ΑΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Τμήμα Πληροφορικής

Μελέτη του πρωτοκόλλου SCTP

και ανάπτυξη σχετικών opensource προγραμμάτων.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γεωργακάκης Χρήστος

Επιβλέπον : Χαρχαλάκης Στέφανος

Θεσσαλονίκη 2008

Περιεχόμενα

Περίληψη

Στη παρούσα πτυχιακή ασχοληθήκαμε με το πρωτόκολλο **SCTP**.

Αρχικά αναφέραμε τα κύρια χαρακτηριστικά του πρωτοκόλλου αυτού και στη συνέχεια περιγράψαμε το πρωτόκολλο εκτενέστερα, συγκρίνοντας τις υπηρεσίες που αυτό υποστηρίζει με τις αντίστοιχες υπηρεσίες του TCP.

Έπειτα, περιγράψαμε την τροποποίηση 3 open-source εφαρμογών, του **openbsd-inetd**, του **ftp.proxy.1.2.3** και του **wu-ftpd**. Περιγράψαμε την λειτουργία της κάθε εφαρμογής ξεχωριστά και περιγράψαμε τις σημαντικότερες μεθόδους από κάθε εφαρμογή, ώστε να καταλάβουμε τον τρόπο που αυτή δουλεύει. Στη συνέχεια παρουσιάσαμε βήμα προς βήμα την διαδικασία τροποποίησης των εφαρμογών αυτών ώστε να μας παρέχουν την υποστήριξη για το SCTP πρωτόκολλο.

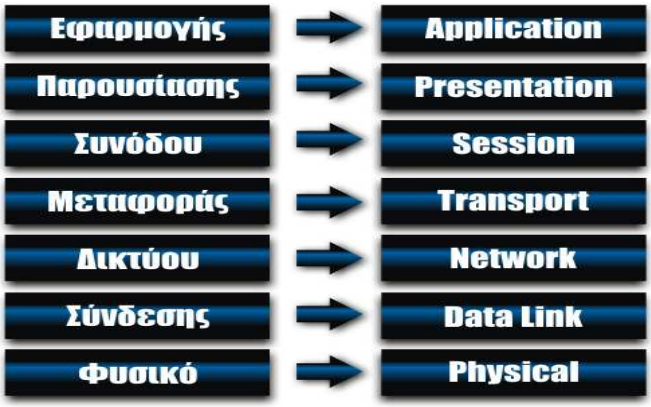
Τέλος, είδαμε πώς δουλεύει στην πράξή το FTP πρωτόκολλο μέσω του πρωτοκόλλου SCTP και κάναμε κάποιες απλές μετρήσεις μεταφοράς διάφορων αρχείων πάνω στα δύο κύρια πρωτόκολλα, το TCP και το SCTP. Με τον τρόπο αυτό συγκρίναμε τα αποτελέσματα και είδαμε τις πραγματικές δυνατότητες του SCTP πρωτοκόλλου.

1. Εισαγωγή.

1.1 Γενικά για το μοντέλο OSI και το μοντέλο TCP/IP.

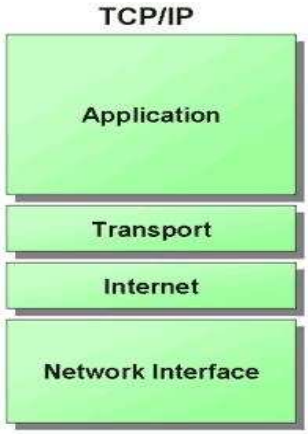
Η ανάπτυξη των υπολογιστικών συστημάτων έφερε την ανάγκη για τη διασύνδεση και επικοινωνία μεταξύ τους, είτε ως αυτόνομα, είτε ως συστήματα που ανήκουν σε διαφορετικά δίκτυα. Η διασύνδεση μεταξύ των δικτύων επιτυγχάνεται με τα πρωτόκολλα του τρίτου και τετάρτου επιπέδου του OSI.

Το **μοντέλο OSI** διατυπώθηκε αρχικά, στο τέλος της δεκαετίας του ’70, από τον Διεθνή Οργανισμό Τυποποίησης (International Organization for Standardization, ISO). Ουσιαστικά ήταν μια σειρά από οδηγίες για την αρχιτεκτονική δικτύων έτσι ώστε να υπάρχει συμβατότητα ανάμεσα στις τεχνολογίες δικτύων των διάφορων κατασκευαστών. Οι οδηγίες αυτές συνθέτουν το μοντέλο αναφοράς για τη Διασύνδεση Ανοιχτών Συστημάτων (Open Systems Interconnection, OSI).
Το μοντέλο OSI αποτελείται από επτά επίπεδα(OSI Layers) (σχήμα 1):



Σχήμα 1

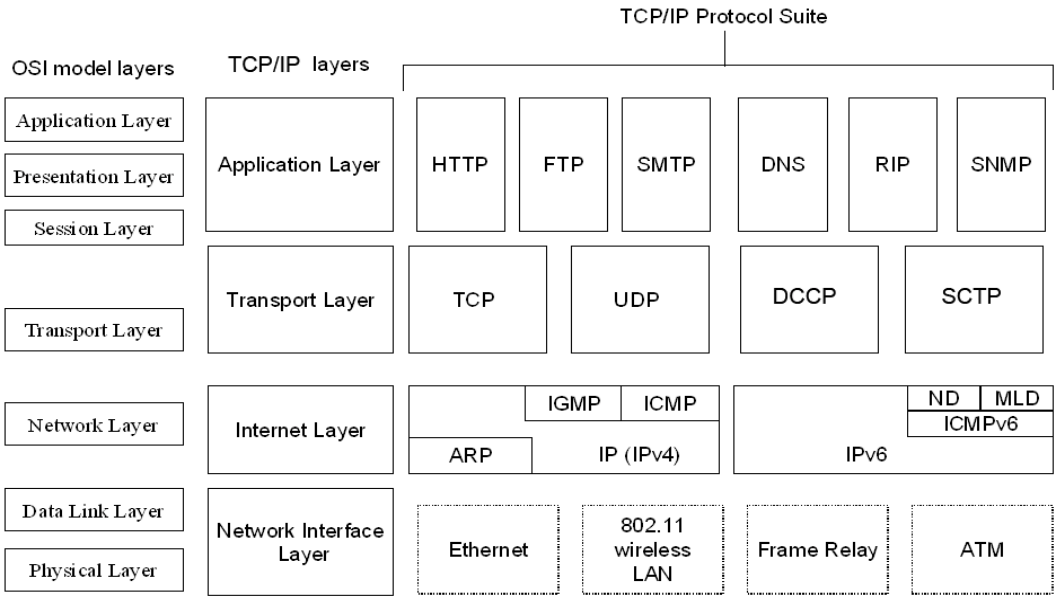
Στις αρχές της δεκαετίας του 70 η επιτροπή αμυντικών ερμηνευτικών προγραμμάτων των ΗΠΑ, DARPA(Defense Advance Resaerch Projects Agency), επιχορήγησε την ανάπτυξη τυποποιήσεων για την επικοινωνία μεταξύ υπολογιστικών συστημάτων, έτσι αναπτύχθηκαν τα **πρωτόκολλα TCP/IP**(σχήμα 2). Το TCP/IP αποτελείται από 4 επίπεδα. Επικράτησε του OSI και έγινε de facto στάνταρτ παγκοσμίως και έγινε η βάση του Διαδικτύου.



Σχήμα 2

Η ονομασία **TCP/IP** προέρχεται από τις συντομογραφίες των δυο κυριότερων πρωτοκόλλων που περιέχει : το **TCP** ή Transmission Control Protocol *(Πρωτόκολλο Ελέγχου Μετάδοσης)* και το **IP** ή Internet Protocol *(Πρωτόκολλο Διαδικτύου)*.

Στο σχήμα 3 φαίνεται η αντιστοιχία των μοντέλων OSI και TCP/IP και τα επιμέρους πρωτόκολλα σε κάθε επίπεδο.



Σχήμα 3

1.2 Χαρακτηριστικά IP

Το IP ή Internet Protocol *(Πρωτόκολλο Διαδικτύου)* είναι πρωτόκολλο του τρίτου επιπέδου του OSI και χρησιμοποιείται για τη διασύνδεση μεταξύ υπολογιστικών συστημάτων που ανήκουν στο ίδιο ή σε διαφορετικά δίκτυα.

Το Διαδίκτυο αποτελείται από πολλά φυσικά δίκτυα που συνδέονται μεταξύ τους. Το IP προωθεί το IP πακέτο (datagram) στον παραλήπτη με βάση το πεδίο της διεύθυνσης προορισμού του κάθε πακέτου. Το πακέτο φτάνει στον παραλήπτη διασχίζοντας ένα ή περισσότερα δίκτυα IP.

Το IP όπως είπαμε είναι πρωτόκολλο 3^{ου} επιπέδου και στην ουσία ασχολείται με τρεις λειτουργίες:

- Διευθυνσιοδότηση (Addressing)
- Κατακερματισμό πακέτων (Fragmentation)
- Επανασυγκόλληση κατακερματισμένων πακέτων (Defrafmantation)

Το πρωτόκολλο IP χρησιμοποιεί μια υπηρεσία που χαρακτηρίζεται ως ένα **αναξιόπιστο, βέλτιστης προσπάθειας, ασυνδεσμικό** σύστημα παράδοσης πακέτων.

Αναξιόπιστο (unreliable) γιατί:

Δεν εξασφαλίζει την απ'άκρο εις άκρο ακεραιότητα των δεδομένων. Το πακέτο μπορεί να χαθεί, να καθυστερήσει, να αναπαραχθεί και να φτάσει στον παραλήπτη με λάθος σειρά. Το IP δεν μπορεί να ανιχνεύσει τέτοιες καταστάσεις ούτε να ενημερώσει τον αποστολέα και τον παραλήπτη για το τι συμβαίνει κατά τη διάρκεια της αποστολής των πακέτων.

Ασυνδεσμικό (connectionless) γιατί:

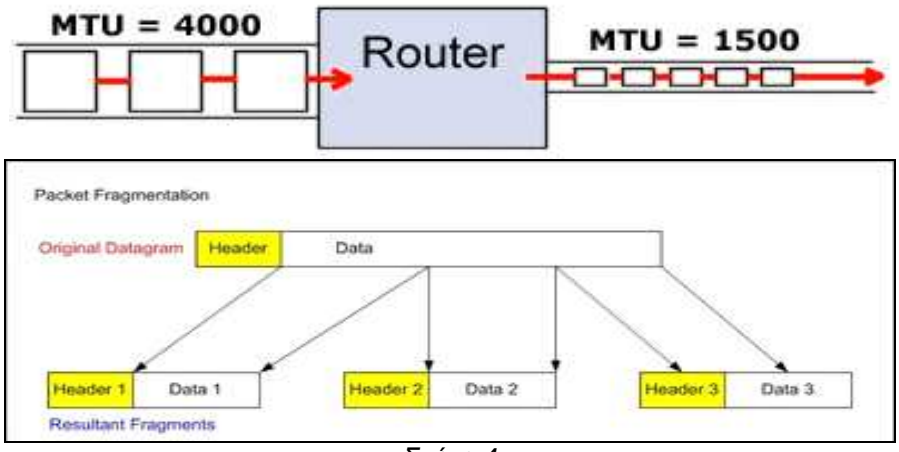
Το κάθε πακέτο αντιμετωπίζεται ξεχωριστά από τα υπόλοιπα. Έτσι κάθε πακέτο ακολουθεί διαφορετική διαδρομή μέχρι να φτάσει στον παραλήπτη. Ο παραλήπτης πιθανότατα θα παραλάβει τα πακέτα με διαφορετική σειρά από αυτή που τα έστειλε ο αποστολέας. Ακόμα είναι πιθανό κάποια από τα πακέτα να χαθούν κατά τη μεταφορά. Γι'αυτό λόγω του ότι δεν υποστηρίζει τεχνικές επανεκπομπής δεδομένων ούτε και έλεγχο ροής, αφήνει τις λειτουργίες αυτές για τα πρωτόκολλα του 4^{ου} επιπέδου(όπως τα TCP, SCTP).

Βέλτιστης προσπάθειας (best effort) γιατί:

Το λογισμικό του δικτύου κάνει φιλότιμες προσπάθειες να παραδώσει τα πακέτα. Η αναξιοπιστία δηλαδή προκύπτει μόνο όταν εξαντλούνται οι πόροι ή παθαίνουν βλάβη τα υποστηρικτικά δίκτυα.

Το IP παραλαμβάνει τα δεδομένα από το επίπεδο μεταφοράς. Τα δεδομένα αυτά έχουν μέγιστο μέγεθος 64 Kbytes. Το IP μεταδίδει αυτά τα δεδομένα μέσω του δικτύου. Αν κατά τη μεταφορά τα πακέτα αυτά περνάν από

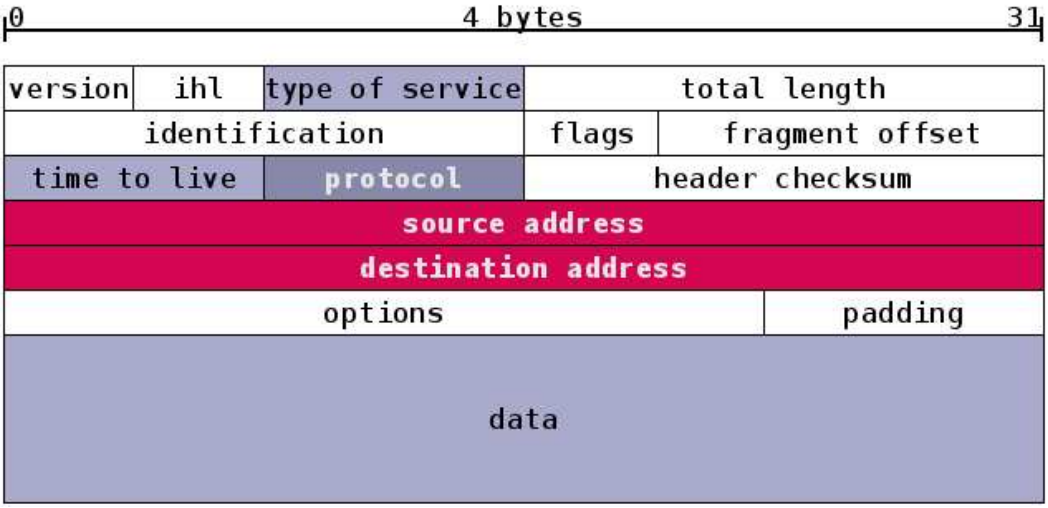
δίκτυο με περιορισμό στο μέγιστο μέγεθος πλαισίου,αν για παράδειγμα τα πακέτα πρέπει να μεταφερθούν από δίκτυο με μέγιστο μέγεθος πλαισίου 1500 byte, τότε το IP κατακερματίζει τα πακέτα και τα στέλνει στο δίκτυο αυτό(σχήμα 4). Το μέγιστο μέγεθος πλαισίου καλείται και ***MTU*** (Maximum Transfer Unit).



Σχήμα 4

Ο κατακερματισμός γίνεται από οποιαδήποτε ενδιάμεση συσκευή (π.χ. Router) του δικτύου ενώ η επανασυγκόλληση των IP πακέτων γίνεται από τον τελικό παραλήπτη.

Στο σχήμα 5 βλέπουμε τη μορφή ενός πακέτου IP.

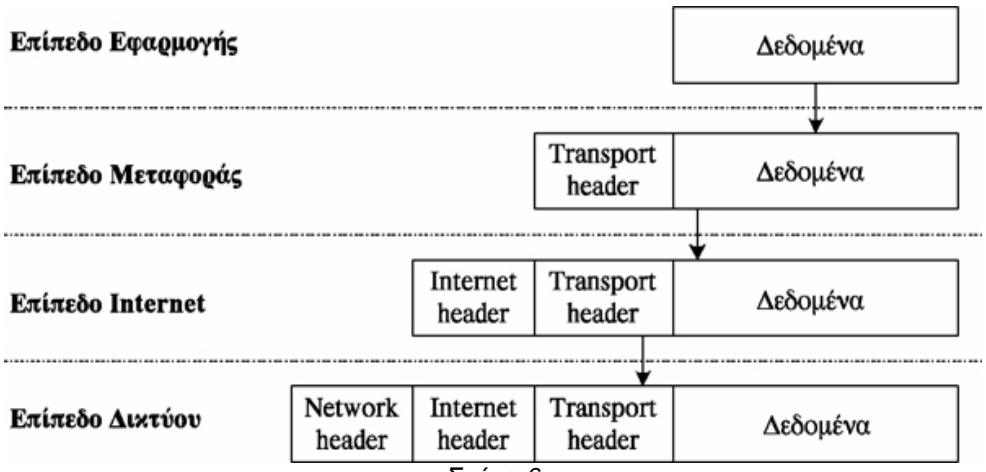


Σχήμα 5

Όπως βλέπουμε και από το πακέτο, το IP πρωτόκολλο ενθυλακώνει (encapsulates) το πακέτο του προηγούμενου επιπέδου (επίπεδο μεταφοράς), προσθέτοντας στην επικεφαλίδα(header) πληροφορίες όπως διεύθυνση

αποστολέα,διεύθυνση παραλήπτη, πρωτόκολλο 4ου επιπέδου(ποιό πρωτόκολλο μεταφοράς χρησιμοποιείται), και άλλες πληροφορίες.

Στο σχήμα 6 βλέπουμε την διαδικασία της ενθυλάκωσης των δεδομένων σε κάθε επίπεδο



Σχήμα 6

1.3 TCP

Το πρωτόκολλο TCP ή ***Transmission Control Protocol*** *(Πρωτόκολλο Ελέγχου Μετάδοσης)* είναι ένα από τα βασικά πρωτόκολλα του επιπέδου μεταφοράς (3^{ο} επίπεδο TCP/IP). Σκοπός του πρωτοκόλλου TCP είναι να επιβεβαιώνεται η αξιόπιστη(reliable) αποστολή και λήψη δεδομένων, να μεταφέρονται τα δεδομένα χωρίς λάθη μεταξύ του στρώματος δικτύου (network layer) και του στρώματος εφαρμογής (application layer) και, φτάνοντας στο πρόγραμμα του στρώματος εφαρμογής, να έχουν σωστή σειρά. Λόγω της αξιοπιστίας που προσφέρει το πρωτόκολλο TCP, Οι περισσότερες σύγχρονες υπηρεσίες στο Διαδίκτυο βασίζονται σ'αυτό. Για παράδειγμα το SMTP (Simple Mail Transfer Protocol), Telnet, το FTP (File Transfer Protocol) και πιο σημαντικό το HTTP (Hypertext Transfer Protocol), γνωστό ως υπηρεσίες World Wide Web (WWW - Παγκόσμιος Ιστός). Το TCP χρησιμοποιείται σχεδόν παντού, για αμφίδρομη επικοινωνία μέσω δικτύου.

1.3.1 Χαρακτηριστικά TCP

Τα βασικά χαρακτηριστικά του TCP πρωτοκόλλου είναι τα παρακάτω:

- **Εξασφαλίζει ότι τα πακέτα του φτάνουν στον παραλήπτη με την ίδια σειρά που τα έστειλε ο αποστολέας.**

  Το TCP βάζει τα πακέτα που φθάνουν από το επίπεδο Internet στη σωστή σειρά και τα μεταβιβάζει στον παραλήπτη.

