i]->text;
        fileinfo[i].f_stream=i;
        fileinfo[i].startdelay=0;
        }
        fileinfo[realnooffiles-1].startdelay=delay;

        /* connect to the client */
        datasockfd=mysctp_connect(a,p);

        /* Set the send buffer size, then fetch it and print its value.  */
        int sendbuff = 400;
        int info=ext_setsockopt(datasockfd, SOL_SOCKET, SO_SNDBUF, (void *)&sendbuff, sizeof(sendbuff));
        printf("info= %d\n", info);
        int optlen = sizeof(sendbuff);
        sendbuff=0;
        ext_getsockopt(datasockfd, SOL_SOCKET, SO_SNDBUF, &sendbuff, &optlen);
        printf("send buffer size = %d\n", sendbuff);

        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"150 Opening data connection.\n ");
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");

        /* sent each file with one thread */
        for (i=0;i<realnooffiles;i++)
```

```
        pthread_create(&tid[i], NULL, threadget, &fileinfo[i]);

        while(threadsok!=realnooffiles) ;
        pthread_cond_broadcast(&start_cond);

        for (i=0;i<realnooffiles;i++)
        pthread_join(tid[i],NULL);

        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"226 Closing data connection.\n ");
        ext_write(connfd,sentline,strlen(sentline));
        bzero(&sentline,sizeof(sentline));

        ext_close(datasockfd);
        release(head);
        return;
}

void pwd_cmd(struct lex *head,int connfd)
{
        err_msg("pwd_cmd\n");(void) fflush(stdout);
        char path[PATH_SIZE + 1];

        if (getwd(path) == (char *)NULL)        {
        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"550 %s.\n", path);
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");
                                                        }

        else                                            {
        bzero(&sentline,MAXLINE);
        snprintf(sentline,sizeof(sentline),"257 \"%s\" is current directory.\n", path);
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");
                                                        }



        release(head);
        return;
}

void list_cmd(struct lex *head,int connfd)
{
        err_msg("list_cmd\n");(void) fflush(stdout);
        FILE *fd;
        char line[MAXLINE];
        struct lex *argum1;
        argum1=head->next;




        datasockfd=mysctp_connect(a,p);

        bzero(&sentline,sizeof(sentline));
        snprintf(sentline,sizeof(sentline),"150 Opening data connection.\n ");
        ext_write(connfd,sentline,strlen(sentline));
        strcpy(sentline,"");


        DIR *dp;
        struct dirent *ep;

        dp = opendir (argum1->text);
        if (dp != NULL)
          {
            while (ep = readdir (dp))
                        {ext_write(datasockfd,ep->d_name,strlen(ep->d_name));
                        ext_write(datasockfd,"\n",strlen("\n"));}
            (void) closedir (dp);
          }
        else
                        ext_write(datasockfd,"Couldn't open the directory.",strlen("Couldn't open the directory."));
```

```
	bzero(&sentline,sizeof(sentline));
	snprintf(sentline,sizeof(sentline),"226 Closing data connection.\n ");
	ext_write(connfd,sentline,strlen(sentline));
	strcpy(sentline,"");

	ext_close(datasockfd);
	release(head);
	return;
}

void delay_cmd(struct lex *head,int connfd)
{
	struct lex *argum1;
	argum1=head->next;

	if (argum1==NULL) delay=0;
	else delay=atoi(argum1->text);
	printf("DELAY set from user :%d",delay);

	snprintf(sentline,sizeof(sentline),"200 Server set delay for %d sec.\n",delay);
	ext_write(connfd,sentline,strlen(sentline));
	strcpy(sentline,"");

	release(head);
	return;
}


void * threadget(void * arg)
{
	struct fileinfo *fptr;
	char line[MAXLINE1400];
	FILE *fd;
	int n;


	fptr=(struct fileinfo *)arg;
	printf("opennig %s for stream %d\n",fptr->f_name,fptr->f_stream);(void) fflush(stdout);

	fd = Fopen(fptr->f_name, "rb");

	pthread_mutex_lock(&mutex);
	threadsok++;
	Pthread_cond_wait(&start_cond,&mutex);
	pthread_mutex_unlock(&mutex);

	sleep(fptr->startdelay);

	while ( (n=fread(line,sizeof(char), MAXLINE1400,fd))>0)
	{
		Sctp_sendmsg(datasockfd, line, n,
				NULL, 0,
				0,0,
				fptr->f_stream,
				0, 0);
		sleep(0);
	}

	Fclose(fd);
	printf("Thread ended\n");
	return(NULL);
}

void
Pthread_cond_wait(pthread_cond_t *cptr, pthread_mutex_t *mptr)
{
	int			n;

	if ( (n = pthread_cond_wait(cptr, mptr)) == 0)
		return;
	errno = n;
	err_sys("pthread_cond_wait error");
}
```

Αρχείο mysctp_connect.c

```
#include    "unp.h"

int
mysctp_connect(const char *host, const char *serv)
{
            struct sockaddr_in servaddr;
            struct sctp_event_subscribe evnts;
            struct sctp_initmsg initm;
            int sock_fd,port;

            port=atoi(serv);



            if((sock_fd = ext_socket(PF_INET, SOCK_STREAM, IPPROTO_SCTP))<0)
            err_sys("socket error");

            bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family = AF_INET;
            servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
            servaddr.sin_port = htons(port);
            inet_pton(AF_INET,host, &servaddr.sin_addr);

            bzero(&evnts, sizeof(evnts));
            evnts.sctp_data_io_event = 1;
            ext_setsockopt(sock_fd,IPPROTO_SCTP, SCTP_EVENTS,
                        &evnts, sizeof(evnts));


            bzero(&initm,sizeof(initm));
            initm.sinit_num_ostreams = 10;
            initm.sinit_max_instreams = 10;
            ext_setsockopt(sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                        &initm, sizeof(initm));

            ext_connect(sock_fd,(SA *)&servaddr,sizeof(servaddr));

            return(sock_fd);
}


int
Mysctp_connect(const char *host, const char *serv)
{
            return(mysctp_connect(host, serv));
}
```

Αρχείο mysctp_listen.c

```
#include    "unp.h"


int mysctp_listen(int port)
{
            int sock_fd;
            struct sockaddr_in servaddr;
            struct sctp_event_subscribe evnts;


      sock_fd = ext_socket(PF_INET, SOCK_STREAM, IPPROTO_SCTP);



            bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family = AF_INET;
            servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
            servaddr.sin_port = htons(port);
```

```
        ext_bind(sock_fd, (SA *) &servaddr, sizeof(servaddr));

        bzero(&evnts, sizeof(evnts));

        /* Enable ancillary data and notifications */
        evnts.sctp_data_io_event = 1;
        evnts.sctp_association_event=1;
        if (ext_setsockopt(sock_fd, IPPROTO_SCTP, SCTP_EVENTS, &evnts, sizeof(evnts)) < 0) {
        perror("setsockopt SCTP_EVENTS");
        exit(1);
        }
        /* To enable selected ancillary events */
        /*evnts.sctp_data_io_event = 1;
        evnts.sctp_association_event=1;
        ext_setsockopt(sock_fd, IPPROTO_SCTP, SCTP_EVENTS,
                        &evnts, sizeof(evnts));
        */

        ext_listen(sock_fd, LISTENQ);

        return(sock_fd);
}
```

Αρχείο sctpwrapper.c

```
/*
 *
 * Purpose: wrapper functions
 *
 */



#include "unp.h"




int
Sctp_recvmsg(int s, void *msg, size_t len,
                struct sockaddr *from, socklen_t *fromlen,
                struct sctp_sndrcvinfo *sinfo,
                int *msg_flags)
{
        int ret;
        ret = sctp_recvmsg(s,msg,len,from,fromlen,sinfo,msg_flags);
        if(ret < 0){
                printf("sctp_recvmsg error");
                exit(1);
        }
        return(ret);
}


int
Sctp_sendmsg (int s, void *data, size_t len, struct sockaddr *to,
                socklen_t tolen, uint32_t ppid, uint32_t flags,
                uint16_t stream_no, uint32_t timetolive, uint32_t context)
{
        int ret;
        ret = sctp_sendmsg(s,data,len,to,tolen,ppid,flags,stream_no,
                                timetolive,context);
        if(ret < 0){
                printf("sctp_sendmsg error");
                exit(1);
        }
        return(ret);
}

int
sctp_get_no_strms(int sock_fd,struct sctp_sndrcvinfo* sri)
{
        int retsz;
```

```
        struct sctp_status status;
        retsz = sizeof(status);
        bzero(&status,sizeof(status));

        sctp_opt_info(sock_fd,(*sri).sinfo_assoc_id, SCTP_STATUS,
                &status, &retsz);
        printf("no_of_instrms %d\n",status.sstat_outstrms);
        return(status.sstat_outstrms);
}

/*sctp_assoc_t
sctp_address_to_associd(int sock_fd, struct sockaddr *sa, socklen_t salen)
{
        struct sctp_paddrparams sp;
        int siz;

        siz = sizeof(struct sctp_paddrparams);
        bzero(&sp,siz);
        memcpy(&sp.spp_address,sa,salen);
        sctp_opt_info(sock_fd,0,
                SCTP_PEER_ADDR_PARAMS, &sp, &siz);
        return(sp.spp_assoc_id);
}
*/
```

Αρχείο global.h

```
        extern char a[16];      //for data connection (command PORT)
        extern char p[6];


//Megista oria

#define PATH_SIZE  50                   //maximum  path length
#define MAXTOKENPATH_SIZE               //maximum length of each argument
#define MAXARGS         20                      //maximum arguments nimber
#define     MAXLINE                     5600 /*4096          max text line length */



struct lex {
    int class;                  // type of lectic unit
    char text[MAXTOKEN+1];   // text of lactic unit
    void (*fnaddr)();       // command function
    struct lex *next;   // pointer of next token
};

//types of lactic unit

#define COMMAND         -55
#define IDENTF          -66
#define EOL             -77




//command to function

struct kwtab {
    char *name;             /* name of command */
    void (*fnaddr)();       /* name of the function */

};


//shell variable structure

struct shglobals  {
        struct lex yyval;       // structure of current token
        char Linebuf[MAXLINE];  // current input line
        char token[MAXTOKEN];   // current token
        char *linep;                    // help pointer
```

```
};

struct shglobals *shglp;

//for easy variable definesion

#define yyval                   (shglp->yyval)
#define Linebuf                 (shglp->Linebuf)
#define token                   (shglp->token)
#define linep                   (shglp->linep)
#define currdev                 (shglp->currdev)

// functions definition
void initialize();
void getline(int connfd);
void process(struct lex *head,int connfd);
void release(struct lex *head);
struct lex *lexical(void);
int yylex(void);