- **Δημιουργεί ένα «εικονικό κύκλωμα» ανάμεσα στον αποστολέα και τον παραλήπτη.**

  Το TCP είναι ένα **connection oriented** πρωτόκολλο, που σημαίνει ότι πριν τη διαδικασία αποστολής/λήψης των δεδομένων, ιδρύεται ανάμεσα στον αποστολέα και τον παραλήπτη ένα εικονικό κύκλωμα και μέσω αυτού γίνεται η επικοινωνία. Είναι κάτι ανάλογο με μια τηλεφωνική κλήση. Πρώτα ιδρύεται η σύνδεση και μετά έχουμε επικοινωνία.

- **Παρέχει Έλεγχο Συμφόρησης (congestion control).**

  Το TCP με αυτή την λειτουργία ελέγχει τον ρυθμό με τον οποίο τα δεδομένα δίνονται στο δίκτυο ανάλογα με την «κατάσταση» του δικτύου. Γι' αυτόν τον λόγο περιλαμβάνει διάφορους συγκεκριμένους αλγορίθμους που έχουν ως σκοπό είτε να αποφύγουν εξ αρχής τη συμφόρηση, είτε να αποκριθούν σε αυτή. Χρησιμοποιούνται διάφοροι μηχανισμοί για να επιτευχθεί υψηλή απόδοση και να μην υπερφορτωθεί το δίκτυο.
  Σε ένα δίκτυο η συμφόρηση λαμβάνει χώρα σε δύο περιπτώσεις: α) Όταν ένας γρήγορος υπολογιστής στέλνει τα δεδομένα στο δίκτυο με ταχύτητα μεγαλύτερη από αυτή με την οποία μπορεί να μεταφέρει το ίδιο το δίκτυο, και β) όταν πολλοί υπολογιστές στέλνουν ταυτόχρονα δεδομένα σε μια συγκεκριμένη διεύθυνση προορισμού.

- **Παρέχει έλεγχο ροής (flow control).**

  Το κάθε άκρο μίας TCP σύνδεσης έχει ένα προκαθορισμένο χώρο μνήμης (**buffer**). Αυτός ο buffer αποθηκεύει τα πακέτα μέχρι να γεμίσει. Όταν γεμίσει τότε το tcp στέλνει όλα τα πακέτα στον παραλήπτη.
  Ο παραλήπτης γνωστοποιεί το μέγεθος του input buffer (window) στον αποστολέα. Το TCP του παραλήπτη επιτρέπει στον αποστολέα να στείλει τόσα δεδομένα όσα μπορεί να αποθηκεύσει ο buffer. Αυτό αποτρέπει την περίπτωση ένας γρήγορος υπολογιστής να στέλνει περισσότερα δεδομένα από όσα μπορεί να αποθηκεύσει ο buffer ενός αργού υπολογιστή. Έτσι ελέγχεται ο ρυθμός με τον οποίο ο αποστολέας στέλνει τα δεδομένα. Η μέθοδος αυτή ονομάζεται «έλεγχος ροής» και εγγυάται την ακεραιότητα των δεδομένων.

- **Επιτρέπει τα δεδομένα να πάρουν τη μορφή του «τμήματος» του οποίου το μέγεθος είναι μεταβλητό.**

  Το TCP «τεμαχίζει» τα πακέτα του επιπέδου Internet σε τμήματα (segments) και αποφασίζει για το μέγεθος του κάθε segment. Έτσι δημιουργεί ακολουθίες (sequences): πακέτα
  διαιρούνται σε segments, συναρμολογούνται στο
  προορισμό, απορρίπτονται διπλά αντίγραφα (duplicates).

- **Εξασφαλίζει αξιοπιστία με επανεκπομπή (retransmission) μη-παραληφθέντων πακέτων.**

  Το TCP θέτει χρονικά όρια μέσα στα οποία περιμένει επιβεβαίωση από το άλλο άκρο ότι τα πακέτα παραλήφθηκαν. Αν δεν έρθει επιβεβαίωση μέσα σε αυτά τα χρονικά όρια τα πακέτα αποστέλλονται ξανά.

- **Παρέχει μια πλήρως Αμφίδρομη Σύνδεση (Full Duplex Connection).**

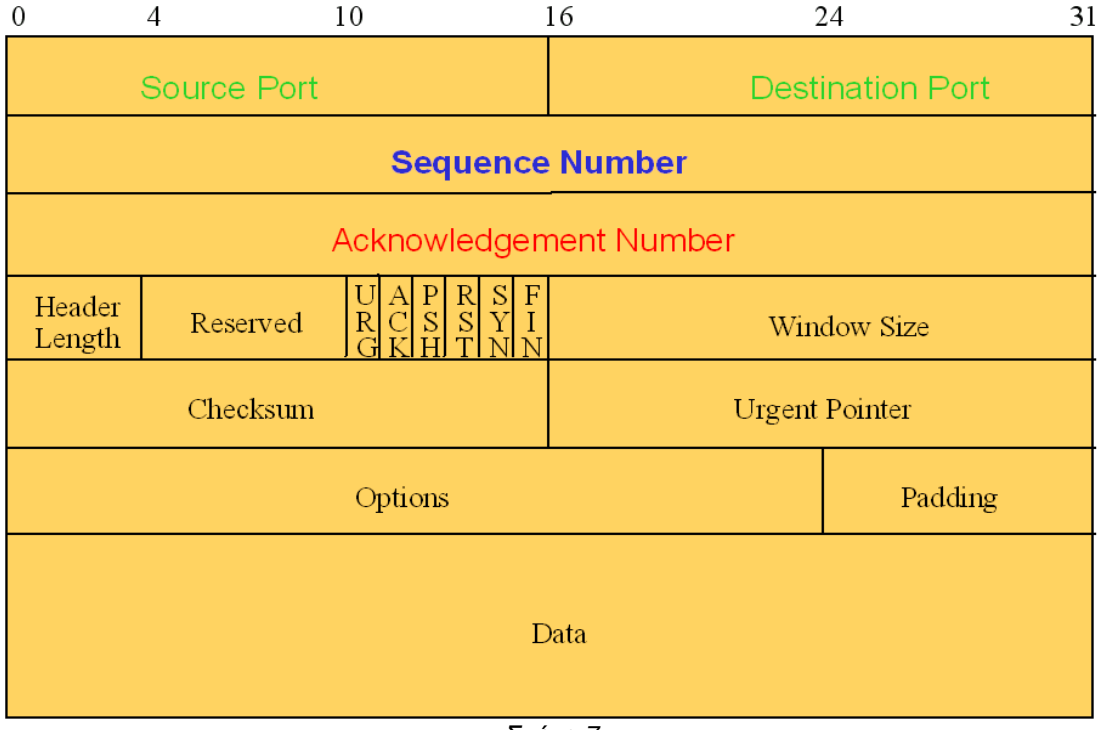
  Οι συνδέσεις που παρέχονται από το TCP δίνουν τη δυνατότητα ταυτόχρονης μεταφοράς δεδομένων και προς τις δύο κατευθύνσεις. Τέτοιου είδους συνδέσεις ονομάζονται αμφίδρομες (Full Duplex).

Αυτή η «πολυπλεξία» επιτυγχάνεται με τις «***Θύρες» (ports)***. Η κάθε σύνδεση στο TCP έχει το δικό της port και έτσι το TCP μπορεί να ξεχωρίζει και να ελέγχει την κάθε σύνδεση. Η θύρα σε συνδυασμό με την IP διεύθυνση δημιουργούν τη διεύθυνση του επιπέδου μεταφοράς που μέσω αυτής «καταλαβαίνει» το TCP την εφαρμογή που εκτελείται από τον συγκεκριμένο υπολογιστή.

1.3.2 Δομή του TCP πακέτου

Για να ελέγχεται η επικοινωνία ανάμεσα στον client και τον server, το TCP τεμαχίζει τα IP πακέτα σε τμήματα (segment).

Το σχήμα 7 δείχνει την μορφή ενός **TCP segment**:



Σχήμα 7

- **Source port** (16 bits): Προσδιορίζει την port (θύρα) του αποστολέα.
- **Destination port** (16 bits): Προσδιορίζει την port (θύρα) του παραλήπτη.
- **Sequence number** (32 bits): Αριθμός που χρησιμοποιείται για να εξασφαλιστεί η σωστή ακολουθία των δεδομένων που αποστέλλονται.

  Ο sequence number (αριθμός ακολουθίας) έχει διπλό ρόλο:

  1. Όταν SYN =1 τότε είναι ο αρχικός αριθμός ακολουθίας (ISN - initial sequence number) και η πρώτη οκτάδα δεδομένων του πακέτου είναι ο ISN+1.

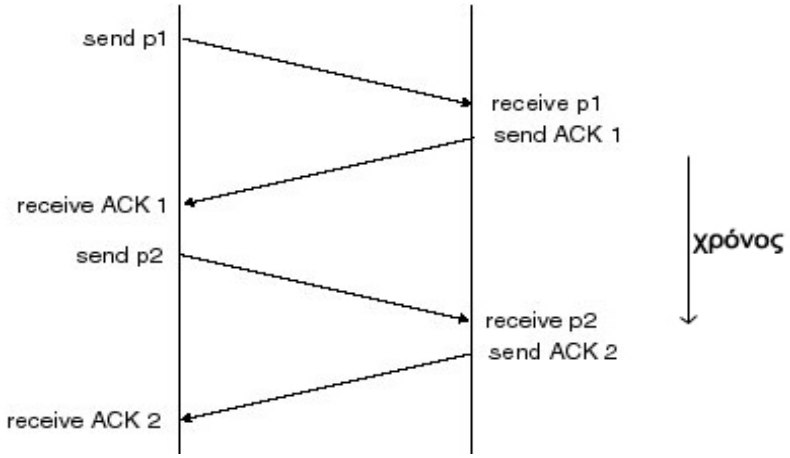
  2. Αλλιώς, όταν SYN =0 , τότε η πρώτη οκτάδα δεδομένων είναι ο αριθμός ακολουθίας.

- **Acknowledgement number** (32 bits): Όταν ACK=1 η τιμή αυτού του πεδίου δείχνει τον επόμενο sequence number (αριθμό ακολουθίας) που αναμένει ο αποστολέας.
- **Header Length (ή Data offset)** (4 bits): Καθορίζει το μέγεθος της επικεφαλίδας (πολλαπλάσιο του 32) και επομένως δείχνει και την αρχή των δεδομένων.
- **Reserved** (6 bits): Πεδίο 6 "κρατημένων" bit για μελλοντική χρήση. Το πεδίο αυτό δεν χρησιμοποιείται προς το παρόν.
- **Flags** (6x1 bit): Περιέχει τις σημαίες(flags):
  - **URG**: Αν έχει την τιμή 1 τότε τα πακέτα πρέπει να εξυπηρετηθούν επειγόντως.
  - **ACK**: Αν έχει την τιμή 1 τότε το πακέτο είναι ένα acknowledgement.
  - **PSH** (PUSH): Αν έχει την τιμή 1 τότε το συγκεκριμένο πακέτο θα αποσταλεί κατευθείαν στον παραλήπτη χωρίς να μπει στον buffer.
  - **RST**: Αν έχει την τιμή 1 τότε η σύνδεση τερματίζεται και αρχίζει πάλι από την αρχή (reset).
  - **SYN**: Η SYN σημαία καθορίσει την ίδρυση μιας σύνδεσης.
  - **FIN**: Αν έχει την τιμή 1 η σύνδεση διακόπτεται.
- **Window** (16 bits): Το πεδίο αυτό καθορίζει τον αριθμό των byte που μπορούν να σταλούν στον αποστολέα χωρίς επιβεβαίωση (acknowledgement).
- **Checksum (ή CRC)** (16 bits) : Το πεδίο αυτό χρησιμοποιείται για έλεγχο λαθών στην επικεφαλίδα και στα δεδομένα.
- **Urgent pointer** (16 bits): Εάν είναι ενεργοποιημένο το URG bit ελέγχου, τότε αυτό το πεδίο δείχνει τον αριθμό ακολουθίας (sequence number) των bytes που βρίσκονται αμέσως μετά το τελευταίο byte από τα επείγοντα δεδομένα.
- **Options** (variable size): Μεταβλητή, η οποία καθορίζει ειδικές επιλεγόμενες ρυθμίσεις.
- **Padding**: Ο χώρος που απομένει μετά τη σημαία options. Το μήκος της κεφαλίδας πρέπει να είναι πολλαπλάσιο των 8 bit γι'αυτό στο τέλος

προστίθενται μηδενικά-0. Έτσι το data offset θα δείχνει σωστά την αρχή των δεδομένων.

1.3.3 Παροχή αξιοπιστίας

Το TCP όπως αναφέραμε προηγουμένως εγγυάται την παράδοση των δεδομένων χωρίς την απώλεια πακέτων ή τον διπλασιασμό των δεδομένων. Η αξιοπιστία της μεταφοράς των δεδομένων επιτυγχάνεται με την τεχνική της «θετικής επιβεβαίωσης με αναμετάδοση» (positive acknowledgement with retransmission). Ο παραλήπτης επικοινωνεί με την προέλευση στέλνοντας ένα μήνυμα *επιβεβαίωσης* (acknowledgement - ACK) κατά τη λήψη των δεδομένων. Ο αποστολέας έτσι καταγράφει το κάθε segment που στέλνει και περιμένει να λάβει από τον παραλήπτη την επιβεβαίωση προτού στείλει το επόμενο segment (σχήμα 8).

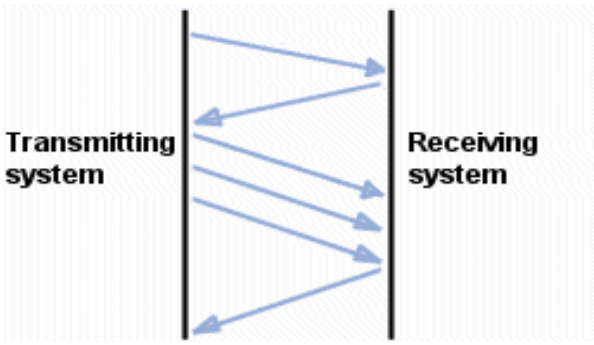


Σχήμα 8

Για να γνωρίζει πότε ένα πακέτο μπορεί να έχει χαθεί το TCP 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. Έτσι κατά την μεταφορά των πακέτων ο αποστολέας ξεκινά αυτόν τον χρόνο μεταφοράς (timer) για κάθε πακέτο. Όταν αυτός ο χρόνος αυτός λήξει και δεν έχει πάρει ο αποστολέας την αναμενόμενη επιβεβαίωση, τότε ο αποστολέας θεωρεί ότι το πακέτο χάθηκε και το μεταδίδει ξανά το πακέτο στον παραλήπτη. Με την αποστολή του πακέτου ο αποστολέας ξεκινά πάλι τον timer. Ο timer αυτή τη φορά αρχικοποιείται στον διπλάσιο από τον προηγούμενο χρόνο.

Με την επαναποστολή των πακέτων λόγω λήξης του χρονομέτρου όμως δημιουργείται το πρόβλημα του διπλασιασμού των πακέτων(duplicate). Τα διπλά πακέτα προκαλούνται κι όταν υπάρχουν μεγάλες καθυστερήσεις σε ένα δίκτυο.
Για την αποφυγή των «διπλών πακέτων» το TCP χρησιμοποιεί έναν **αριθμό ακολουθίας (sequence number)** τον οποίο και εκχωρεί σε κάθε πακέτο. Με την αποστολή κάθε πακέτου ο αριθμός αυτός αυξάνεται κατά 1. Μόλις ένα πακέτο φτάσει στον προορισμό του, ο παραλήπτης στέλνει μια επιβεβαίωση παραλαβής πίσω στον αποδέκτη όπου πάνω υπάρχει ο (μοναδικός) αριθμός ακολουθίας του πακέτου αυτού.

Όπως έχουμε αναφέρει, μια από τις σημαντικότερες υπηρεσίες του TCP είναι ο ***έλεγχος ροής (flow control)***. Η διαδικασία αυτή είναι γνωστή και ως *windowing*. 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.
Πρακτικά *μέγεθος του παραθύρου (window size)* καθορίζει το μέγεθος των δεδομένων/πακέτων που μπορούν να σταλούν από τον αποστολέα πριν ο αποδέκτης στείλει επιβεβαίωση (ACK). Έτσι στην αποστολή δεδομένων που δείχνει το σχήμα 8, έχουμε window size ίσο με 1. Το TCP χρησιμοποιεί την μέθοδο των *συρόμενων παραθύρων (sliding window)* η οποία επιτρέπει σε πολλαπλά πακέτα δεδομένων να μεταφέρονται ταυτόχρονα για να χρησιμοποιείται αποδοτικότερα η bandwidth (εύρος ζώνης) του δικτύου.
Όπως βλέπουμε στο σχήμα 9, τα πακέτα 3,4 και 5 θα σταλθούν μαζικά,χωρίς να περιμένει ο αποστολέας επιβεβαίωση για τι κάθε ένα. Ο αποστολέας αναμένει μία επιβεβαίωση μετά την αποστολή των πακέτων και για τα τρία πακέτα.