//command function
void user_cmd(struct lex *head,int connfd);
void pass_cmd(struct lex *head,int connfd);
void cwd_cmd(struct lex *head,int connfd);
void port_cmd(struct lex *head,int connfd);
void retr_cmd(struct lex *head,int connfd);
void pwd_cmd(struct lex *head,int connfd);
void list_cmd(struct lex *head,int connfd);
void delay_cmd(struct lex *head,int connfd);
```

Οποιαδήποτε άλλα αρχεία έχουν χρησιμοποιηθεί , όπως αρχεία επικεφαλίδων και αρχεία που περιλαμβάνουν wrapper functions υπάρχουν στο βιβλίο Unix Network Programming, και δεν θεωρείται σκόπιμο να τα περιλάβουμε σε αυτό το παράρτημα.

ΠΑΡΑΡΤΗΜΑ Β – ΚΩΔΙΚΑΣ FTP CLIENT ΣΕ SCTP

Ο πλήρης κώδικας των αρχείων του FTP client που δημιουργήθηκαν και χρησιμοποιήθηκαν κατά τη διάρκεια αυτής της διπλωματικής εργασίας παρατίθεται με εκτενή σχόλια ανά αρχείο, στο παράρτημα αυτό.

Αρχείο shell.c

```
#include <stdio.h>
#include <string.h>
#include "global.h"
#include "sh.h"
#include "cmds.h"


int connected=0;  //global to check if connected
int sockfd; //socked descriptor
int ulogin=0;

/*available command*/

struct kwtab yykey[] = {
      "exit",           exit_cmd,
      "help",                     help_cmd,
      "open",                     open_cmd,
            "close",    close_cmd,
            "user",               user_cmd,
            "pwd",                pwd_cmd,
            "cwd",                cwd_cmd,
            "list",               list_cmd,
            "get",                get_cmd,
            "block",    block_cmd,
            "delay",    delay_cmd,
      "0",

};


int main(int argc, char ** argv)
{
initialize();

for( ; ; )                          //main loop
    {
            printf(PROMPT);
            (void) fflush(stdout);
            getline();
            process(lexical());
    }
}


/* initialize shell */

void initialize()
{

/* memory allocation for shell data */
if ( ( shglp = (struct shglobals *)malloc(sizeof(struct shglobals)) )
                == NULL )
   {
   printf("Cannot allocate memory for shell global variables.\n");
   exit(1);
   }


system("clear");        //clear the screen
```

```
printf("\nFTP Shell started --- Enter command (see help) ...\n\n");
}

/* getline -- read line from user */

void getline()
{
int c;
char lastline[MAXLINE];

linep = linebuf;
while ( (c = getchar()) != '\n')
        *linep++ = c;

*linep = (char)NULL;
linep = linebuf;
}


/* process -- process the input line */

void process(struct lex *head)
{
struct lex *lptr;
void (*func)();

lptr = head;
if ( lptr == (struct lex *)NULL)
        return;
/* every line mast start with a command */
if ( lptr->class != COMMAND)
        {
        printf("%s : No such command - ",lptr->text) ;
        printf("Type 'help' for a list of commands.\n");
        release(head);
        }
else
        {
        func = lptr->fnaddr;
        lptr = lptr->next;

                /* execute the proper function for the specified command*/
                (*func)(head);
        }
}


/* release -- free the list memory */

void release(struct lex *head)
{
struct lex *curr, *prev;
        curr = head;
        while( curr != (struct lex *)NULL )
                {
                prev = curr;
                curr = curr->next;
                free(prev);
                }
}


/* lexical analysis */

struct lex *lexical()
{
        struct lex *head, *currtok, *prevtok;

        head = currtok = prevtok = (struct lex *)NULL;
        while( yylex() != EOL )
                {
                currtok = (struct lex *)malloc(sizeof(struct lex));
                if ( head == (struct lex *)NULL )
                        head = currtok;
                if ( prevtok != (struct lex *)NULL )
```

```
                                        prevtok->next = currtok;
                        *currtok = yyval;
                        prevtok = currtok;
                        }
                if( prevtok != (struct lex *)NULL )
                        prevtok->next = (struct lex *)NULL;
                return( head );
}


/* yylex -- identify next token */

int yylex()
{
    int c,d,i;
    int dev;
    char *tp;

    /* ignore spaces ... */
    while ( (c = *linep++) == ' ');
    /* check end of line */
    if (c == (char)NULL) return(EOL);
    yyval.class = IDENTF;
    for (tp=token,i=1 ; ; i++) {
/* find next token */
                if (i<MAXTOKEN) *tp++=c;
/* “cat” big tokens */
                c=*linep;
                if (c==(char)NULL||c==' ') break;
                linep++;
    }
    *tp=(char)NULL;
    strcpy(yyval.text,token);
                        {
                        /* check if the first token is a recognized command*/
                        for( i = 0 ; yykey[i].name != "0" ; i++)
                                        if ( strcmp(yykey[i].name,token) == 0)
                                        {
                                        yyval.fnaddr = yykey[i].fnaddr;
                                        yyval.class = COMMAND;
                                        return(COMMAND);
                                        }
                        /* else is a simple argument */
                        return(IDENTF);
                        }
}
```

Αρχείο sh.h

```
#define PROMPT                  "FTP>  \b"

//type of lactic unit

#define COMMAND                 -55
#define IDENTF                  -66
#define EOL                     -77




//faction and command

struct kwtab {
      char *name;               /* name of command */
      void (*fnaddr)();         /* name of the function */

};


//shell variable structure

struct shglobals  {
                struct lex yyval;       // current token structure
                char linebuf[MAXLINE];  // current input line
```

```
        char token[MAXTOKEN];   // current token
        char *linep;                    // help pointer
};

struct shglobals *shglp;

//for easy variable definition

#define yyval                   (shglp->yyval)
#define linebuf                 (shglp->linebuf)
#define token                   (shglp->token)
#define linep                   (shglp->linep)
#define currdev                 (shglp->currdev)

// functions definition
void initialize(void);
void getline(void);
void process(struct lex *head);
void release(struct lex *head);
struct lex *lexical(void);
int yylex(void);
```

Αρχείο cmd.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "global.h"
#include "cmds.h"
#include "unp.h"


#define MAX_FILE_NO 10


// FUCTIONS FOR THE FTP COMMANDS

void help_cmd(struct lex *head)
{
 printf("FTP Shell    Version 1.0  Available commands are :\n\n");
    printf("exit            --  terminates current sub shell.\n");
    printf("help            --  prints list of available commands.\n");
    printf("open                    -- connect to the specified ftp host.\n");
    printf("close                   -- close current connection with a server.\n");
    printf("user                    -- login to the server.\n");
    printf("pwd                     -- prints current working directory.\n");
    printf("cwd                     -- change working directory.\n");
    printf("list                    -- list files in specified directory.\n");
    printf("get                     -- download file from server.\n");
    printf("delay                   -- set server delay between files.\n");
    printf("\n");

    release(head);
}

void exit_cmd(struct lex *head)
{
        release(head);
        exit(0);
}

void delay_cmd(struct lex *head)
{
        struct lex *argum1;
        argum1=head->next;
        char recvline[MAXLINE];
        char sentline[MAXLINE]="\n";


        if (!connected) {
                printf("Not connected to a server use open first.\n");
                return;
                        }
```

```
	if (!ulogin)		{
			printf("You are not logged in, use user first .\n");
			return;
				}
	if (argum1==NULL) {
			printf("usage: %s <seconds>\n", head->text);
			return;
	}

	snprintf(sentline,sizeof(sentline),"DELAY %s\n",argum1->text);
	ext_write(sockfd,sentline,strlen(sentline));
	strcpy(sentline,"\n");

	bzero(&recvline,sizeof(recvline));
	if(ext_read(sockfd,recvline, MAXLINE)==0)
			err_quit("connect:server terminated prematurely");
	Fputs(recvline,stdout);


	release(head);
	return;
}

void block_cmd(struct lex *head)
{
	for (; ;); //block here

	release(head);
	exit(0);
}

void printline_cmd(struct lex *head)
{
	struct lex *headd;
	headd=head;
	do
	{
			printf ("%s ",head->text);
			head=head->next;
	}while(head!=NULL);
	printf("\n");
	release(headd);

}

void open_cmd(struct lex *head)
{
	struct lex *argum1;
	argum1=head->next;
	struct lex *argum2;
	if (argum1!=NULL) argum2=argum1->next;
	char recvline[MAXLINE];

	struct sctp_sndrcvinfo sri;
	int msg_flags,rd_sz;

	if (connected) {
			printf("Already connected to a server use close first.\n");
			return;
				}
	connected=0;

	if (argum1==NULL || argum2==NULL) {
			printf("usage: %s <host-name> <port/service>\n", head->text);
			return;
	}

	sockfd= mysctp_connect(argum1->text,atoi(argum2->text));

	bzero(&recvline,sizeof(recvline));
	ext_read(sockfd, recvline, MAXLINE);

	if (strncmp("220",recvline,3)==0) connected=1;

	Fputs(recvline,stdout);
```

```
	release(head);
	return;

}


void close_cmd(struct lex *head)
{
	if (connected) {
				ext_close(sockfd);
				connected=0;
				ulogin=0;
				printf("Connection closed.\n");
				}
	else printf("Not connected use open first.\n");

	release(head);
	return;

}



/*
 * Send new user information (re-login)
 */
void user_cmd(struct lex *head)
{
	struct lex *argum1;
	argum1=head->next;
	struct lex *argum2;
	if (argum1!=NULL) argum2=argum1->next;
	char recvline[MAXLINE];
	char sentline[MAXLINE]="\n";
	char *chminus;

	if (ulogin)  {
		printf("You are logged in.\n");
		return;
			}
	if (!connected) {
		printf("Not connected to a server use open first.\n");
		return;
			}

	if (argum1==NULL || argum2==NULL) {
		printf("usage: %s <UserName> <password>\n", head->text);
		return;
	}



	snprintf(sentline,sizeof(sentline),"USER %s\n",argum1->text);
	ext_write(sockfd,sentline,strlen(sentline));
	strcpy(sentline,"\n");

	bzero(&recvline,sizeof(recvline));
	if(ext_read(sockfd,recvline, MAXLINE)==0)
		err_quit("connect:server terminated prematurely");

	if (strncmp("230",recvline,3)==0) {		Fputs(recvline,stdout);
							ulogin=1;}
	else if (strncmp("331",recvline,3)==0) {
				snprintf(sentline,sizeof(sentline),"PASS %s\n",argum2->text);
				ext_write(sockfd,sentline,strlen(sentline));
				strcpy(sentline,"\n");
				bzero(&recvline,sizeof(recvline));
				if(ext_read(sockfd,recvline, MAXLINE)==0)
					err_quit("connect:server terminated prematurely");

				if (strncmp("230",recvline,3)==0) ulogin=1;
				Fputs(recvline,stdout);
```

```
                                        }
                else Fputs(recvline,stdout);

                chminus=strchr(recvline,45);       // 45 ASCI of "-"
                if ((chminus-recvline)==3)
                                                do{
                        bzero(&recvline,sizeof(recvline));
                        if(ext_read(sockfd,recvline, MAXLINE)==0)
                                err_quit("connect: server terminated prematurely");
                        Fputs(recvline,stdout);
                        chminus=strchr(recvline,45);
                                                }while(chminus!=NULL);

                release(head);
                return;

}

/*
 * Get current working directory
 */
void pwd_cmd(struct lex *head)
{
                struct lex *argum1;
                argum1=head->next;
                char recvline[MAXLINE];
                char answerno[3];
                char sentline[MAXLINE]="\n";


                if (!connected) {
                        printf("Not connected to a server use open first.\n");
                        return;
                                }
                if (!ulogin)            {
                        printf("You are not logged in, use user first .\n");
                        return;
                                }

                if (argum1!=NULL) {
                        printf("usage: %s\n", head->text);
                        return;
                }



                snprintf(sentline,sizeof(sentline),"PWD\n");