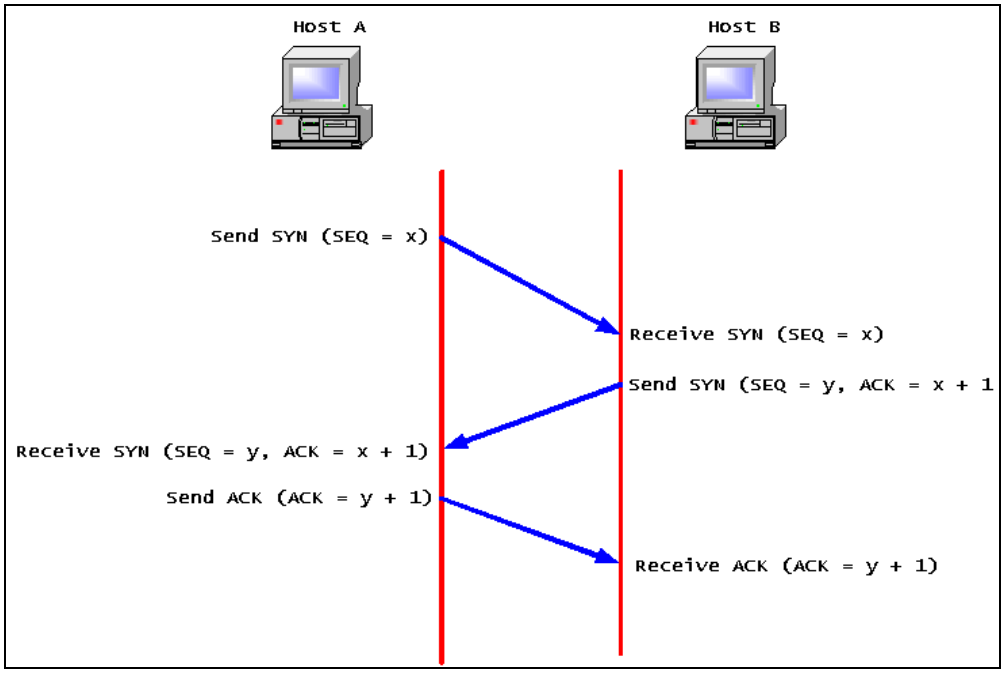
Σχήμα 9

Τέλος το TCP Παρέχει ***Έλεγχο Συμφόρησης(congestion control)***. Το TCP χρησιμοποιεί διάφορους μηχανισμούς είτε για την αποφυγή της συμφόρησης, είτε να αποκριθεί σε αυτή. Χρησιμοποιούνται διάφοροι μηχανισμοί για να επιτευχθεί υψηλή απόδοση και να μην υπερφορτωθεί το δίκτυο. Αυτοί οι μηχανισμοί περιλαμβάνουν:

- τον αλγόριθμο slow-start,
- τον αλγόριθμο congestion avoidance,
- τον αλγόριθμο fast retransmit και
- τον αλγόριθμο fast recovery

1.3.4 Τριμερής χειραψία (3-way handshake)

Όπως είπαμε το TCP απαιτεί να υπάρχει μια εδραιωμένη σύνδεση ανάμεσα σε δυο σταθμούς πριν ξεκινήσει η διαδικασία της μεταφοράς δεδομένων στην μεταξύ τους επικοινωνία. Ο σταθμός/υπολογιστής που απαιτεί τη σύνδεση ονομάζεται «πελάτης» (client) ενώ ο σταθμός/υπολογιστής που αποδέχεται τη σύνδεση ονομάζεται εξυπηρετητής. Η συγχρονισμός μεταξύ client και server επιτυγχάνεται με μια τεχνική που ονομάζεται **τριμερής χειραψία (3-way handshake)**. Ο συγχρονισμός πραγματοποιείται με τη μεταφορά των πακέτων που μεταφέρουν το αρχικό αριθμό ακολουθίας (Initial Sequence Number-ISN) και τη σημαία SYN (SYN = 1). Τα πακέτα που έχουν την σημαία SYN ονομάζονται και SYNs. Με τον συγχρονισμό, κάθε άκρο στέλνει το δικό του ISN και λαμβάνει το ISN του άλλου άκρου. Έτσι κάθε πλευρά πρέπει να λαμβάνει το ISN την άλλης πλευράς και να απαντάει με ένα ACK (acknowledgment). Γι ‘αυτόν τον λόγο η 3-way handshake χρησιμοποιεί **τρία βήματα** για να εγκαθιδρύσει μια σύνδεση (σχήμα 10).



Σχήμα 10

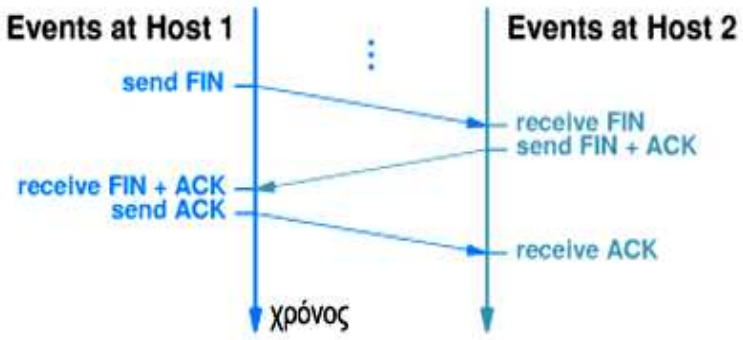
1. Αρχικά ο host A(client) στέλνει ένα πακέτο με το SYN bit ενεργοποιημένο στον host B(server). Ο client θέτει το πεδίο αριθμού ακολουθίας στην επικεφαλίδα TCP (TCP header) στον αρχικό αριθμό ακολουθίας του (ISN - initial sequence number) [seq = x].
2. Ο host B παραλαμβάνει το πακέτο και καταγράφει το sequence number του x και απαντάει και απαντάει με ένα SYN όπου στο πεδίο αριθμού ακολουθίας βάζει τον δικό του ISN [seq=y] και στο ACK βάζει την τιμή x+1. Αυτό σημαίνει ότι ο host B έλαβε όλα τα byte της πληροφορίας από τον host A έτσι ο host A περιμένει την επόμενη οκτάδα (x+1).
3. Ο host A απαντάει, αυτή τη φορά, με ένα πακέτο ACK στο οποίο βάζει την τιμή y+1. Έτσι ο host A επιβεβαιώνει τη λήψη όλων των οκτάδων από τον host B (μέσω του αριθμού y) και στέλνει την τιμή y+1 στην επιβεβαίωση στον host B ο οποίος αναμένει την επόμενη οκτάδα δεδομένων.

Σε αυτό το σημείο, τα δύο μέρη συνδέονται και μπορούν πλέον να ανταλλάξουν δεδομένα.

Όπως βλέπουμε, το TCP χρησιμοποιεί τους **αριθμούς ακολουθίας (sequence numbers)**. Με τους αριθμούς αυτούς ο παραλήπτης ξέρει πότε ο αποστολέας απαντά στην συγκεκριμένη σύνδεση. Κάθε μηχανή επιλέγει έναν τυχαίο αριθμό ακολουθίας τον οποίο θα χρησιμοποιήσει για να προσδιορίζει τα

byte που στέλνει. Και οι δύο πλευρές της επικοινωνίας αρχικά συμφωνούν σε έναν αρχικό αριθμό ακολουθίας (ISN) ώστε οι αριθμοί οκτάδων που χρησιμοποιούνται μέσα στα ACKs να συμφωνούν με τους αριθμούς που χρησιμοποιούνται στα τμήματα δεδομένων.

Ο **τερματισμός της σύνδεσης** επιτυγχάνεται με την παρακάτω διαδικασία(σχήμα 11), μία 3-way handshake με την κάθε πλευρά να τερματίζει ανεξάρτητα:



Σχήμα 11

1. Όταν κάποιο άκρο επιθυμεί να κλείσει τη σύνδεση, στέλνει ένα segment με το FIN ενεργοποιημένο (FIN = 1)
2. Η άλλη πλευρά(παραλήπτης) παραλαμβάνει το πακέτο και στέλνει ένα ACK στο άλλο άκρο και στη συνέχεια, στέλνει ένα segment FIN
3. Η πλευρά που ξεκίνησε τον τερματισμό, επιβεβαιώνει το FIN στέλνοντας ένα ACK.

Με αυτόν τον τρόπο, για έναν τυπικό τερματισμό χρειάζεται ένα ζεύγος πακέτων FIN και ACK από κάθε κατεύθυνση. Μια σύνδεση μπορεί να είναι μισάνοιχτη (half-open), δηλαδή η μία πλευρά να έχει τερματίσει, όχι όμως και η άλλη. Η πλευρά που έχει τερματίσει δεν μπορεί να στείλει πλέον δεδομένα, ενώ η άλλη μπορεί.

1.4 UDP

Το UDP ή **User Datagram Protocol (Πρωτόκολλο Αυτοδύναμων πακέτων Χρήστη)** είναι πρωτόκολλο 3^{ου} επιπέδου. Παρέχει τον βασικό μηχανισμό που χρησιμοποιούν τα προγράμματα εφαρμογών για την αποστολή αυτοδύναμων πακέτων(datagrams). Όπως και το TCP έτσι και το UDP χρησιμοποιεί το IP για να μεταφέρει μηνύματα. Το UDP σε αντίθεση με το TCP παρέχει μια αναξιόπιστη(unreliable) – ασυνδεσματική(connectionless) υπηρεσία παράδοσης πακέτων.

- **Αναξιόπιστη** γιατί δεν χρησιμοποιεί επιβεβαιώσεις(ACKs) για να σιγουρευτεί ότι τα μηνύματα έφτασαν στον προορισμό, δεν ταξινομεί τα εισερχόμενα μηνύματα και δεν παρέχει επανεκπομπή των μηνυμάτων.
- **Ασυνδεσματική** (connectionless) υπηρεσία σημαίνει ότι το πρωτόκολλο δεν απαιτεί από τα δύο άκρα να ιδρύσουν κάποια σύνδεση πριν την αποστολή δεδομένων, όπως γίνεται με τα connection-oriented πρωτόκολλα όπως το TCP.

Έτσι το UDP δεν παρέχει αξιοπιστία στην επικοινωνία. Τα πακέτα μπορεί να φτάσουν στον παραλήπτη με λάθος σειρά, να φτάσουν διπλά ή να μην φτάσουν καθόλου. Η έλλειψη της αξιοπιστίας από την άλλη καθιστά το πρωτόκολλο UDP αρκετά πιο γρήγορο και αποτελεσματικό απ'ότι το TCP, τουλάχιστον για τις εφαρμογές εκείνες που δεν απαιτούν αξιόπιστη επικοινωνία.

1.4.1 Χαρακτηριστικά του UDP

Τα κύρια **χαρακτηριστικά του UDP** είναι τα εξής:

- **Αναξιόπιστο** – Όπως είπαμε δεν υπάρχει η δυνατότητα επιβεβαίωσης λήψης πακέτου από τον παραλήπτη, ούτε η επαναμετάδοση ενός χαμένου πακέτου.
- **Δεν υπάρχει σειρά** - Τα πακέτα UDP, σε αντίθεση με το TCP, δεν αριθμούνται και κατά συνέπεια δεν υπάρχει κάποια συγκεκριμένη σειρά με την οποία θα πρέπει να φτάσουν στον παραλήπτη.
- **Ελαφρύ** – Το UDP δεν παρέχει την αξιοπιστία του TCP για τον λόγο ότι δεν εφαρμόζει τους μηχανισμούς αξιόπιστης επικοινωνίας που υπάρχουν στο δεύτερο. Οι μηχανισμοί αυτοί του TCP επιβαρύνουν την ταχύτητα αποστολής των δεδομένων μιας και κάθε πακέτο ελέγχεται ξεχωριστά. Έτσι το UDP είναι αρκετά πιο γρήγορο από το TCP.

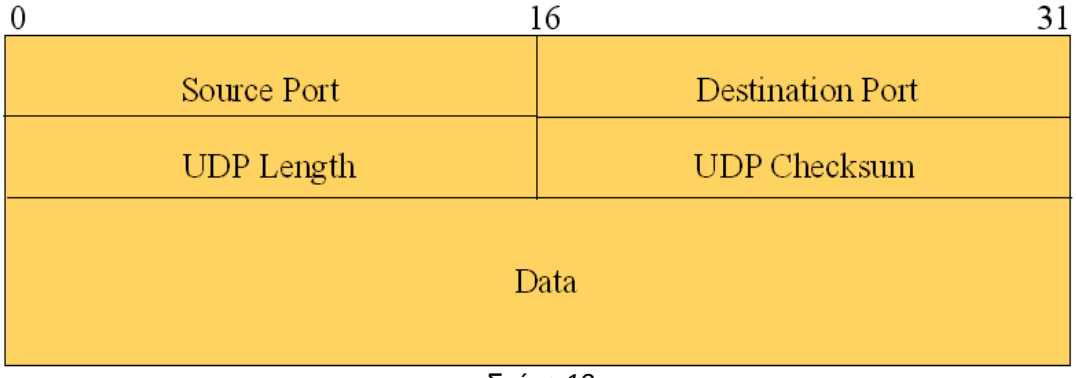
- **Datagrams** - Κάθε πακέτο UDP ονομάζεται επίσης και "datagram", θεωρείται δε ως μεμονωμένη οντότητα που θα πρέπει να μεταδοθεί ολόκληρη. Γι αυτό και έχουμε μια connectionless επικοινωνία.

1.4.2 Ανάγκη για UDP

Οι εφαρμογές *audio* και *video streaming* χρησιμοποιούν κατά κόρον πακέτα UDP. Για τις εφαρμογές αυτές είναι πολύ σημαντικό τα πακέτα να παραδοθούν στον παραλήπτη σε σύντομο χρονικό διάστημα ούτως ώστε να μην υπάρχει διακοπή στην ροή του ήχου ή της εικόνας. Έτσι χρησιμοποιούν το πρωτόκολλο UDP ώστε τα δεδομένα να μεταφερθούν γρήγορα. Τα προγράμματα εφαρμογής που χρησιμοποιούν UDP είναι υπεύθυνα για τον χειρισμό του προβλήματος της αξιοπιστίας. Έτσι στην περίπτωση που χαθεί κάποιο πακέτο, οι εφαρμογές αυτές διαθέτουν κάποιους ειδικούς μηχανισμούς διόρθωσης και παρεμβολής ούτως ώστε ο τελικός χρήστης να μην παρατηρεί καμία αλλοίωση ή διακοπή στην ροή του ήχου και της εικόνας λόγω του χαμένου πακέτου. Σε αντίθεση με το TCP, το UDP υποστηρίζει *broadcasting*, δηλαδή την αποστολή ενός πακέτου σε όλους τους υπολογιστές ενός δικτύου, και *multicasting*, δηλαδή την αποστολή ενός πακέτου σε συγκεκριμένους υπολογιστές ενός δικτύου.

1.4.3 Δομή του UDP πακέτου

Στο σχήμα 12 φαίνεται η μορφή ενός **UDP πακέτου**:



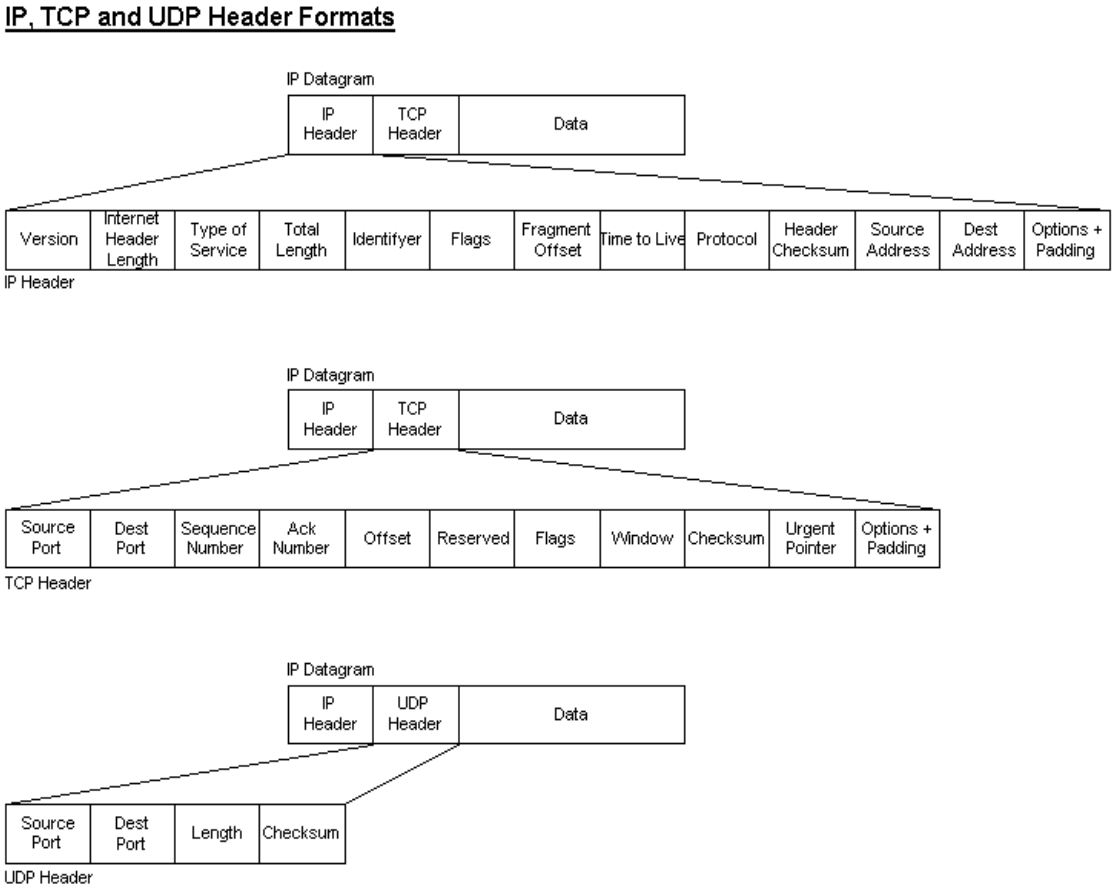
Σχήμα 12

- **Source port (16 bit) :** Η πόρτα του αποστολέα. Εάν ο παραλήπτης επιθυμεί να στείλει κάποια απάντηση, θα πρέπει να την στείλει στην πόρτα

αυτήν. Το συγκεκριμένο πεδίο δεν είναι υποχρεωτικό και στις περιπτώσεις που δεν χρησιμοποιείται θα πρέπει να έχει την τιμή μηδέν.

- **Destination port (16 bit) :** Η πόρτα του παραλήπτη στην οποία θα πρέπει να παραδοθεί το πακέτο.
- **Length (16 bit) :** Το μέγεθος του πακέτου σε bytes. Το μικρότερο δυνατό μέγεθος είναι 8 bytes, αφού η κεφαλίδα αυτή καθ' αυτή καταλαμβάνει τόσο χώρο. Το μεγαλύτερο μέγεθος ενός UDP πακέτου δεν μπορεί να ξεπερνάει τα 65,527 bytes, αλλά πρακτικά το όριο μειώνεται στα 65,507 bytes λόγω διαφόρων περιορισμών που εισάγει το πρωτόκολλο IPv4 στο επίπεδο δικτύου.
- **Checksum (16 bit) :** Το πεδίο αυτό χρησιμοποιείται για έλεγχο λαθών στην επικεφαλίδα και στα δεδομένα.

Στο σχήμα 13 βλέπουμε την ενθυλάκωση των δεδομένων στις επικεφαλίδες IP, TCP και UDP.



Σχήμα 13

1.5 Σύντομη περιγραφή του πρωτοκόλλου SCTP

Το ***SCTP*** ή αλλιώς ***Stream Control Transmission Protocol*** *(πρωτόκολλο ελέγχου μετάδοσης ρευμάτων)* είναι ένα νέο πρωτόκολλο επιπέδου μεταφοράς το οποίο αναπτύχθηκε με στόχο την μεταφορά τηλεπικοινωνιακών σημάτων σε δίκτυα βασιζόμενα στο IP πρωτόκολλο. Όπως και το TCP, το SCTP είναι πρωτόκολλο με σύνδεση (connection oriented) και προσφέρει αξιόπιστες υπηρεσίες μεταφοράς, εξασφαλίζοντας ότι τα δεδομένα στο δίκτυο θα μεταδοθούν χωρίς λάθη και με τη σωστή σειρά. Πολλές από τις υπηρεσίες του TCP (όπως congestion control, flow control κτλ) θα τις βρούμε και στο SCTP πρωτόκολλο. Ωστόσο, σε αντίθεση με το TCP το SCTP μεταφέρει μηνύματα, όχι απλά bytes. Γι'αυτόν τον λόγο, το SCTP χαρακτηρίζεται και ως message-oriented πρωτόκολλο. Στο σχήμα 13 φαίνονται οι υπηρεσίες που προσφέρουν τα πρωτόκολλα μεταφοράς SCTP, TCP και UDP.