                ext_write(sockfd,sentline,strlen(sentline));
                strcpy(sentline,"\n");

                bzero(&recvline,sizeof(recvline));
                if(ext_read(sockfd,recvline, MAXLINE)==0)
                        err_quit("connect:server terminated prematurely");

                Fputs(recvline,stdout);

                release(head);
                return;

}

/*
 * Change working directory
 */
void cwd_cmd(struct lex *head)
{
                struct lex *argum1;
                argum1=head->next;
                char recvline[MAXLINE];
                char answerno[3];
                char sentline[MAXLINE]="\n";


                if (!connected) {
                        printf("Not connected to a server use open first.\n");
```

```
                return;
                        }
        if (!ulogin)    {
                printf("You are not logged in, use user first .\n");
                return;
                        }

        if (argum1==NULL) {
                printf("usage: %s <pathname>\n", head->text);
                return;
        }



        snprintf(sentline,sizeof(sentline),"CWD %s\n",argum1->text);
        ext_write(sockfd,sentline,strlen(sentline));
        strcpy(sentline,"\n");

        bzero(&recvline,sizeof(recvline));
        if(ext_read(sockfd,recvline, MAXLINE)==0)
                err_quit("connect:server terminated prematurely");

        Fputs(recvline,stdout);

        release(head);
        return;

}



/*
 * Retrieve file list from server
 */
void list_cmd(struct lex *head)
{
        struct lex *argum1;
        argum1=head->next;
        char recvline[MAXLINE];
        char sentline[MAXLINE]="\n";
        int                     listenfd, connfd,n ;
        socklen_t               len;
        char                    buff[MAXLINE];
        char *p, *a;
        struct sockaddr_in ss;
        struct sockaddr_in *ss1;
        struct sockaddr_in *ss2;


        if (!connected) {
                printf("Not connected to a server use open first.\n");
                return;
                        }
        if (!ulogin)    {
                printf("You are not logged in, use user first .\n");
                return;
                        }

        if (argum1==NULL) {
                printf("usage: %s <remote-directory>\n", head->text);
                return;
        }



//prepare data listenning socket
listenfd=mysctp_listen(1111);

                        //for the PORT command
#define     UC(b)       (((int)b)&0xff)
                        len = sizeof(ss2);
                        bzero(&ss2,sizeof(ss2));
                        int noofaddr=sctp_getladdrs(sockfd,0, (SA **) &ss2);
                        printf("Number of locally bound addresses is %d\n",noofaddr);
                        a = (char *)&((*ss2).sin_addr);
```

```
                        /* print all bound addresses*/

                        printf ("%d,%d,%d,%d\n",UC(a[0]), UC(a[1]), UC(a[2]), UC(a[3]));
                        char *b;
                        while (noofaddr!=1)
                        {
                                ss2++;
                                b = (char *)&((*ss2).sin_addr);
                                printf ("%d,%d,%d,%d\n",UC(b[0]), UC(b[1]), UC(b[2]), UC(b[3]));
                                noofaddr--;
                        }

                        len = sizeof(ss);
                        bzero(&ss,sizeof(ss));
                        sctp_getladdrs(listenfd,0, (SA **) &ss1);
                        p=(char *)&((*ss1).sin_port);

//#define   UC(b)       (((int)b)&0xff)
                        snprintf(sentline,sizeof(sentline),"PORT %d,%d,%d,%d,%d,%d\n",
                            UC(a[0]), UC(a[1]), UC(a[2]), UC(a[3]),
                            UC(p[0]), UC(p[1]));

                        ext_write(sockfd,sentline,strlen(sentline));
                        Fputs(sentline,stdout);(void) fflush(stdout);
                        strcpy(sentline,"\n");

            bzero(&recvline,sizeof(recvline));
            if(ext_read(sockfd,recvline, MAXLINE)==0)
                        err_quit("connect:server terminated prematurely");

            Fputs(recvline,stdout);(void) fflush(stdout);



            //LIST command (we have set the listenning socket)

            snprintf(sentline,sizeof(sentline),"LIST %s\n",argum1->text);
            ext_write(sockfd,sentline,strlen(sentline));
            strcpy(sentline,"\n");

            bzero(&recvline,sizeof(recvline));
            if(ext_read(sockfd,recvline, MAXLINE)==0)
                        err_quit("connect:server terminated prematurely");

            Fputs(recvline,stdout);(void) fflush(stdout);


if (strncmp("150",recvline,3)==0)
              {
//accept the server incoming data connection

                        len = sizeof(ss);
                        bzero(&ss,sizeof(ss));
                        printf("waiting for server to connect\n");
                        connfd = ext_accept(listenfd, (SA *)&ss, &len);
                        printf("connection from %s\n", Sock_ntop((SA *)&ss, len));

//read the data the server sent

            union sctp_notification *snp;
            struct sctp_sndrcvinfo sri;
            int msg_flags;
            int rd_sz;
            struct sctp_event_subscribe events;

            /* Enable ancillary data and notifications */
            //memset(&events,1,sizeof(events));
            events.sctp_data_io_event = 1;
            events.sctp_association_event=1; /* enable to detect socket close */
            if(ext_setsockopt(connfd, IPPROTO_SCTP, SCTP_EVENTS, &events, sizeof(events)) < 0)
            {
      perror("setsockopt SCTP_EVENTS");
      exit(1);
            }
```

```
		bzero(&sri,sizeof(sri));
		msg_flags=0;
		while ((rd_sz=Sctp_recvmsg(connfd, buff, MAXLINE,NULL,0,&sri,&msg_flags))>0)
		{
			/* Intercept notifications here */
			if(msg_flags & MSG_NOTIFICATION)
			{
		snp = (union sctp_notification *)buff; /*for later use*/
			continue;
			}

			buff[rd_sz] = 0;	/* null terminate */
			Fputs(buff, stdout);
		}


		bzero(&recvline,sizeof(recvline));
		if(ext_read(sockfd,recvline, MAXLINE)==0)
			err_quit("connect: server terminated prematurely");

		Fputs(recvline,stdout);(void) fflush(stdout);
//close all open sockets
		ext_close(connfd);
		}
		ext_close(listenfd);


		release(head);
		return;

}



/*
 * Retrieve file from server
 */
void get_cmd(struct lex *head)
{
		struct lex *tocount;
		struct lex *argum[MAX_FILE_NO];

		char recvline[MAXLINE];
		char sentline[MAXLINE]="\n";
		int				listenfd, connfd,n ;