Service/Features	SCTP	TCP	UDP
Message-Oriented	yes	no	yes
Byte-Oriented	no	yes	no
Connection-Oriented	yes	yes	no
Full Duplex	yes	yes	yes
Reliable data transfer	yes	yes	no
Partially-Reliable data transfer	opt	no	no
Ordered data delivery	yes	yes	no
Unordered delivery	yes	no	yes
Flow control	yes	yes	no
Congestion Control	yes	yes	no
ECN Capable	yes	yes	no
Selective Acknowledgments	yes	opt	no
Path MTU discovery	yes	yes	no
Application PDU fragmentation	yes	yes	no
Application PDU bundling	yes	yes	no
Multistreaming	yes	no	no
Multihoming	yes	no	no
Dynamic Multihoming	opt	no	no
SYN flooding attack prevention	yes	no	n/a
Allows half-closed state	no	yes	n/a
Reach-ability check	yes	opt	no
Pseudo-header for checksum	no	yes	yes
Time wait state	no	yes	n/a
Authentication	opt	opt	no
CRC based checksum	yes	no	no

Σχήμα 13

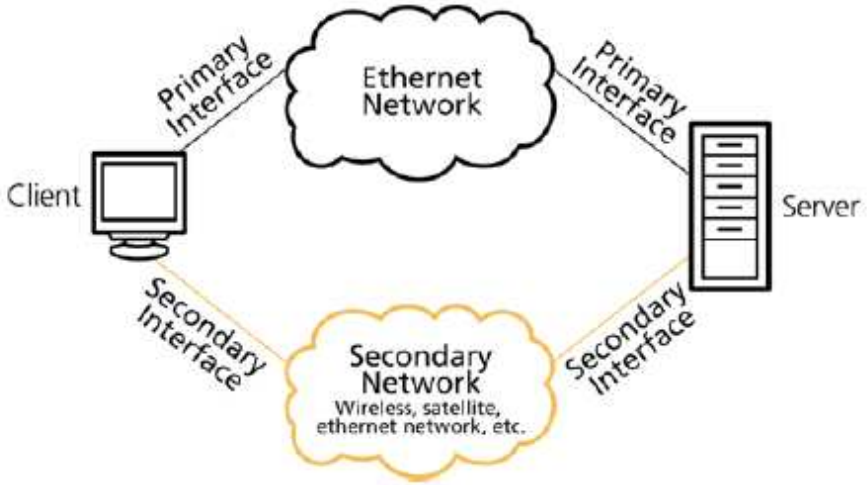
Δύο από τα πιο σημαντικά χαρακτηριστικά του SCTP είναι το Multi-Streaming και το Multi-Homing.

- Το **Multi-Streaming** επιτρέπει στα δεδομένα να τεμαχιστούν σε πολλά μικρότερα κομμάτια και να τοποθετηθούν σε πολλαπλά streams(ρεύματα ή ροές). Αυτά τα streams είναι ανεξάρτητα μεταξύ τους και έχουν την ιδιότητα να παραδίδονται 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 (σχήμα 14).

Multi-Streaming

Σχήμα 14

- Το **Multi-Homing** επιτρέπει στα άκρα μιας SCTP επικοινωνίας να έχουν πολλαπλές IP διευθύνσεις. Έτσι όταν λέμε ότι ένας host είναι multi-homed εννοούμε ότι έχει πολλές διασυνδέσεις πάνω στο IP δίκτυο (σχήμα 15). Το Multi-Homing προστατεύει τη σύνδεση από αποτυχίες του δικτύου κι αυτό γιατί σε περίπτωση βλάβης του δικτύου, το SCTP θα ψάξει για τις «εναλλακτικές» διευθύνσεις IP ώστε να συνεχιστεί η μεταφορά δεδομένων κανονικά. Μια διεύθυνση ορίζεται ως η κύρια διεύθυνση (primary address) και χρησιμοποιείται ως η διεύθυνση στην οποία όλα τα πακέτα θα αποστέλλονται. 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.

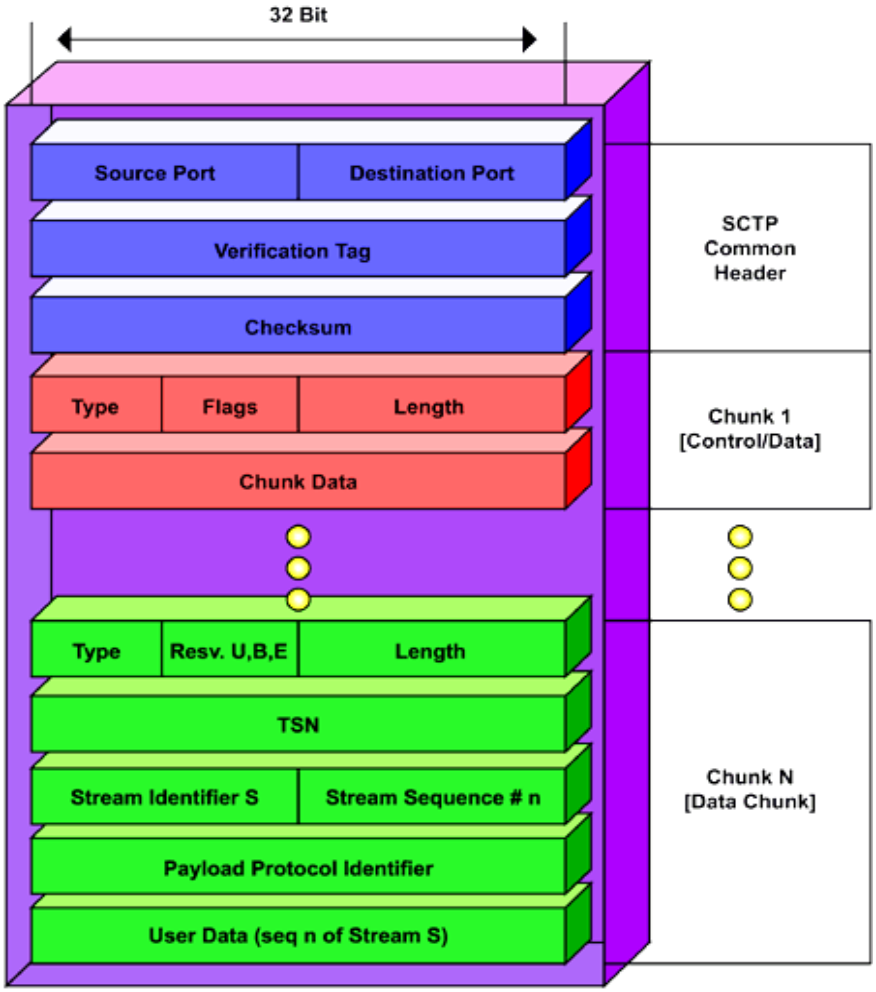


Multi-Homing

Σχήμα 15

Ένα νέο στοιχείο που λαμβάνει χώρα στο SCTP είναι η έννοια της ***συσχέτισης (association)*** η οποία είναι μια γενίκευση της σύνδεσης του TCP. Σε μια συσχέτιση τα δύο άκρα συνδέονται μεταξύ τους και ανταλλάσσουν δεδομένα. Τα δεδομένα αυτά είναι είτε δεδομένα της εφαρμογής του άκρου είτε δεδομένα ελέγχου. Με την συσχέτιση επιτυγχάνεται η ευελιξία και η αξιοπιστία της μεταφοράς των δεδομένων αυτών, όπως επίσης και η σωστή σειρά παράδοσης των πακέτων.

Το ***SCTP πακέτο*** έχει την παρακάτω μορφή (σχήμα 16):



Σχήμα 16

Καταρχάς βλέπουμε ότι τα πακέτα χωρίζονται σε ***τεμάχια (chunks)***. Κάθε chunk μπορεί να ανήκει σε διαφορετικό λογικό stream. Τόσο τα δεδομένα εφαρμογής όσο και τα δεδομένα ελέγχου βρίσκονται μέσα στα chunks. Έτσι έχουμε τα τεμάχια δεδομένων (data chunks) και τα τεμάχια ελέγχου (control chunks) αντίστοιχα. Ο αριθμός των chunks (N) καθορίζεται από το πρωτόκολλο και εξαρτάται από την MTU (Maximum Transfer Unit).

Η *επικεφαλίδα* λοιπόν περιλαμβάνει τα εξής πεδία:

- **Source port address (16 bits) :** Προσδιορίζει την port (θύρα) του αποστολέα.
- **Destination port address (16 bits) :** Προσδιορίζει την port (θύρα) του παραλήπτη.

- **Verification tag (32 bits) :** Κάθε άκρο έχει μια μοναδική τιμή στο verification tag η οποία και καθορίζει σε ποιό association ανήκει το κάθε sctp πακέτο.
- **Checksum (32 bits) :** Το πεδίο αυτό χρησιμοποιείται για έλεγχο λαθών στην επικεφαλίδα και στα δεδομένα.

Κάθε *τεμάχιο (chunk)* έχει τα ακόλουθα πεδία:

- **chunk-type field (8 bits) :** Προσδιορίζει τον τύπο του chunk.
- **chunk flag (8 bits) :** Προσδιορίζει της σημαίες που χρησιμοποιούνται στο association.
- **chunk length (16 bits) :** Προσδιορίζει το μέγεθος του chunk σε bytes.
- **chunk data :** Περιέχει τα δεδομένα του chunk.

Συνολικά υπάρχουν 14 τύποι για τα chunks. Ο ένας απ'αυτούς είναι τα data chunks. Οι υπόλοιποι 13 τύποι είναι τύποι για τα control chunks και φαίνονται στον πίνακα 1.

ID Value	*Chunk Type*
0	Payload Data (DATA)
1	Initiation (INIT)
2	Initiation Acknowledgement (INIT ACK)
3	Selective Acknowledgement (SACK)
4	Heartbeat Request (HEARTBEAT)
5	Heartbeat Acknowledgement (HEARTBEAT ACK)
6	Abort (ABORT)
7	Shutdown (SHUTDOWN)
8	Shutdown Acknowledgement (SHUTDOWN ACK)
9	Operation Error (ERROR)
10	State Cookie (COOKIE ECHO)
11	Cookie Acknowledgement (COOKIE ACK)
12	Reserved for Explicit Congestion Notification Echo (ECNE)
13	Reserved for Congestion Window Reduced (CWR)
14	Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	reserved by IETF
63	IETF-defined Chunk Extensions
64 to 126	reserved by IETF
127	IETF-defined Chunk Extensions
128 to 190	reserved by IETF
191	IETF-defined Chunk Extensions
192 to 254	reserved by IETF
255	IETF-defined Chunk Extensions

Πίνακας 1

Το κεφάλαιο 2 περιγράφει εκτενέστερα το SCTP πρωτόκολλο.

1.6 Σύντομη περιγραφή του πρωτοκόλλου DCCP

Το DCCP είναι πρωτόκολλο που ανήκει στο επίπεδο μεταφοράς (transport layer) του μοντέλου OSI και συνδυάζει την μη-αξιόπιστη(unreliable) μετάδοση του UDP με τον έλεγχο συμφόρησης(congestion control) του TCP. Όπως και το SCTP, και το DCCP είναι *message-oriented* πρωτόκολλο.

1.6.1 Βασικά χαρακτηριστικά DCCP

Το ***DCCP*** ή αλλιώς ***Datagram Congestion Control Protocol (Πρωτόκολλο Ελέγχου Συμφόρησης Αυτοδύναμων πακέτων)*** είναι ένα πρωτόκολλο μεταφοράς για unreliable μεταφορά πακέτων. Ο κύριος στόχος και η διαφορά του από το διαδεδομένο UDP, είναι ότι παρέχει έλεγχο συμφόρηση στις unreliable ροές. Το DCCP είναι χρήσιμο κυρίως για εφαρμογές που έχουν περιορισμό στον χρόνο παράδοσης των δεδομένων, δεν απαιτούν τα δεδομένα να μεταφέρονται αξιόπιστα και ταυτόχρονα απαιτούν έλεγχο συμφόρησης. Τέτοιες εφαρμογές είναι για παράδειγμα διάφορες πολυμεσικές streaming εφαρμογές(audio/video streaming), on-line παιχνίδια όπως επίσης και εφαρμογές IP τηλεφωνίας(VOIP). Τέτοιου είδους εφαρμογές δεν μπορούν να κάνουν χρήση του TCP πρωτοκόλλου μιας και το TCP προδίδει καθυστέρηση στη μεταφορά των δεδομένων λόγω των μηχανισμών που εκτελεί για την αξιοπιστία των δεδομένων. Ταυτόχρονα δεν μπορούν να κάνουν χρήση του UDP γιατί το UDP δεν παρέχει έλεγχο συμφόρησης.

Το DCCP έχει σχεδιαστεί έτσι ώστε να διαχωρίζεται η κύρια λειτουργία του πρωτοκόλλου από τον μηχανισμό ελέγχου συμφόρησης. Έτσι δίνεται η δυνατότητα στο DCCP να χρησιμοποιηθούν διαφορετικοί *μηχανισμοί ελέγχου συμφόρησης*, ανάλογα με τις ανάγκες τις κάθε ροής.

Τα βασικά χαρακτηριστικά του DCCP συνοψίζονται παρακάτω :

- **Επιλογή μηχανισμού ελέγχου συμφόρησης.**
  Ο κύριος στόχος του DCCP είναι το να προσφέρει μηχανισμούς ελέγχου συμφόρησης για ροές δεδομένων. Το DCCP μπορεί να υποστηρίξει διαφορετικούς μηχανισμούς οι οποίοι αναγνωρίζονται από έναν αριθμό *CCID (Congestion Control IDentifier)*. Μέχρις στιγμής δύο μηχανισμοί έχουν οριστεί από τον IETF : [13]

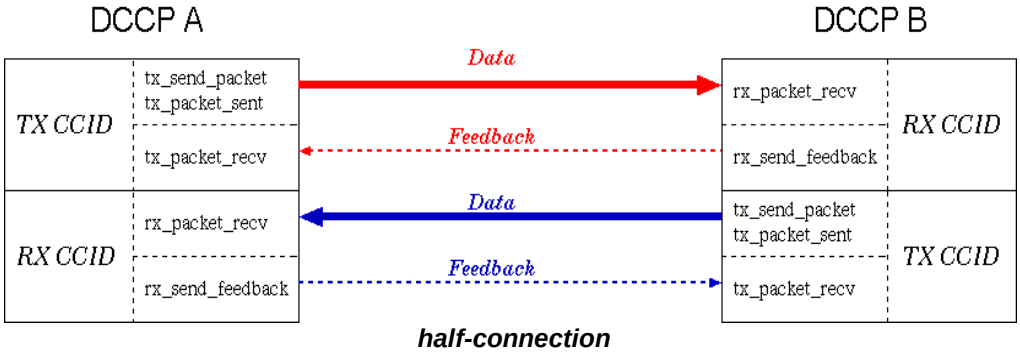
  1. **TCP like congestion control (CCID 2)** : Πρόκειται για έναν TCP friendly μηχανισμό που ρυθμίζει το ρυθμό μετάδοσης με τη χρήση ενός παραθύρου συμφόρησης (congestion window) όπως το TCP. Προτείνεται για εφαρμογές που θέλουν να εκμεταλλευθούν γρήγορα το εύρος ζώνης ενός δικτύου. Δεν είναι κατάλληλο για εφαρμογές VoIP.

2. **TCP Friendly congestion control (CCID 3)** : Πρόκειται ουσιαστικά για την υλοποίηση του *TFRC* μηχανισμού (του TCP) που οδηγεί σε πιο ομαλές αλλαγές του ρυθμού μετάδοσης.

Οι εφαρμογές μπορούν να επιλέξουν τον μηχανισμό που ταιριάζει καλύτερα στα ιδιαίτερα χαρακτηριστικά τους. Τέλος είναι σε εξέλιξη και ένας τρίτος μηχανισμός ελέγχου συμφόρησης , ο TCP-Friendly Rate Control for Small Packets(CCID 4) ο οποίος θα μπορεί να χρησιμοποιηθεί αποτελεσματικά από VoIP εφαρμογές.

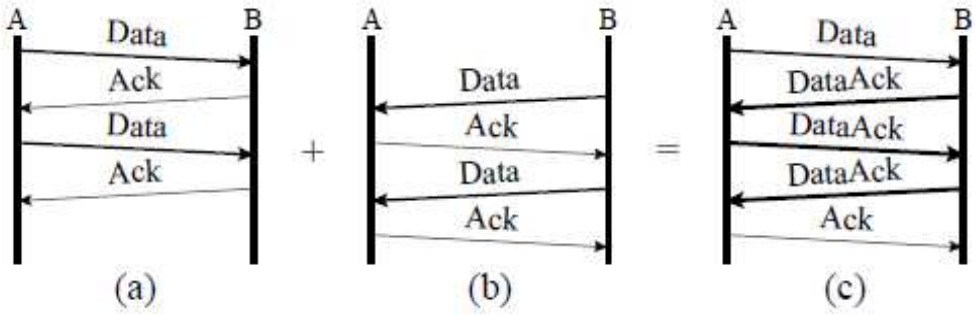
- **Unreliable μεταφορά datagrams με επιβεβαίωση.** Όπως και στο UDP δεν υπάρχει αναμετάδοση των χαμένων πακέτων. Οι επιβεβαιώσεις πακέτων,που είναι αναγκαίες για τους μηχανισμούς ελέγχου συμφόρησης, μπορεί να γίνουν με διαφορετικούς τρόπους όπως με διανύσματα επιβεβαίωσης ή με επιβεβαίωση για κάθε πακέτο.

- **Reliable handshakes κατά την σύνδεση.** Το DCCP είναι ένα connection-oriented πρωτόκολλο, διευκολύνοντας έτσι την επικοινωνία μέσω ενδιάμεσων συσκευών όπως NATs και firewalls.



half-connection
Σχήμα 17

Όπως και το TCP έτσι και το DCCP παρέχει μία σύνδεση διπλής κατεύθυνσης(bidirectional-full duplex) όπου τα δεδομένα και οι επιβεβαιώσεις ρέουν και προς τις δύο κατευθύνσεις.
Ωστόσο πολλές εφαρμογές που χρησιμοποιούν το DCCP , απαιτούν ασύμμετρη *(asymmetric)* επικοινωνία. Το DCCP διαιρεί μια σύνδεση σε δύο λογικές **«μισές συνδέσεις» (*half-connections)***. Η κάθε μία από αυτές αποτελείται από την αποστολή των δεδομένων από το ένα άκρο στο άλλο και από την αποστολή των επιβεβαιώσεων (σχήμα 17). Έτσι όταν η επικοινωνία είναι διπλής κατεύθυνσης, οι δύο αυτές half-connections δουλεύουν ταυτόχρονα (σχήμα 18). Η κάθε μία από τις δύο half-connections είναι ανεξάρτητη από την άλλη και μπορεί να παρέχει διαφορετικό μηχανισμό ελέγχου συμφόρησης από την άλλη.

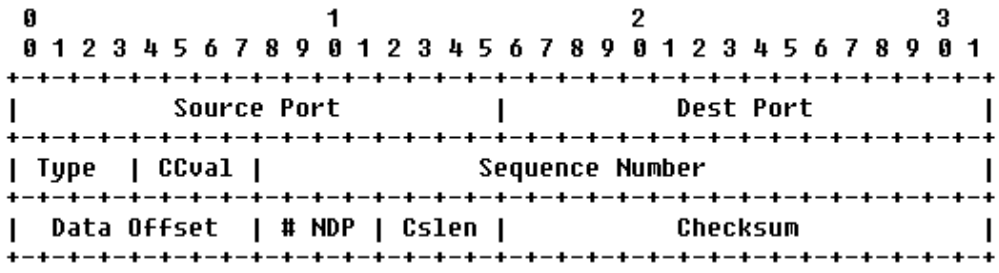


Σχήμα 18

- **Μηχανισμούς ελέγχου συμφόρησης που αξιοποιούν τον ECN μηχανισμό.**
  Προαιρετικούς μηχανισμούς που ενημερώνουν την εφαρμογή με υψηλή αξιοπιστία ποια πακέτα έφτασαν στον δέκτη και αν αυτά ήταν ECN marked, κατεστραμμένα, ή απορρίφθηκαν στον buffer λήψης. Με τον ECN (Explicit Congestion Notification) μηχανισμό τα πακέτα που μεταφέρονται σε ένα δίκτυο όπου υπάρχει συμφόρηση, μαρκάρονται και δεν απορρίπτονται.

- **Δυνατότητα εύρεσης του Path Maximum Transmission Unit (PMTU) μέγιστου μεγέθους του πακέτου μετάδοσης σε δικτυακή διαδρομή).**

Στο Σχήμα 19 βλέπουμε τη μορφή του *DCCP header*.



Σχήμα 19

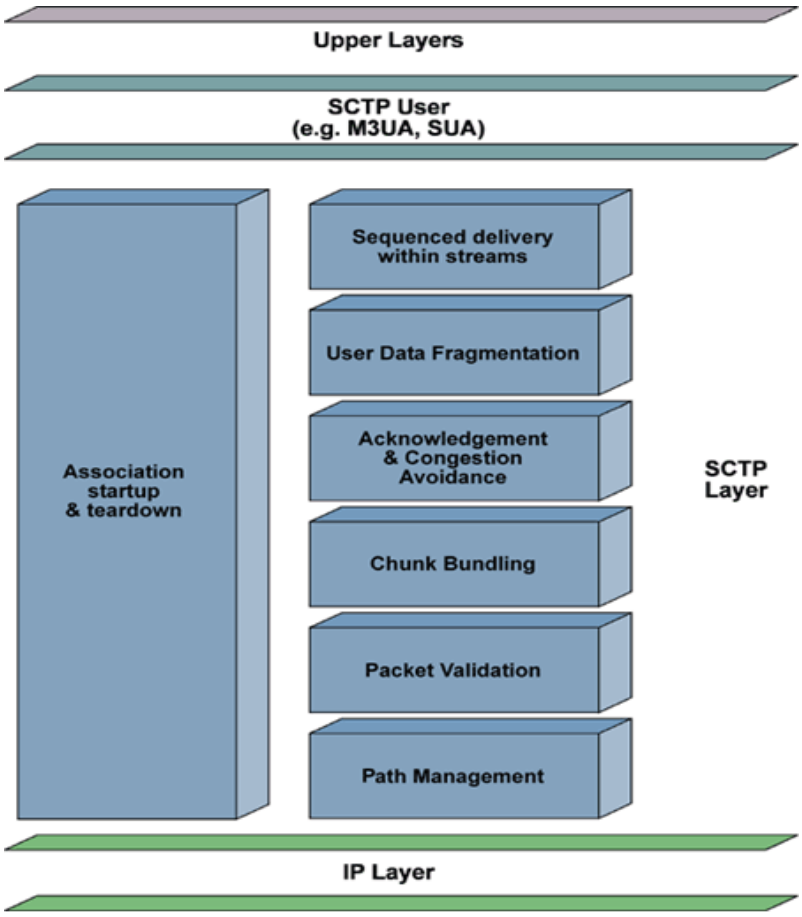
2. SCTP.

2.1 Περιγραφή πρωτοκόλλου.

Όπως αναφέραμε και στην ενότητα 1.5, το SCTP (***Stream Control Transmission Protocol)*** είναι ένα νέο πρωτόκολλο επιπέδου μεταφοράς που σχεδιάστηκε για τη μεταφορά των δεδομένων σε δίκτυα βασιζόμενα στο IP πρωτόκολλο. Η βασική υπηρεσία που προσφέρει το SCTP είναι η αξιόπιστη μεταφορά των δεδομένων μεταξύ δύο κόμβων. Η αξιοπιστία αυτή επιτυγχάνεται με τη διαδικασία της ***συσχέτισης (association),*** στην οποία τα δύο άκρα συνδέονται μεταξύ τους και παραμένουν συνδεδεμένα μέχρι να μεταφερθούν όλα τα δεδομένα ανάμεσά τους. Τα δεδομένα μέσα στη συσχέτιση μεταφέρονται με αξιόπιστα και φτάνουν στον παραλήπτη με τη σωστή σειρά.

Λόγω λοιπών της συσχέτισης αυτής το SCTP θεωρείται ως connection-oriented πρωτόκολλο αλλά εδώ η συσχέτιση είναι πιο γενικευμένη από την σύνδεση του TCP. Όπως είδαμε και στην παράγραφο 1.5, ένα sctp πακέτο περιέχει DATA και CONTROL chunks. Τα CONTROL chunks φαίνονται στον πίνακα 1.

Στο σχήμα 20 βλέπουμε τις λειτουργίες που συνθέτουν το πρωτόκολλο SCTP.



Σχήμα 20

Οι κύριες λειτουργίες του SCTP είναι οι εξής:

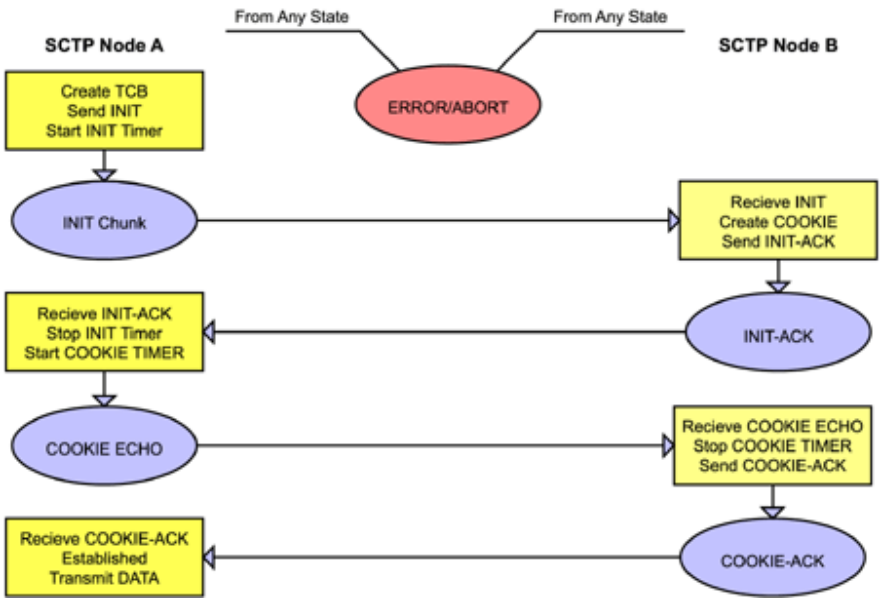
- **Έναρξη, ανταλλαγή δεδομένων και τερματισμός μιας συσχέτισης** (Association startup, Data Transmission and association takedown)
- **Σειρά παράδοσης των δεδομένων μέσα σε "streams"** (Sequenced delivery within streams)
- **Τεμαχισμός (κατακερματισμός) των δεδομένων** (User Data Fragmentation)
- **Επιβεβαιώσεις και έλεγχος συμφόρησης** (Acknowledgement and congestion avoidance)
- **Κατασκευή των chunks** (Chunk Bundling)
- **Ταυτοποίηση πακέτων** (Packet Validation)
- **Διαχείριση μονοπατιού** (Path Management)

2.1.1 Έναρξη, ανταλλαγή δεδομένων και τερματισμός μιας συσχέτισης

Η διαδικασία της **συσχέτισης (association)** αρχίζει όταν ένα άκρο στέλνει μια αίτηση για σύνδεση προς ένα άλλο άκρο. Κατά την διαδικασία της **αρχικοποίησης (initialization)** , ανταλλάσσονται τέσσερα μηνύματα μεταξύ των δύο κόμβων. Η διαδικασία αυτή είναι γνωστή και ως **4-way handshake (Τετραπλή χειραψία).** Κατά την αποστολή του τρίτου και τέταρτου μηνύματος, όταν δηλαδή η σύνδεση (association) έχει ήδη επικυρωθεί, μπορούν να αρχίσουν οι δύο κόμβοι να ανταλλάσσουν δεδομένα. Ένας μηχανισμός cookie έχει προστεθεί μέσα στην ακολουθία μηνυμάτων ώστε να παρέχει προστασία από κάποιας μορφής επίθεσης στην σύνδεση. Για μεγαλύτερη ευκολία, θα ονομάσουμε τον έναν κόμβο στην μια άκρη της σύνδεσης κόμβο Α και τον άλλο στην άλλη άκρη, κόμβο Β (σχήμα 21).

Κατά τη διαδικασία του association, οι δύο κόμβοι μπορούν οποιαδήποτε στιγμή να στείλουν ένα ABORT ή ένα ERROR chunk. Και στις δύο αυτές περιπτώσεις η σύνδεση τερματίζεται επειγόντως.

- **Association Initiation – έναρξη συσχέτισης**



Σχήμα 21

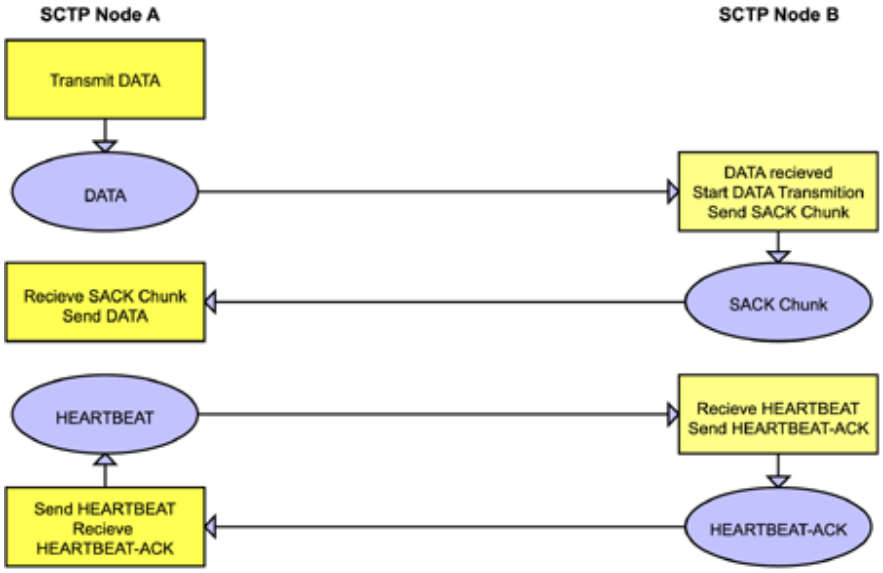
1. Ο κόμβος Α στέλνει ένα INIT chunk στον κόμβο Β και αρχίζει τον INIT timer.
2. Ο κόμβος Β παραλαμβάνει το INIT chunk και στέλνει ένα INIT ACK μαζί με ένα COOKIE στον κόμβο Α
3. Ο κόμβος Α παραλαμβάνει το INIT ACK chunk και σταματάει αμέσως τον INIT timer. Έπειτα στέλνει στον κόμβο Β ένα COOKIE ECHO chunk και αρχίζει τον COOKIE timer. Στο πακέτο αυτό μπορούν να σταλθούν και DATA chunks.
4. Ο κόμβος Β ελέγχει την εγκυρότητα του COOKIE. Αν το βρεί έγκυρο στέλνει στον κόμβο Α ένα COOKIE ACK.
5. Ο κόμβος Α παραλαμβάνει το COOKIE ACK και προχωράει στην επόμενη φάση , της διαδικασία αποστολής δεδομένων.

Ο κύριος σκοπός του **COOKIE** είναι η προστασία της σύνδεσης από τα ***Blind Denial of Service attacks*** *(τυφλών επιθέσεων άρνησης υπηρεσιών)* γνωστές και ως *Flooding Attack*s (πλημμύρες) ή *SYN Attacks*. Στα **Flooding Attacks** , ο επιτιθέμενος/αποστολέας στέλνει πακέτα αρχικοποίησης (INIT για το SCTP ή SYN για το TCP) με σκοπό να σπαταλήσει τη μνήμη του παραλήπτη/ εξυπηρετητή. Ο μηχανισμός αυτός λοιπόν μας προστατεύει από τέτοιου είδους επιθέσεις. Έτσι ο εξυπηρετητής αντί να σπαταλά μνήμη κάθε φορά για ένα μπλοκ ελέγχου μετάβασης, δημιουργεί ένα cookie με τις πληροφορίες του μπλοκ και με μια διάρκεια ζωής (timer) 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 γνωρίζει τις πληροφορίες του cookie) δεν θα στείλει ποτέ το μήνυμα αυτό (το COOKIE ECHO δηλαδή) και ο εξυπηρετητής δεν θα σπαταλήσει επιπλέον μνήμη για να δημιουργήσει ένα νέο μπλοκ. Το TCP πρωτόκολλο σε αντίθεση με το SCTP, είναι ευάλωτο σε τέτοιου είδους επιθέσεις λόγω του προβλήματος των «μισάνοιχτων» (half-open) συνδέσεων (3-way handshake).

- **<u>Association Data Transmission – Μεταφορά Δεδομένων</u>**

Ο κόμβος Α και ο κόμβος Β ξεκινούν την αποστολή δεδομένων (σχήμα 22). Κατά τη διάρκεια της μεταφοράς των δεδομένων, οι κόμβοι Α και Β ανταλλάσσουν **HEARTBEAT** και HEARTBEAT ACK chunks. Ουσιαστικά τα πακέτα αυτά ελέγχουν την συνδεσιμότητα των άκρων. Όπως περιγράψαμε στην παράγραφο 1.5, ένα από τα βασικά χαρακτηριστικά του SCTP είναι η υποστήριξη Multi-Homing στην οποία όπως είπαμε ο κάθε σταθμός εκτός από την κύρια(primary) διεύθυνση μπορεί να έχει και άλλες εναλλακτικές IP διευθύνσεις. Τα πακέτα κτύπου καρδιάς (heartbeat) στέλνονται περιοδικά σε όλες τις μη χρησιμοποιούμενες αυτές διευθύνσεις (εναλλακτικές διευθύνσεις). Ένας μετρητής διατηρεί τον αριθμό των πακέτων HEARTBEAT που έχουν σταλεί

στις διευθύνσεις αυτές χωρίς να έχουμε λάβει μια επιβεβαίωση (HEARTBEAT ACK). Όταν αυτός ο μετρητής ξεπεράσει μια αρχικά καθορισμένη μέγιστη τιμή, οι διευθύνσεις αυτές ορίζονται ως ανενεργές. Τα HEARTBEATS συνεχίζουν να στέλνονται σε ανενεργές διευθύνσεις μέχρι να πάρουμε πίσω ένα HEARTBEAT ACK. Τότε η συγκεκριμένη διεύθυνση ορίζεται και ως ενεργή (active). Έτσι με τα HEARTBEATS εξασφαλίζεται η αξιοπιστία αποστολής των δεδομένων αφού σε περίπτωση βλάβης της σύνδεσης στην κύρια διεύθυνση, τα δεδομένα θα συνεχίσουν να στέλνονται στον προορισμό από τις εναλλακτικές διευθύνσεις.



Σχήμα 22

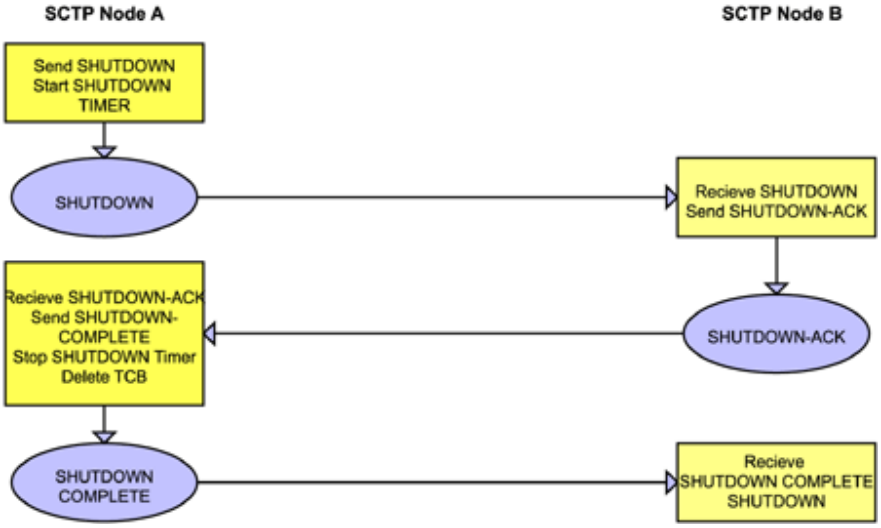
Η μεταφορά λοιπόν των δεδομένων γίνεται με την παρακάτω διαδικασία:

1. Ο κόμβος Α στέλνει DATA chunks στον κόμβο Β
2. Για κάθε DATA chunk που λαμβάνει ο κόμβος Β στέλνει στον Α μια επιβεβαίωση (SACK chunk)
3. Η διαδικασία αυτή επαναλαμβάνεται και για τους δύο κόμβους μέχρι κάποιος από αυτούς να στείλει ένα SHUTDOWN chunk (μέσα σε κάποιο πακέτο) ώστε να ζητήσει τερματισμό της σύνδεσης.

Όπως βλέπουμε, ταυτόχρονα με την ανταλλαγή δεδομένων, οι δύο κόμβοι ανταλλάσσουν και HEARTBEATS.