		socklen_t		len;
		char				buff[MAXLINE];
		char *p, *a;
		struct sockaddr_in ss;
		struct sockaddr_in *ss1;
		struct sockaddr_in *ss2;
		FILE *fp[MAX_FILE_NO];

		float datarecv=0;
		int realnooffiles,i,strsz;		// for the number of files on the RETR command
		struct timeval t1,t2;
		long dt;


		/* count no of files in RETR command*/
		realnooffiles=0;
		tocount=head;
		for (i=0;i<MAX_FILE_NO;i++)
		{
		if ((tocount->next)!=NULL)
			{
			tocount=tocount->next;
			realnooffiles++;
			}
		}

		printf("No of files to request: %d\n",realnooffiles);
```

```
        if (!connected) {
                printf("Not connected to a server use open first.\n");
                return;
                        }
        if (!ulogin)            {
                printf("You are not logged in, use user first .\n");
                return;
                        }

        if (realnooffiles==0) {
                printf("usage: %s <remote-files>\n", head->text);
                return;
        }

        /* fill in arguments pointers*/
        argum[0]=head->next;
        for (i=1;i<realnooffiles;i++)
        argum[i]=argum[i-1]->next;



//prepare data listenning socket
sleep(1);
listenfd=mysctp_listen(2222);

                        //for the PORT command
                        len = sizeof(ss2);
                        bzero(&ss2,sizeof(ss2));
                        int noofaddr=sctp_getladdrs(sockfd,0, (SA **) &ss2);
                        printf("Number of locally bound addresses is %d\n",noofaddr);
                        //a = (char *)&((*ss2).sin_addr);
        struct sockaddr_in tmpaddr;
        inet_pton(AF_INET, "10.0.0.2", &tmpaddr.sin_addr);
        a = (char *)&(tmpaddr.sin_addr);


                        len = sizeof(ss);
                        bzero(&ss,sizeof(ss));
                        sctp_getladdrs(listenfd,0, (SA **) &ss1);
                        p=(char *)&((*ss1).sin_port);

#define     UC(b)       (((int)b)&0xff)
                        snprintf(sentline,sizeof(sentline),"PORT %d,%d,%d,%d,%d,%d\n",
                            UC(a[0]), UC(a[1]), UC(a[2]), UC(a[3]),
                            UC(p[0]), UC(p[1]));

                        ext_write(sockfd,sentline,strlen(sentline));
                        Fputs(sentline,stdout);(void) fflush(stdout);
                        strcpy(sentline,"\n");

        bzero(&recvline,sizeof(recvline));
        if(ext_read(sockfd,recvline, MAXLINE)==0)
                err_quit("connect:server terminated prematurely");

        Fputs(recvline,stdout);(void) fflush(stdout);



        //RETR command (we have set the listenning socket)

        snprintf(sentline,sizeof(sentline),"RETR %s\n",argum[0]->text);
        for (i=1;i<realnooffiles;i++)
        {
                strsz = strlen(sentline);
                        if(sentline[strsz-1] == '\n')
                        {
                                sentline[strsz-1] = '\0';
                                strsz--;
                        }
                snprintf(sentline+strsz,sizeof(sentline)- strsz," %s\n",argum[i]->text);
        }
        ext_write(sockfd,sentline,strlen(sentline));
        Fputs(sentline,stdout);(void) fflush(stdout);
        bzero(&sentline,sizeof(sentline));
```

```
        bzero(&recvline,sizeof(recvline));
        if(ext_read(sockfd,recvline, MAXLINE)==0)
                err_quit("connect: server terminated prematurely");

        Fputs(recvline,stdout);(void) fflush(stdout);


if (strncmp("150",recvline,3)==0)
         {
//accept the server incoming data connection

                len = sizeof(ss);
                bzero(&ss,sizeof(ss));
                connfd = ext_accept(listenfd, (SA *)&ss, &len);
                printf("connection from %s\n", Sock_ntop((SA *)&ss, len));

                /* Set the receive buffer size, then fetch it and print its value.  *
                int sendbuff = 4;
                int info=ext_setsockopt(connfd, SOL_SOCKET, SO_RCVBUF, (void *)&sendbuff, sizeof(sendbuff));
                printf("info= %d\n", info);
                int optlen = sizeof(sendbuff);
                sendbuff=0;
                ext_getsockopt(connfd, SOL_SOCKET, SO_RCVBUF, &sendbuff, &optlen);
                printf("receive buffer size = %d\n", sendbuff);*/

//Create the files
for (i=0;i<realnooffiles;i++)
fp[i]=Fopen(argum[i]->text,"wb");


gettimeofday(&t1,NULL);
datarecv=0;
//read the data the server sent and save it to the file

        union sctp_notification *snp;
        struct sctp_sndrcvinfo sri;
        int msg_flags;
        struct sctp_event_subscribe events;

        /* Enable ancillary data and notifications */
        //memset(&events,1,sizeof(events));
        events.sctp_data_io_event = 1;
        events.sctp_association_event=1; /* enable to detect socket close */
        if(ext_setsockopt(connfd, IPPROTO_SCTP, SCTP_EVENTS, &events, sizeof(events)) < 0)
        {
    perror("setsockopt SCTP_EVENTS");
    exit(1);
        }
        bzero(&sri,sizeof(sri));
        msg_flags=0;

printf("MAXLINE= %d\n", MAXLINE);

        while ( (n = Sctp_recvmsg(connfd, buff, MAXLINE,NULL,0,&sri,&msg_flags))>0)
        {

                /* Intercept notifications here */
                if(msg_flags & MSG_NOTIFICATION)
                {
                snp = (union sctp_notification *)buff; /*for later use*/
                        continue;
                }

                datarecv+=n;
                buff[n] = 0;            /* null terminate */


                fwrite(buff,sizeof(char),n, fp[sri.sinfo_stream]);


                        gettimeofday(&t2,NULL);
                        dt = (t2.tv_sec-t1.tv_sec)*1000000+(t2.tv_usec-t1.tv_usec);
                if (dt>500000)
                {
```

```
                                    printf("---- STATUS ---- Rate:%.2f MBytes/sec ----\n",(datarecv/dt));
                                    (void) fflush(stdout);
                                    gettimeofday(&t1,NULL);
                                    datarecv=0;
                        }
                }

                for (i=0;i<realnooffiles;i++)
                Fclose(fp[i]);

                bzero(&recvline,sizeof(recvline));
                if(ext_read(sockfd,recvline, MAXLINE)==0)
                        err_quit("connect:server terminated prematurely");

                Fputs(recvline,stdout);(void) fflush(stdout);
//close all open sockets
                ext_close(connfd);
                }
                ext_close(listenfd);




                release(head);
                return;

}
```

Αρχείο cmds.h

```
// command fuction definision

void help_cmd(struct lex *head);
void exit_cmd(struct lex *head);
void printline_cmd(struct lex *head);
void open_cmd(struct lex *head);
void close_cmd(struct lex *head);
void user_cmd(struct lex *head);
void pwd_cmd(struct lex *head);
void cwd_cmd(struct lex *head);
void list_cmd(struct lex *head);
void get_cmd(struct lex *head);
void block_cmd(struct lex *head);
void delay_cmd(struct lex *head);
```

Οποιαδήποτε άλλα αρχεία περιλαμβάνονται μέσα στον πιο πάνω κώδικα είναι ίδια με αυτά που χρησιμοποιήθηκαν στον FTP server και περιλαμβάνονται στο παράρτημα Α.

ΠΑΡΑΡΤΗΜΑ Γ – PERL SCRIPTS ΓΙΑ ΑΝΑΛΥΣΗ SCTP TRACE FILE

Τς αρχεία που ακολουθούν περιλαμβάνουν το κώδικα που χρησιμοποιήθηκε στη διπλωματική για να γίνει ανάλυση από τα trace files στο πρωτόκολλο SCTP και να μπορέσουμε να παράγουμε τις αναλυτικές γραφικές που υπάρχουν στη διπλωματική αυτή.

Αρχείο join.pl

```
#!/usr/bin/perl

while(<STDIN>) {
        if(/ IP /) {
                print "\n";
        }
        chop;
        print;
}
```

Αρχείο remove_com_con.pl

```
#!/usr/bin/perl

while(<STDIN>) {

        if(!/10.0.0.2.1234/&&!/147.102.13....12345/){
                print;
        }
}
```