- **Association Shutdown – Τερματισμός συσχέτισης**

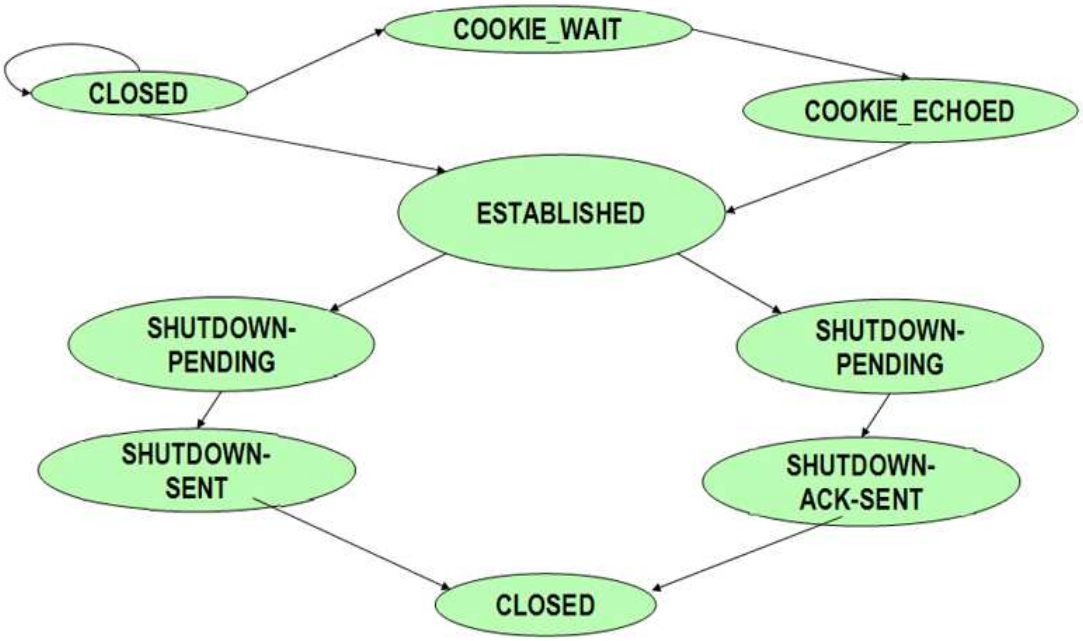
Η διαδικασία του τερματισμού (σχήμα 23) μιας συσχέτισης αρχίζει όταν κάποιος από τους δύο κόμβους στείλει ένα SHUTDOWN chunk στον άλλο κόμβο.



Σχήμα 23

1. Ο κόμβος Α στέλνει ένα SHUTDOWN chunk στον κόμβο Β και αρχίζει τον shutdown timer
2. Ο κόμβος Β παραλαμβάνει το SHUTDOWN chunk και δημιουργεί ένα SHUTDOWN ACK chunk(επιβεβαίωση) το οποίο και στέλνει στον κόμβο Α
3. Ο κόμβος Α παραλαμβάνει το SHUTDOWN ACK και ανταποκρίνεται σταματώντας τον SHUTDOWN timer. Έπειτα δημιουργεί ένα SHUTDOWN COMPLETE chunk το οποίο και στέλνει στον κόμβο Β. Με το chunk αυτό σηματοδοτείται και ο τερματισμός μιας συσχέτισης.

Κατά τη διαδικασία του association λοιπόν, παρατηρούμε ότι οι sctp σταθμοί περνούν από διάφορα στάδια-καταστάσεις. Στο σχήμα 24 βλέπουμε το **διάγραμμα κατάστασης (state diagram)** των σταθμών σε μια sctp σύνδεση.



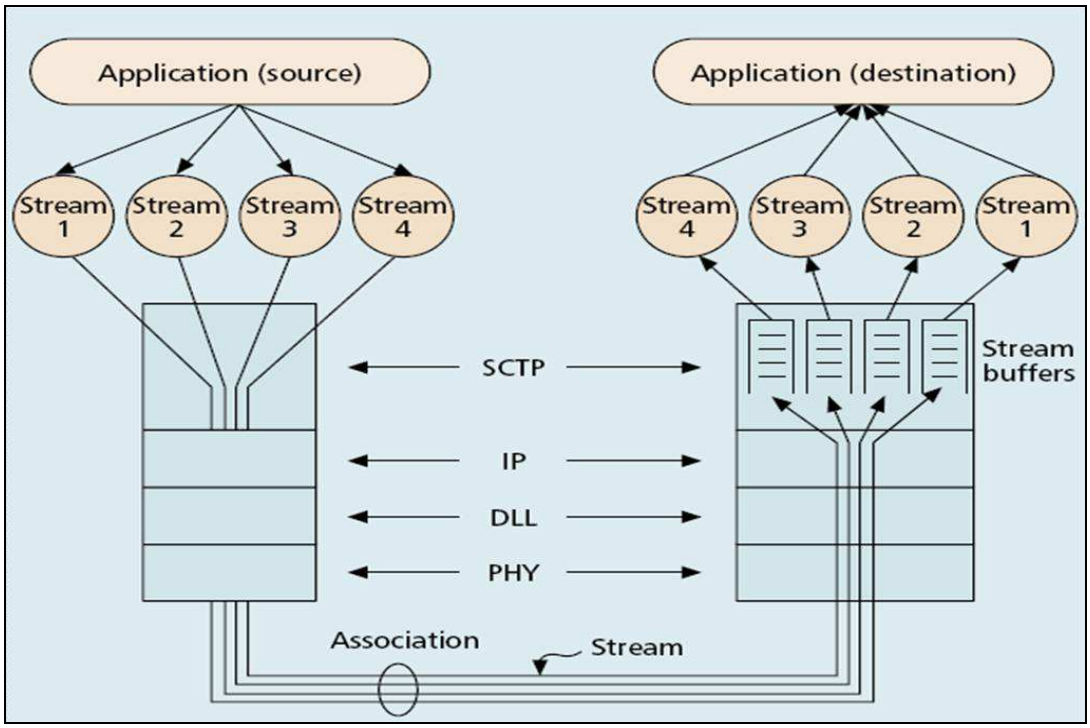
Σχήμα 24

2.1.2 Σειρά παράδοσης των δεδομένων μέσα σε “streams”

Στην παράγραφο 1,5 μιλήσαμε για την λειτουργία **Multi-Streaming** του πρωτοκόλλου SCTP με την οποία τα δεδομένα τεμαχίζονται σε πολλά μικρότερα κομμάτια και τοποθετούνται σε πολλά **streams**(ρεύματα ή ροές)(σχήμα 25). Ο όρος “stream” εδώ έχει να κάνει με την ακολουθία των μηνυμάτων που μεταφέρονται στο δίκτυο(από το αμέσως επόμενου επιπέδου πρωτόκολλο). Τα μηνύματα αυτά θα φτάσουν στον παραλήπτη σε σωστή σειρά. Έτσι λοιπόν στο SCTP μπορούμε να μιλάμε για ακολουθία μηνυμάτων σε αντίθεση με το TCP πρωτόκολλο το οποίο αναφέρεται σε ακολουθία από bytes(byte-oriented). Γι’αυτό και το SCTP χαρακτηρίζεται ως **stream-oriented** πρωτόκολλο.

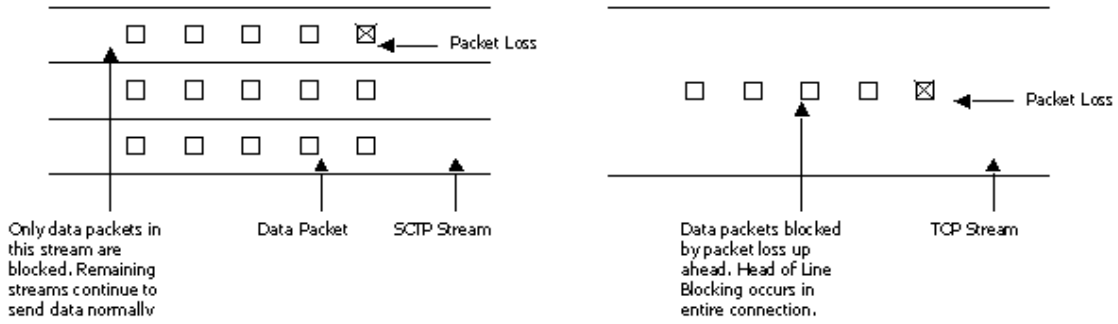
Σχήμα 25

Κατά την έναρξη μιας συσχέτισης(association) , το ένα άκρο ενημερώνει το απέναντι άκρο για τον μέγιστο αριθμό από streams που μπορεί να υποστηρίξει. Κάθε μήνυμα που μεταφέρεται μέσα στην συσχέτιση έχει το δικό του ***αριθμό ακολουθίας (Stream Sequence Number, SSN)***. Έτσι ο αποστολέας του μηνύματος ξέρει πότε το μήνυμα παραδίδεται σε σωστή σειρά και σε ποιό stream ανήκει. Όπως είπαμε τα streams είναι ανεξάρτητα μεταξύ τους. Αν για παράδειγμα χαθεί κάποιο μήνυμα που βρίσκεται σε ένα stream, θα επηρεάσει μόνο την παράδοση μέσα στο συγκεκριμένο stream. Τα υπόλοιπα streams δεν επηρεάζονται από αυτό και συνεχίζουν τη ροή των μηνυμάτων (σχήμα 26).



Σχήμα 26

Σε αντίθεση με το SCTP, το TCP θεωρεί ότι υπάρχει μόνο ένα stream δεδομένων όπου τα δεδομένα φτάνουν στον παραλήπτη σε σειρά σε σχέση με την σειρά αποστολής των bytes μέσα στο stream. Αυτό προκαλεί επιπρόσθετη καθυστέρηση όταν χάνονται δεδομένα αφού διακόπτεται όλη η ροή των δεδομένων για να στείλει ο αποστολέας ξανά το μήνυμα που χάθηκε(ή να στείλει ένα πακέτο εκτός σειράς). Επομένως καταλαβαίνουμε πόσο σημαντική είναι η λειτουργία Multi-Streaming του SCTP πρωτοκόλλου αφού κάθε stream είναι ανεξάρτητο από τα άλλα και έτσι επιτυγχάνεται η καλύτερη απόδοση της μεταφοράς των δεδομένων (σχήμα 27).



Σχήμα 27

Τα πλεονεκτήματα λοιπόν που παρέχονται από τα πολλαπλά αυτά streams, είναι η ομαδοποίηση δεδομένων και ο έλεγχος της πληροφορίας μέσα στο ίδιο πακέτο καθώς και η ευελιξία στην σειρά παράδοσης των πακέτων (*ordered delivery*). Με την σειρά παράδοσης μπορούμε να αποφύγουμε το **head-of-line** (HOL) blocking το οποίο εμφανίζεται στο TCP. Ωστόσο Το SCTP παρέχει και έναν μηχανισμό ο οποίος παρακάμπτει την υπηρεσία της σωστής σειράς παράδοσης. Έτσι αν χρησιμοποιηθεί η σημαία για την παράδοση εκτός σειράς(*unordered delivery*), 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). Τέλος δίνεται και η δυνατότητα να έχουμε μερική-σειρά(*partial-ordered delivery*) στην παράδοση των μηνυμάτων. Αυτό είναι χρήσιμο κυρίως σε εφαρμογές πολυμέσων.

Η σειρά της παράδοσης των παραλαμβανόμενων δεδομένων καθορίζεται από δύο σύνολα αριθμών. Το Stream Sequence Number που περιγράψαμε πριν και το Stream ID. Κάθε DATA chunk (το πράσινο τμήμα στο σχήμα 16) έχει το δικό του έγκυρο ***stream identifier (Stream ID)***. Άν ένας σταθμός παραλάβει ένα DATA chunk με μη έγκυρο stream identifier τότε απορρίπτει το DATA chunk και στέλνει αμέσως στον αποστολέα ένα ERROR chunk και μαρκάρει το πεδίο του "Invalid Stream Identifier" του chunk αυτού ώστε ο αποστολέας όταν παραλάβει το ERROR chunk να καταλάβει γιατί απέρριψε το πακέτο ο παραλήπτης.

Στο TCP τα segments χρησιμοποιούν το sequence number με το οποίο εξασφαλίζεται η σωστή ακολουθία των δεδομένων που αποστέλλονται.
Στο SCTP , χρησιμοποιείται ένα παρόμοιο σύνολο αριθμών για τα DATA chunks. Ο αριθμός αυτός ονομάζεται ***Transmission Sequence Number*** **(TSN)**.
Εδώ καταλαβαίνουμε τον διαχωρισμό που κάνει η λειτουργία Multi-Streaming, όπου σε κάθε stream διαχωρίζεται η ακολουθία των δεδομένων από την μεταφορά. Συγκεκριμένα το *Stream Sequence Number* όπως είπαμε χρησιμοποιείται για την σειρά των μηνυμάτων μέσα στο stream, ενώ το *Transmission Sequence Number* χρησιμοποιείται για την μεταφορά/επανεκπομπή των μηνυμάτων. Έτσι το TSN χρησιμοποιείται για το acknowledgement των DATA chunks. Τα control chunks δεν χρειάζονται κάποιον μηχανισμό επιβεβαίωσης για τον λόγο ότι επιβεβαιώνονται με άλλα control chunks (όπως για παράδειγμα το INIT το οποίο επιβεβαιώνεται από το INIT-ACK chunk).

Τα Stream Sequence Number σε όλα τα streams πρέπει να αρχίζουν με την τιμή 0 όταν αρχικοποιείται μια συσχέτιση και φτάνουν μέχρι την τιμή 65535. Όταν ένα SSN φτάσει την τιμή αυτή τότε η επόμενη τιμή θα αρχίζει πάλι από το 0.

2.1.3 Τεμαχισμός (κατακερματισμός) των δεδομένων

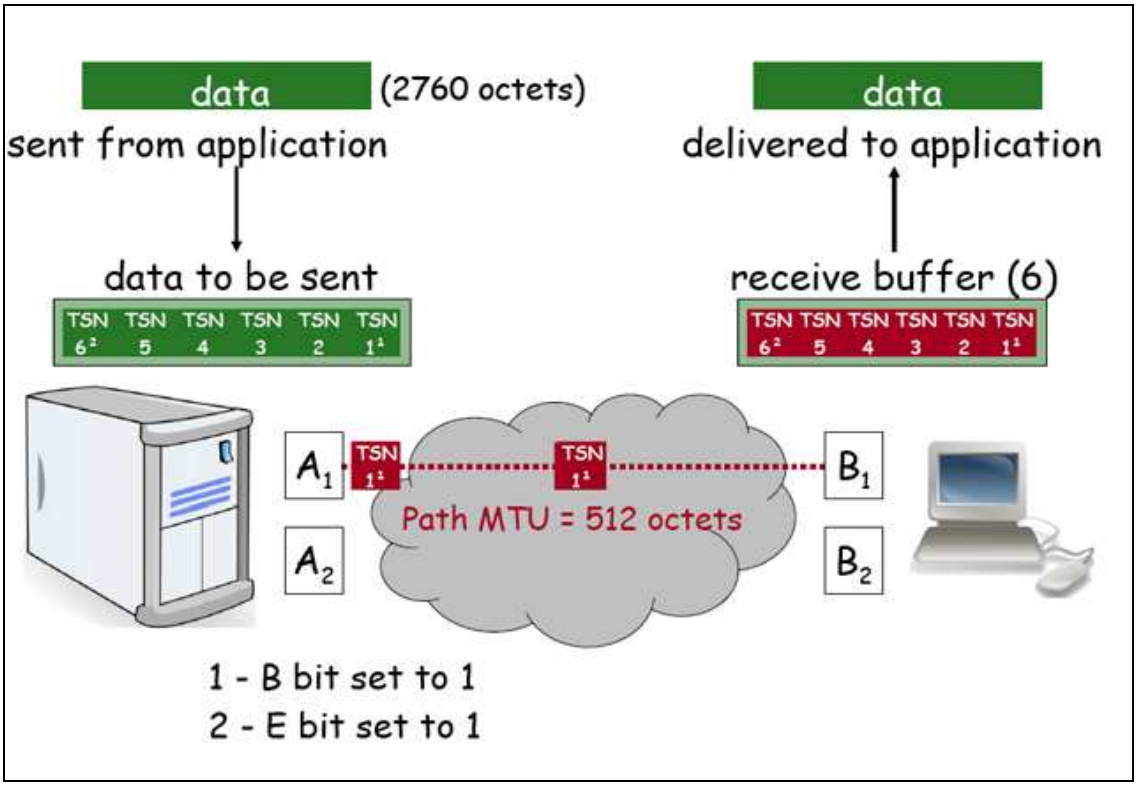
Το SCTP όποτε χρειάζεται, τεμαχίζει τα μηνύματα σε μικρότερα ώστε να μπορέσουν αυτά να σταλούν στον προορισμό τους. Συγκεκριμένα, όταν κάποιο πακέτο περνάει από κάποιο δίκτυο, το οποίο έχει κάποιο όριο στην MTU, και το μήνυμα αυτό είναι μεγαλύτερο από το όριο αυτό, τότε το SCTP θα τεμαχίσει (fragments) το πακέτο σε άλλα μικρότερα τα οποία θα στείλει στον προορισμό. Η επανένωση (**reassembled**) των τεμαχισμένων αυτών πακέτων στο αρχικό μήνυμα γίνεται από τον παραλήπτη. Σε περίπτωση που υπάρχει τεμαχισμός δεδομένων, αυτή γνωστοποιείται στους δύο σταθμούς με τα E και B bits των DATA chunk.

Για να λειτουργήσει σωστά η διαδικασία του **τεμαχισμού δεδομένων**, θα πρέπει τόσο ο αποστολέας όσο και ο παραλήπτης να υποστηρίζουν τη λειτουργία αυτή (**fragmentation**). Αν κατά την αποστολή δεδομένων κριθεί αναγκαίος ο τεμαχισμός κάποιου μηνύματος (ή κάποιων μηνυμάτων) , τότε γίνεται έλεγχος για το αν οι δύο σταθμοί υποστηρίζουν τεμαχισμό δεδομένων. Αν υποστηρίζουν, η διαδικασία προχωρά κανονικά, αν δεν υποστηρίζει έστω και ένας από τους δύο, τότε αυτός που δεν υποστηρίζει στέλνει ένα ERROR chunk

και η αποστολή των δεδομένων διακόπτεται (ενημερώνοντας την εφαρμογή για το πρόβλημα).

Στην παράγραφο 1,5 μιλήσαμε για την λειτουργία Multi-Homing του SCTP. Multi-Homing έχουμε όταν τα άκρα έχουν πολλές IP διευθύνσεις, μία κύρια διεύθυνση και άλλες εναλλακτικές. Αν λοιπόν σε μια σύνδεση, ο κάθε κόμβος είναι multi-homed , και στην αποστολή δεδομένων κριθεί αναγκαία η λειτουργία του τεμαχισμού των δεδομένων, ο τεμαχισμός θα γίνει με βάση το Association Path MTU. Το **Association Path MTU** είναι το μονοπάτι με την μικρότερη MTU. Κι εφόσον μιλάμε για multi-homed δίκτυο, το Association Path MTU θα είναι το μικρότερο MTU από όλες τις διευθύνσεις του παραλήπτη του μηνύματος.

Για το πότε θα ακολουθήσει η διαδικασία του τεμαχισμού των δεδομένων, το SCTP συμβουλεύεται το μέγεθος του SCTP header όπως επίσης και το μέγεθος των DATA chunks. Όταν ένα μήνυμα έχει ήδη τεμαχιστεί, δεν μπορεί να ξανά-τεμαχιστεί. Αν κάτι τέτοιο κριθεί αναγκαίο τότε ο κατακερματισμός αφήνεται για το αμέσως επόμενο επίπεδο (όπως για παράδειγμα ο κατακερματισμός στο IP πρωτόκολλο όπου πλέον μιλάμε για κατακερματισμό IP πακέτων).



Σχήμα 28

Στο σχήμα 28 φαίνεται η διαδικασία του κατακερματισμού.

1. Ο αποστολέας πρέπει να «σπάσει» το κύριο μήνυμα σε σειρές από DATA chunks με τέτοιο τρόπο, ώστε το κάθε chunk μαζί με την το SCTP header να χωράει σε ένα IP Datagram μικρότερο ή ίσο με το association Path MTU(εδώ είναι ίσο με 512 bytes).
2. Ο αποστολέας αναθέτει σε κάθε ένα από αυτά τα DATA chunks έναν TSN αριθμό ώστε να εξασφαλίσει την ακολουθία των chunks. Σε όλα αυτά τα chunks ο αποστολέας αναθέτει τον ίδιο αριθμό SSN.
3. Ο αποστολέας τέλος, χρησιμοποιεί τα bits E και B για να ενημερώσει τον παραλήπτη ότι τα δεδομένα που ακολουθούν είναι «κομματιασμένα». Έτσι θέτει τα B/E bits του πρώτου DATA chunk με την ακολουθία ‘10’, τα B/E bits του τελευταίου DATA chunk με την ακολουθία ‘01’ και τα B/E bits όλων των άλλων ενδιάμεσων DATA chunk με την ακολουθία ‘00’.

Ο παραλήπτης εξετάζοντας την ακολουθία των B/E bits του κάθε DATA chunk καταλαβαίνει ποια δεδομένα προέρχονται από κατακερματισμό και έτσι τα επανασυνθέτει. Αφού επανασυνθέσει λοιπόν τα chunks και σχηματίσει το αρχικό μήνυμα, στη συνέχεια ελέγχει το μήνυμα και καταλαβαίνει σε ποιό stream ανήκει όπως επίσης και την ακολουθία του μέσα στο stream.

2.1.4 Επιβεβαιώσεις και έλεγχος συμφόρησης

Στην παράγραφο 2.1.2 έγινε αναφορά στο ***Transmission Sequence Number*** . Το SCTP αναθέτει σε κάθε DATA chunk ένα TSN το οποίο χρησιμοποιείται από τις **επιβεβαιώσεις** των chunk στο άλλο άκρο. Επίσης το SCTP χρησιμοποιεί ακόμα ένα πεδίο στα DATA chunks. Το πεδίο αυτό ονομάζεται Cumulative TSN Ack. Το ***Cumulative TSN Ack*** είναι το TSN του τελευταίου DATA chunk που επιβεβαιώθηκε. Το πεδίο αυτό είναι χρήσιμο για τον μηχανισμό ελέγχου συμφόρησης.

Με τις επιβεβαιώσεις λοιπόν επιτυγχάνεται η αξιοπιστία κατά τη μεταφορά των δεδομένων.

Οι επιβεβαιώσεις και ο έλεγχος συμφόρησης είναι οι δύο κύριες λειτουργίες με τις οποίες επιτυγχάνεται η επανεκπομπή των δεδομένων των οποίων οι επιβεβαιώσεις δεν έχουν φτάσει στον αποστολέα. Η επανεκπομπή των δεδομένων επιτυγχάνεται με την λειτουργία του ελέγχου συμφόρησης (congestion avoidance). Η λειτουργία αυτή είναι παρόμοια με τον έλεγχο συμφόρησης του TCP πρωτοκόλλου.

Σε περιπτώσεις στις οποίες υπάρχει συμφόρηση των δεδομένων σε ένα δίκτυο, το SCTP εξασφαλίζει τη γρήγορη παράδοση των χρονικά κρίσιμων δεδομένων. Με τον έλεγχο συμφόρησης ουσιαστικά εξασφαλίζεται το γεγονός ότι το SCTP θα λειτουργεί και υπό δυσμενείς συνθήκες λειτουργίας. Όταν συμβεί συμφόρηση δεδομένων σε ένα SCTP δίκτυο, το SCTP πρέπει να ακολουθεί τα σωστά *βήματα ελέγχου συμφόρησης*, εκτελώντας κάποιους συγκεκριμένους ***αλγόριθμους ελέγχου συμφόρησης***, για να επαναφέρει το δίκτυο στην σωστή του λειτουργία, όσο πιο σύντομα γίνεται. Οι αλγόριθμοι ελέγχου συμφόρησης που χρησιμοποιούνται από SCTP είναι βασισμένοι στο [RFC2581 - TCP Congestion Control] και είναι οι:

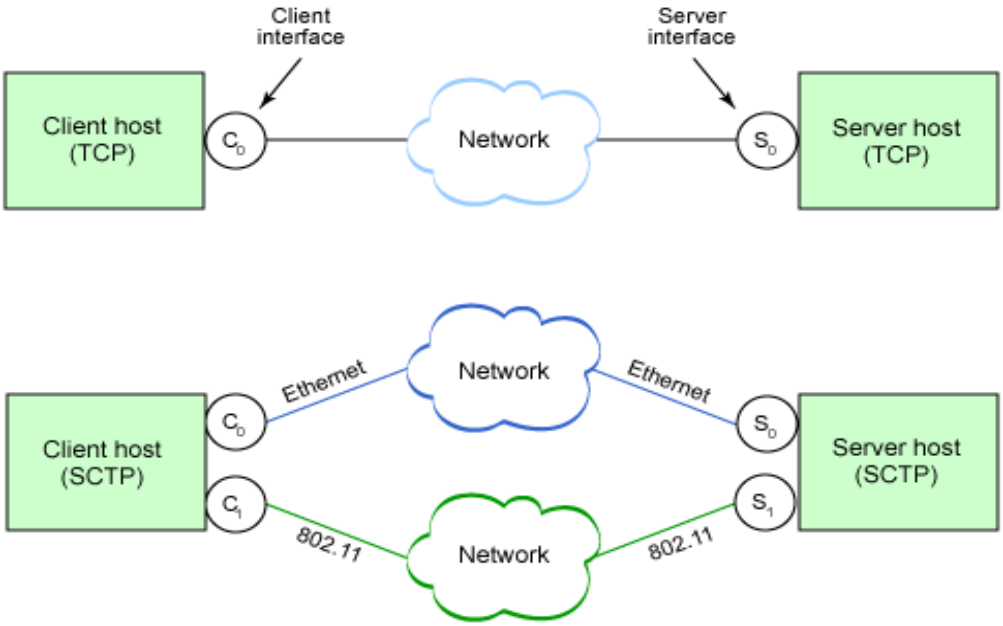
- **Αργή Έναρξη και Αποφυγή Συμφόρησης**(Slow Start and Congestion Control)
- **Γρήγορη Αναμετάδοση και Γρήγορη Αποκατάσταση** (Fast Retransmit and Fast Recovery).

Ο έλεγχος συμφόρησης στο SCTP λαμβάνει χώρα σε ολόκληρο το *association* και όχι σε κάθε *stream* ξεχωριστά.

2.1.4.1 Διαφορές του ελέγχου συμφόρησης του SCTP από το TCP

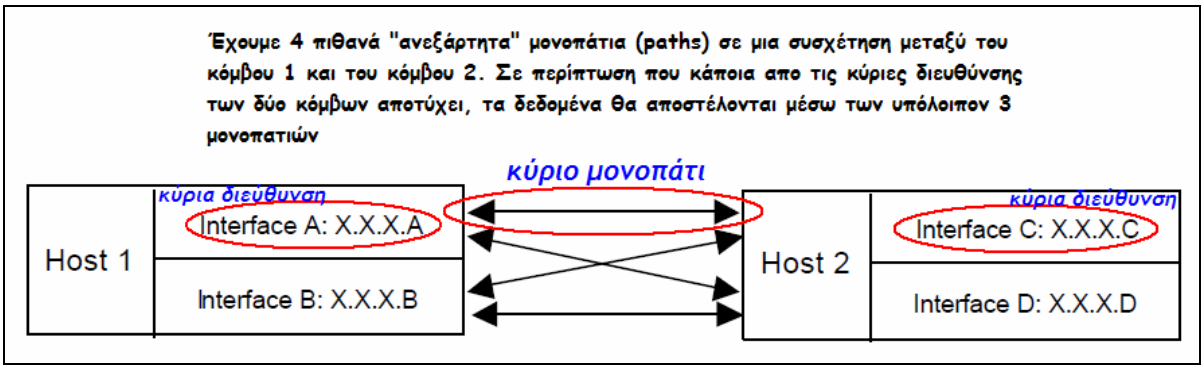
Τα *Gap Ack Blocks* στο SCTP φέρνουν την ίδια σημασιολογική έννοια όπως τα SACK του TCP. Στο SCTP, τα DATA chunks που έχει αναγνωριστεί από τα SACK, συμπεριλαμβανομένων των δεδομένων που έφθασαν στον παραλήπτη εκτός σειράς, θεωρούνται ότι έχουν παραδοθεί πλήρώς μόνο όταν το TSN του τελευταίου DATA chunk είναι ίσο με το Cumulative TSN. Στο TCP το παράθυρο συμφόρησης ή αλλιώς *Congestion window (cwnd)* είναι ουσιαστικά το ανώτερο όριο μεταξύ του μεγαλύτερου αναγνωρισμένου αριθμού ακολουθίας και το πιο πρόσφατο τμήμα δεδομένων που μπορεί να σταλεί μέσα στο παράθυρο συμφόρησης. Στο SCTP το cwmd απλά ελέγχει τον αριθμό των δεδομένων που δεν έχουν αναγνωριστεί. Τα SACK του SCTP οδηγούν σε διαφορετικές υλοποιήσεις *γρήγορης αναμετάδοσης(fast* Retransmit) και *γρήγορης αποκατάστασης (fast recovery)* από το TCP (χωρίς SACK).

Οι μεγαλύτερες διαφορές ανάμεσα στο TCP και στο SCTP ωστόσο, είναι στην λειτουργία του **Multi-Homing**. Στο SCTP , οι 2 κόμβοι σε μια επικοινωνία ενδεχομένως να έχουν πολλές εναλλακτικές IP (σχήμα 29).



Σχήμα 29

Οι διαφορετικές αυτές διευθύνσεις μπορούν οδηγήσουν σε διαφορετικές πορείες στο δίκτυο μεταξύ των δύο κόμβων (σχήμα 30).



Σχήμα 30

Κατά συνέπεια χρειάζεται ένα ξεχωριστό σύνολο παραμέτρων ελέγχου συμφόρησης για κάθε μια από τις διαδρομές. Οι αλγόριθμοι ελέγχου συμφόρησης για mutli-homing κάνουν τις ακόλουθες υποθέσεις:

- Ο αποστολέας χρησιμοποιεί συνήθως την ίδια διεύθυνση προορισμού (primary address) μέχρι να υπάρξει άλλη εντολή για αλλαγή διεύθυνσης σε έναν εναλλακτικό προορισμό (alternative address) σε περίπτωση που μια κύρια διεύθυνση χαρακτηριστεί ανενεργή. Επίσης, το SCTP μπορεί να αναμεταδώσει σε διαφορετική διεύθυνση μεταφοράς από την αρχική.

- Ο αποστολέας κρατά μια ξεχωριστή παράμετρο ελέγχου συμφόρησης για κάθε έναν 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 (slow start)*** κατά την πρώτη μετάδοση σε εκείνη την διεύθυνση.

Το TCP εγγυάται την παράδοση των δεδομένων στο πρωτόκολλο ανώτερου-στρώματος με τη σωστή σειρά. Αυτό σημαίνει ότι όταν υπάρχει διακοπή συνέχειας στον αριθμό ακολουθίας, τότε το TCP περιμένει(διακόπτει την μεταφορά δεδομένων) μέχρι να επανέλθει η σωστή σειρά ακολουθίας. Αντίθετα, όπως έχουμε πεί, στο SCTP δεν συμβαίνει αυτό για το λόγο ότι τα δεδομένα μεταφέρονται σε streams. Άρα, σε μια τέτοια περίπτωση το SCTP θα διακόψει τη μεταφορά των δεδομένων μόνο στο συγκεκριμένο stream στο οποίο ανιχνεύτηκε η λανθασμένη σειρά ακολουθίας των δεδομένων(TSN).

2.1.4.2 Αργή έναρξη και αποφυγή συμφόρησης στο SCTP

Οι αλγόριθμοι της αργής έναρξης (slow start) και της αποφυγής συμφόρησης (congestion avoidance) χρησιμοποιούνται από τους SCTP κόμβους για τον έλεγχο του αριθμού των δεδομένων που εισέρχονται στο δίκτυο. Όπως συμβαίνει και με το TCP, έτσι και στο SCTP τα δύο άκρα χρησιμοποιούν τις παρακάτω τρείς μεταβλητές ελέγχου για να καθορίσουν τον ρυθμό μετάδοσης των δεδομένων:

- Το *receiver advertised window size (**rwnd**)* ή αλλιώς *διαφημισμένο μέγεθος παραθύρου δέκτη*, το οποίο ορίζεται από τον παραλήπτη με βάση τον χώρο (buffer) που έχει(ο παραλήπτης) για τα εισερχόμενα πακέτα. Η τιμή του rwnd ορίζεται για ολόκληρο το association.
- Το *congestion control window (**cwnd**)* ή αλλιώς *παράθυρο συμφόρησης,* το οποίο ορίζεται από τον αποστολέα βασισμένο στην κίνηση του δικτύου.
- Το *slow-start threshold (**ssthresh**)* ή αλλιώς *κατώφλι αργής έναρξης*, το οποίο χρησιμοποιείται από τον αποστολέα για να διακρίνει τις φάσεις αργής αρχής και αποφυγής συμφόρησης.

Ακόμα, το SCTP χρησιμοποιεί και την μεταβλητή ελέγχου *partial_bytes_acked* (**pba**) η οποία χρησιμοποιείται κατά τη διάρκεια της φάσης της αποφυγής συμφόρησης για να διευκολύνει τον αποστολέα στη ρύθμιση του cwnd.

Ο SCTP αποστολέας λοιπόν, κρατά ένα σύνολο των μεταβλητών cwnd, ssthresh για κάθε διεύθυνση προορισμού του δέκτη του (όταν είναι multi-homed). Μόνο ένα rwnd κρατείται για ολόκληρη τη συσχέτιση.

- Σε μια SCTP σύνδεση λοιπόν, ο αποστολέας χρησιμοποιεί τον αλγόριθμο **αργής έναρξης** αν η τιμή του cwnd είναι *μικρότερη* από την τιμή του ssthresh(**cwnd < ssthresh**). Εδώ η αρχική τιμή του cwnd πρέπει να είναι <=2*MTU πρίν αρχίσει η μετάδοση δεδομένων. Σε κάθε επιτυχημένη μεταφορά δεδομένων (όταν τα δεδομένα επιβεβαιώνονται δηλαδή) η τιμή του cwnd αυξάνεται μέχρι να γίνει ίση με την τιμή του ssthresh. Το cwnd αυξάνεται μόνο στις εξής περιπτώσεις: α) η επιβεβαίωση περιλαμβάνει το *cumulative ack point (cumack)* και β) ολόκληρο το cwnd χρησιμοποιούνταν πρίν φτάσει η επιβεβαίωση. Τότε λοιπόν η τιμή του cwnd αυξάνεται κατά το ελάχιστο της πιο πρόσφατης επιβεβαίωσης και της MTU της διεύθυνσης από την οποία ο παραλήπτης έλαβε την επιβεβαίωση.
- Άν η τιμή του cwnd είναι *μεγαλύτερη* από την τιμή του ssthresh(**cwnd > ssthresh**), τότε ο αποστολέας χρησιμοποιεί τον αλγόριθμο **αποφυγής συμφόρησης**. Στόχος του αλγορίθμου είναι η αύξηση του cwnd κατά 1*MTU ανα RTT (Round-Trip Time). Εδώ χρησιμοποιείται η μεταβλητή *partial_bytes_acked* (**pba**) η οποία αρχικοποιείται σε 0. Έπειτα, όταν ο αποστολέας παραλαμβάνει ένα SACK που σηματοδοτεί το *cumulative ack point (cumack)* η τιμή του pba αυξάνεται κατά τον αριθμό των bytes που επιβεβαιώθηκαν από το SACK. Έπειτα το cwnd αυξάνεται κατά 1*MTU μόνο στις εξής περιπτώσεις: α) pba <= cwnd β) ολόκληρο το cwnd χρησιμοποιούνταν πρίν φτάσει η επιβεβαίωση. Με την αύξηση του cwnd το pba παίρνει την τιμή [pda-cwnd]. Όταν όλα τα δεδομένα του αποστολέα επιβεβαιώνονται , τότε το pba γίνεται 0.
- Άν η τιμή του cwnd είναι *ίση* με την τιμή του ssthresh, τότε ο αποστολέας μπορεί να χρησιμοποιήσει **είτε τον αλγόριθμο της αργής έναρξής, είτε τον αλγόριθμο της αποφυγής συμφόρησης**.

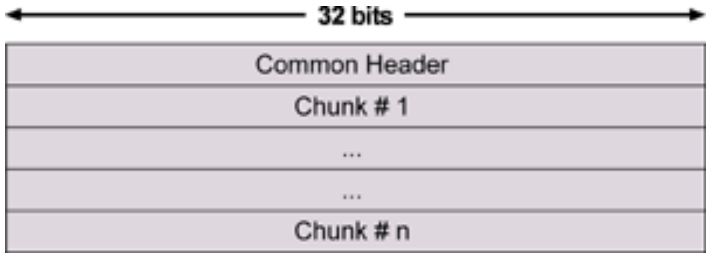
Σε όλες τις περιπτώσεις, είτε όταν εκτελείται ο αλγόριθμος αργής έναρξης είτε αυτός της αποφυγής συμφόρησης, όταν ένας κόμβος δεν διαβιβάζει δεδομένα σε μια δεδομένη διεύθυνση μεταφοράς το cwnd της διεύθυνσης μεταφοράς πρέπει να προσαρμοστεί σε: *max(cwnd/2, 2*MTU) ανά RTO* (Retransmission Timeout).

2.1.4.3 Γρήγορη Αναμετάδοση και Γρήγορη Αποκατάσταση στο SCTP

Ο αλγόριθμος της ***Γρήγορης Αναμετάδοσης*** χρησιμοποιείται για την έξυπνη αναμετάδοση των χαμένων τμημάτων της πληροφορίας σε μια συσχέτιση. Όταν ο παραλήπτης παραλαμβάνει ένα DATA chunk εκτός σειράς, τότε στέλνει στον αποστολέα ένα SACK με το TSN του DATA chunk αυτού. Ο αλγόριθμος Γρήγορης Αναμετάδοσης χρησιμοποιεί τέσσερα πακέτα SACK για να διαπιστώσει ποια δεδομένα χάθηκαν και αναμεταδίδει τα DATA chunk που χάθηκαν χωρίς να περιμένει τη λήξη του χρονόμετρου αναμετάδοσης (retransmission timer). Στη συνέχεια ο αλγόριθμος ***Γρήγορης Αποκατάστασης*** ελέγχει την μετάδοση των δεδομένων αυτών μέχρι να βεβαιωθεί ότι όλα τα χαμένα δεδομένα αναμεταδόθηκαν στον παραλήπτη.