Αρχείο analyse.pl

```
#!/usr/bin/perl

$firstline=1;

while(<STDIN>) {
        if(/ IP /) {

                $time = (split(' '))[0];
                @x=split(":",$time);
                $timeinsec=$x[0]*3600+$x[1]*60+$x[2];
                if ($firstline) {
                        $firstline=0;
                        $timezero=$timeinsec;
                        print "FIRST LINE with time $timezero\n";
                        }
                $timefromzero=$timeinsec-$timezero;
```

```
$size = (split(' ',(split(" length: "))[1]))[0];
chop $size;
print "$timefromzero\t$size\t";

if(/\[DATA\]/) {
        $tsn  = (split(']',(split("TSN: "))[1]))[0];
        $ssn  = (split(']',(split("SSEQ "))[1]))[0];
        $stream  = (split(']',(split("SID: "))[1]))[0];
        print "DATA\t$tsn\t$stream\t$ssn\n";
}

elsif(/\[INIT\]/) {

        print "INIT\n";
}

elsif(/\[INIT ACK\]/) {

        print "INIT ACK\n";
}

elsif(/\[SACK\]/) {
        $cumack  = (split(']',(split("cum ack "))[1]))[0];
        $a_rwnd  = (split(']',(split("a_rwnd "))[1]))[0];
        $gapacks  = (split(']',(split("#gap acks "))[1]))[0];
        $dup  = (split(']',(split("#dup tsns "))[1]))[0];
        print "SACK\t$cumack\t$a_rwnd\t$gapacks\t$dup\n";
}

elsif(/\[HB REQ\]/) {

        print "HEARTBEAT\n";
}

elsif(/\[HB ACK\]/) {

        print "HEARTBEAT ACK\n";
}

elsif(/\[ABORT\]/) {

        print "ABORT\n";
}

elsif(/\[SHUTDOWN\]/) {

        print "SHUTDOWN\n";
}

elsif(/\[SHUTDOWN ACK\]/) {

        print "SHUTDOWN ACK\n";
}

elsif(/\[ERROR\]/) {

        print "ERROR\n";
}

elsif(/\[COOKIE ECHO\]/) {
```

```
                print "COOKIE ECHO\n";
        }

        elsif(/\[COOKIE ACK\]/) {

                print "COOKIE ACK\n";
        }

        elsif(/\[ECNE\]/) {

                print "ECNE\n";
        }

        elsif(/\[CWR\]/) {

                print "CWR\n";
        }

        elsif(/\[SHUTDOWN COMPLETE\]/) {

                print "SHUTDOWN COMPLETE\n";
        }

        else { print "\n" }
    }
}
```

Αρχείο owin.pl

```
#!/usr/bin/perl

while(<STDIN>) {
    if(/DATA/) {
        $tsn_data  = (split)[3];
    }
    if(/SACK/) {
        $owin=$tsn_data-$tsn_sack;
        $time_sack = (split)[0];
        $tsn_sack  = (split)[3];
        print "$time_sack\t$owin\n";
    }
}
```

Αρχείο throughwin.pl

```
#!/usr/bin/perl
#

$USAGE="USAGE: throughwin.pl <winsize> [<fid>] < in_file > out_file \n";
$#ARGV==0 || die $USAGE;

$win = $ARGV[0];
$cnt = 0;

while(<STDIN>) {
    @field = split(" ");
```

```
		$in_time[$cnt] = $field[0];
		$in_pk_size[$cnt] = $field[1];
		$cnt++;
}

$edge = $#in_time;
for($i=$edge+1; $i<=$edge+$win; $i++) {
		$in_time[$i] = $in_time[$edge];
		$in_pk_size[$i] = 0;
}

for($i=0; $i<$edge; $i++) {
		$bytes = 0;
		for($j=$i+1; $j<=$i+$win; $j++) {
			$bytes += $in_pk_size[$j];
		}
		if($in_time[$i+$win] - $in_time[$i] > 0) {
			$throughput = $bytes/($in_time[$i+$win] - $in_time[$i]);
			print "$in_time[$i+$win-1] $throughput\n";
		}
}
```

Αρχείο filter.pl

```
#!/usr/bin/perl

while(<STDIN>) {
		if(!/^(\s+0x)/) {
			print;

		}
}
```

Για να χρησιμοποιηθούν αυτά τα scripts πρέπει να έχουμε το binary trace file και να το μετατρέψουμε σε κείμενο με χρήση του προγράμματος TCPDUMP.

```
tcpdump -r sctp_trace_file -n -vvv -p proto SCTP |./filter.pl
>output.file
```

Σημειώσεις

Παραπομπές

[1] W. Stevens, NOAO
"TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", Request for Comments: 2001, January 1997

[2] V. Jacobson,LBL, R. Braden,ISI,D. Borman, Cray Research,
"TCP Extensions for High Performance", Request for Comments: 1323, May 1992

[3] R. Stewart, Q. Xie, Motorola, K. Morneault, C. Sharp, Cisco, H. Schwarzbauer, Siemens, T. Taylor, Nortel Networks, I. Rytina, Ericsson, M. Kalla, Telcordia, L. Zhang, UCLA, V. Paxson, ACIRI,
"Stream Control Transmission Protocol", Request for Comments: 2960, October 2000

[4] R. Stewart, Cisco Systems, Inc,Q. Xie, Motorola, Inc.,L. Yarroll, TimeSys Corp, J. Wood, DoCoMo USA Labs, K. Poon,Sun Microsystems, Inc.,M. Tuexen,
"Sockets API Extensions for Stream Control Transmission Protocol", Internet-Draft, February 21, 2005

[5] J. Postel,J. Reynolds, ISI,
"FILE TRANSFER PROTOCOL (FTP)", Request for Comments: 959, October 1985

[6] Unix Network Programming, Volume 1: The Sockets Networking API, 3rd Edition, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, Addison Wesley

[7] Advanced Programming in the UNIX Environment, W. Richard Stevens, Addison-Wesley, 1993

Ευρετήριο

Ρ

Σ

Τ

Φ

Χ

Ψ