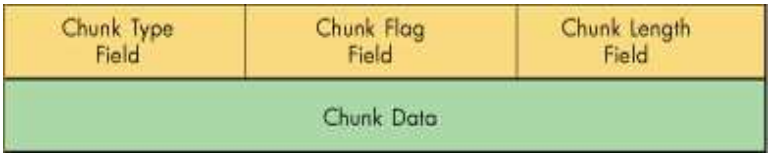
2.1.5 Κατασκευή των Chunk

Όπως αναφέραμε και στην παράγραφο 1,5 ένα πακέτο SCTP αποτελείται από την sctp επικεφαλίδα και από ένα ή περισσότερα «τεμάχια» (chunks) (σχήμα 31).



Σχήμα 31

Κάθε chunk έχει την παρακάτω μορφή (σχήμα 32):



Σχήμα 32

Τα chunks μπορεί να περιέχουν δεδομένα οπότε ονομάζονται DATA chunks ή να περιέχουν πληροφορίες ελέγχου οπότε ονομάζονται CONTROL chunks. Έτσι λοιπόν ένα SCTP πακέτο μπορεί να έχει ένα ή περισσότερα chunks. Η λειτουργία της κατασκευής των chunk είναι υπεύθυνη για την συναρμολόγηση ολόκληρου του SCTP πακέτου (από τον αποστολέα), καθώς και για την αποσυναρμολόγησή του (από τον δέκτη).

Στην παράγραφο 1,5 και στον πίνακα 1 φαίνονται οι 13 τύποι των CONTROL chunk. Ο τύπος των chunk όπως είπαμε καθορίζεται από το πεδίο chunk-type. Συγκεκριμένα το πεδίο αυτό κωδικοποιείται με τέτοιο τρόπο ώστε από τα 2 πρώτα bit του αριθμού να καθορίζεται η ενέργεια που πρέπει να εφαρμοστεί σε περίπτωση που κάποιο από τα δύο άκρα δεν μπορεί να αναγνωρίσει τον τύπο του chunk. Έτσι έχουμε τις παρακάτω ενέργειες:

00 – Όταν τα 2 πρώτα bit πάρουν αυτήν την τιμή, τότε θα πρέπει να σταματήσει η επεξεργασία του συγκεκριμένου πακέτου και το πακέτο αυτό να απορριφθεί. Σταματάει επίσης και η επεξεργασία των επιμέρους chunk του πακέτου.
01 - Όταν τα 2 πρώτα bit πάρουν αυτήν την τιμή, τότε θα πρέπει να σταματήσει η επεξεργασία του συγκεκριμένου πακέτου και το πακέτο αυτό να απορριφθεί. Σταματάει επίσης και η επεξεργασία των επιμέρους chunk του πακέτου και ενημερώνεται ο αποστολέας του chunk αυτού με ένα ERROR chunk όπου στο πεδίο type θα αναγράφεται «Unrecognized Chunk Type» (μη-αναγνωρισμένος τύπος).
10 - Όταν τα 2 πρώτα bit πάρουν αυτήν την τιμή, τότε το chunk αυτό απορρίπτεται. Η επεξεργασία συνεχίζεται.
11 - Όταν τα 2 πρώτα bit πάρουν αυτήν την τιμή, τότε το chunk αυτό απορρίπτεται, αλλά ενημερώνεται ο αποστολέας του chunk αυτού με ένα ERROR chunk όπου στο πεδίο type θα αναγράφεται «Unrecognized Chunk Type» (μη-αναγνωρισμένος τύπος).

2.1.6 Ταυτοποίηση πακέτων

Στην παράγραφο 1,5 (σχήμα 16) βλέπουμε ότι στην επικεφαλίδα του SCTP συμπεριλαμβάνονται δύο πεδία, το **Verification Tag** και το **checksum** (CRC 32bit).

Η τιμή του πεδίου **Verification Tag** καθορίζεται από τα δύο άκρα της σύνδεσης κατά τη διαδικασία του association start-up. Το πεδίο αυτό ουσιαστικά ορίζει σε ποιό association ανήκει κάθε πακέτο, και έτσι τα πακέτα με λανθασμένη τιμή στο πεδίο αυτό απορρίπτονται. Η τεχνική αυτή χρησιμοποιείται για την προστασία της σύνδεσης από τις ***blind masquerade attacks (απειλές τυφλής μεταμφίεσης).*** Στις επιθέσεις αυτές ο εισβολέας έχει ως στόχο να ξεγελάσει τους μηχανισμούς ασφάλειας του δικτύου και να θεωρηθεί ως εξουσιοδοτημένος ή έμπιστος χρήστης. Αυτό επιτυγχάνεται με την υφαρπαγή των στοιχείων ταυτότητας ενός «νόμιμου» χρήστη, καθώς και με την επανάληψη μηνυμάτων που έχουν αντιγραφεί από μία προηγούμενη «νόμιμη» σύνοδο.

Η τιμή του πεδίου **checksum** ορίζεται από τον αποστολέα και προσδίδει μια επιπλέον προστασία κατά την αλλοίωση των δεδομένων (data corruption) στο δίκτυο. Ο παραλήπτης, εκτελεί τον αλγόριθμο Κυκλικού Κώδικα Πλεονασμού (Cyclic Redundancy Check *CRC32c*) στο SCTP πακέτο και αν το αποτέλεσμα είναι διαφορετικό από την τιμή που έχει το πεδίο checksum, τότε απορρίπτει το πακέτο.

2.1.7 Διαχείριση μονοπατιού

Η λειτουργία της διαχείρισης μονοπατιού (path management) επιλέγει την κατάλληλη διεύθυνση μεταφοράς δεδομένων για κάθε ένα sctp πακέτο βασιζόμενη στις οδηγίες που έδωσε ο SCTP χρήστης. Η λειτουργία αυτή ελέγχει τη διαθεσιμότητα την μόνιμης (κύριας) σύνδεσης αλλά και των εναλλακτικών (**multi-homing**). Ο έλεγχος των συνδέσεων γίνεται με τα **HEARTBEATS**. Όπως αναφέραμε και στην παράγραφο 2.1.1 τα πακέτα κτύπου καρδιάς (heartbeat) στέλνονται περιοδικά σε όλες τις διευθύνσεις και έτσι ελέγχεται η συνδεσιμότητα των άκρων. Η λειτουργία της διαχείρισης μονοπατιού είναι επίσης υπεύθυνη και για την ενημέρωση των δύο άκρων κατά την έναρξη μιας συσχέτισης για τις διευθύνσεις που έχει κάθε άκρο (οι δύο σταθμοί ανταλλάσσουν διευθύνσεις ώστε κάθε άκρο να ξέρει τις διευθύνσεις του απέναντι άκρου).

Στον παραλήπτη, η λειτουργία της διαχείρισης μονοπατιού είναι υπεύθυνη για την ταυτοποίηση της συσχέτισης που ανήκει το κάθε πακέτο.

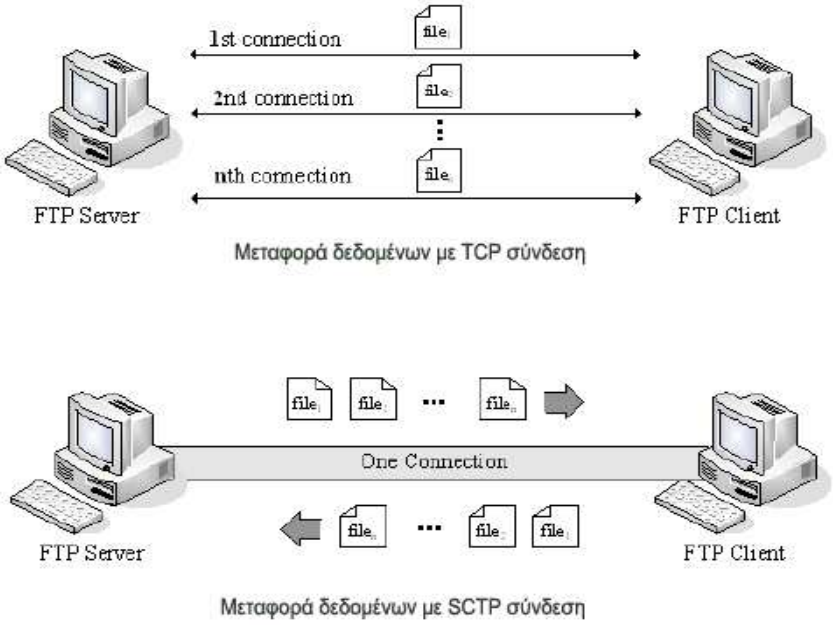
2.2 Εφαρμογές που μπορεί να έχει

Πολλές εφαρμογές μπορούν να χρησιμοποιήσουν το SCTP σαν πρωτόκολλο μεταφοράς. Οι εφαρμογές αυτές επωφελούνται από το SCTP πρωτόκολλο το οποίο εξασφαλίζει:

- Αξιοπιστία μεταφοράς των δεδομένων
- Προστασία της σύνδεσης από επιθέσεις (προστασία από flooding και Masquerade attacks).
- Απόδοση μεταφοράς δεδομένων λόγω των streams (όπου σε κάθε stream τα μηνύματα μεταφέρονται ανεξάρτητα – *Multi-Streaming*)
- Έλεγχο των φυσικών συνδέσεων και διαχείριση μονοπατιού (σε ένα δίκτυο το sctp με τη λειτουργία του *Multi-Homing* εξασφαλίζει ότι τα δεδομένα θα μεταφερθούν από τις εναλλακτικές διευθύνσεις σε περίπτωση βλάβης της κύριας διεύθυνσης)
- Επιλογή αποδοχής δεδομένων με σειρά (ordered delivery) ή χωρίς (un-ordered delivery)

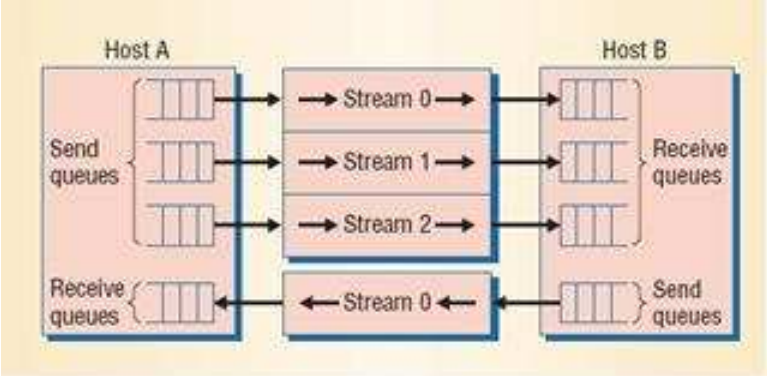
Τέτοιες εφαρμογές, είναι εφαρμογές PSTN τηλεφωνίας, όπου απαιτεί έλεγχο των φυσικών συνδέσεων, προστασία της σύνδεσης από masquerade attacks και μεταφορά δεδομένων με un-ordered delivery, εφαρμογές VoIP (Voice Over IP) , όπου έχουμε τηλεφωνικές κλήσεις μέσω του IP πρωτοκόλλου, εφαρμογές FTP (εφαρμογή ανταλλαγής αρχείων), όπου απαιτείται αξιοπιστία μεταφοράς δεδομένων και απόδοση μεταφοράς, όπως επίσης και προστασία των συνδέσεων από flooding και Masquerade attacks.

Μια εφαρμογή **FTP** μπορεί να επωφεληθεί από το SCTP. Το FTP χρησιμοποιεί για πρωτόκολλο μεταφοράς το TCP. Αυτό του εξασφαλίζει της αξιόπιστη μεταφορά δεδομένων που η εφαρμογή απαιτεί. Όμως το FTP βασίζεται σε δύο ροές (streams). Μια ροή στην πόρτα 21 για τη μεταφορά των μηνυμάτων ελέγχου και μια ροή στην πόρτα 20 για την μεταφορά των δεδομένων. Αυτό έχει σαν συνέπεια να προκαλούνται αρκετά προβλήματα όταν στην σύνδεση των 2 άκρων παρεμβάλλονται firewalls (τοίχος προστασίας). Για παράδειγμα, ένας πελάτης μπορεί να συνδεθεί με έναν εξυπηρετητή μέσω ενός firewall , αλλά ο εξυπηρετητής δεν θα μπορεί να συνδεθεί με τον πελάτη για την ανταλλαγή δεδομένων (το firewall θα μπλοκάρει την πόρτα 20). Γι' αυτό το πρωτόκολλο FTP επεκτείνεται και χρησιμοποιεί "passive" (παθητικές) συνδέσεις. Αυτό λοιπόν αποφεύγεται όταν γίνεται χρήση του SCTP πρωτοκόλλου, μιας και με την λειτουργία Multi-Stream, τα δεδομένα (όπως και τα μηνύματα ελέγχου) στέλνονται σε διαφορετικά streams μέσα σε ένα association (Σχήμα 33).



Σχήμα 33

Το SCTP πρωτόκολλο μπορεί επίσης να χρησιμοποιηθεί και από τις εφαρμογές που κάνουν χρήση του **HTTP** πρωτοκόλλου, όπως για παράδειγμα οι internet browsers (π.χ. ms internet explorer, mozilla firefox κτλ). Το πρωτόκολλο HTTP καθορίζει τις αιτήσεις (*requests*) που μπορεί να στείλει ένας πελάτης σε έναν εξυπηρετητή καθώς και τις απαντήσεις (*replies*) που μπορεί να στείλει ο εξυπηρετητής. Κάθε αίτηση έχει ένα URL και μπορεί να είναι μια HTML σελίδα, μία εικόνα, μία υπηρεσία κτλ. Μ'αυτόν τον τρόπο μεταφέρονται τα δεδομένα ανάμεσα σε έναν εξυπηρετητή και σε έναν πελάτη. Για την μεταφορά των δεδομένων αυτών το HTTP χρησιμοποιεί το πρωτόκολλο TCP. Συγκεκριμένα χρησιμοποιεί μια ξεχωριστή TCP σύνδεση για κάθε αίτηση/απάντηση. Αυτό έχει σαν αποτέλεσμα τη «σπατάλη» χρόνου. Για παράδειγμα, ας σκεφτούμε ότι θέλουμε να φορτώσουμε στον browser μια html σελίδα η οποία αποτελείται από κείμενο και φωτογραφίες. Το HTTP θα δημιουργήσει μια TCP σύνδεση για την εμφάνιση της HTML σελίδας (αίτηση/απάντηση) και στη συνέχεια θα δημιουργήσει τόσες συνδέσεις (διαδοχικά) όσες και οι φωτογραφίες που έχει η σελίδα. Με τη χρήση λοιπόν του SCTP πρωτοκόλλου αντί του TCP, επιτυγχάνονται μεγαλύτερες ταχύτητες φόρτωσης αφού μέσω του Multi-Streaming όλες οι φωτογραφίες της σελίδας μας (και κατά συνέπεια όλες οι αιτήσεις) θα φορτωθούν ταυτόχρονα, μια αίτηση σε κάθε stream (Σχήμα 34).


Host A
Host B
Send queues
Receive queues
Stream 0
Stream 1
Stream 2
Stream 0
Receive queues
Send queues

Σχήμα 34

2.3 Υλοποιήσεις σε λειτουργικά συστήματα και χρήση του από προγράμματα

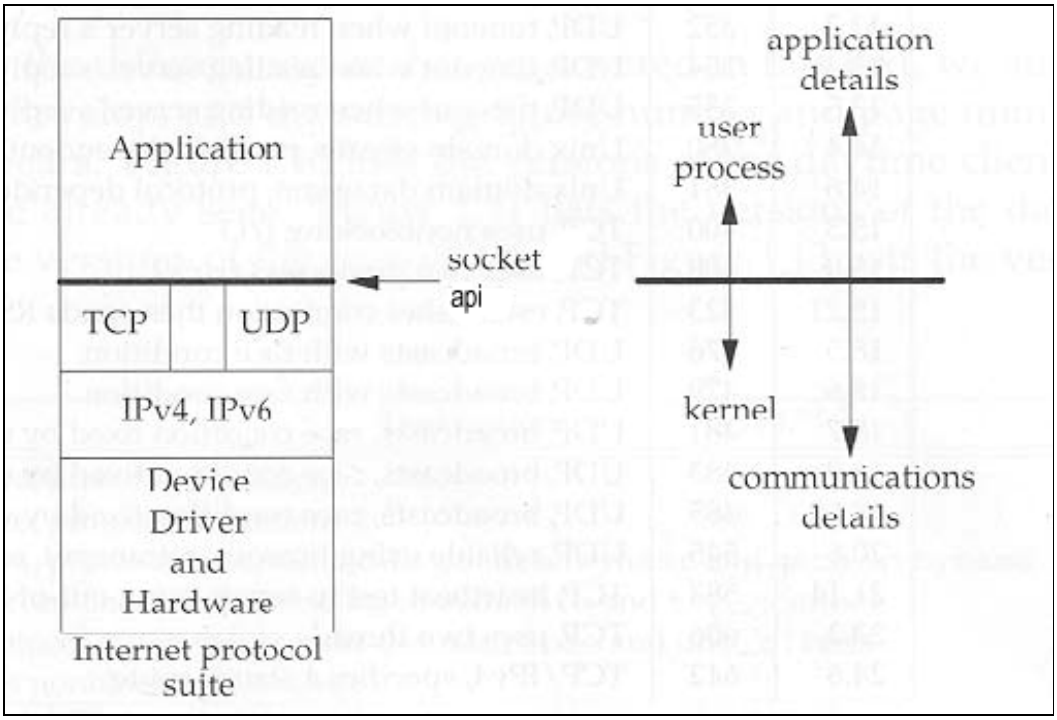
Το SCTP πρωτόκολλο έχει διάφορες υλοποιήσεις σε διάφορα λειτουργικά συστήματα. Υπάρχουν υλοποιήσεις για MS Windows, BSD (FreeBSD/OpenBSD και NetBSD), Unix/Linux/Solaris, Mac. Από το 2001 το SCTP μέσω του **Linux Kernel SCTP project (lksctp)** (sourceforge.net/projects/lksctp) υπάρχει ενσωματωμένο στον πυρήνα Linux σαν πειραματικό πρωτόκολλο δικτύου. Το SCTP υπάρχει σαν module και φορτώνεται στον πυρήνα του λειτουργικού συστήματος **Linux**.

Για την υποστήριξη του SCTP πρωτοκόλλου από τα προγράμματα, χρειάζεται να εγκαταστήσουμε ένα άλλο πακέτο. Το **SCTP tools**. Το πακέτο αυτό περιέχει διάφορες βιβλιοθήκες οι οποίες χρησιμοποιούνται από τις sctp εφαρμογές. Τέλος, το devel πακέτο (development) περιέχει το αρχείο sctp.h (header), με το οποίο μπορούμε να κάνουμε compile και build την εφαρμογή μας, καθώς και τις σελίδες του manual για τις sctp συναρτήσεις.

2.4 Το Socket API

Με τον όρο **API** ή αλλιώς **Διασύνδεση Προγραμματισμού Εφαρμογών (Application Program Interface)** εννοούμε μια διασύνδεση η οποία καθορίζει το σύνολο των λειτουργιών μιας εφαρμογής με το λογισμικό πρωτοκόλλων.

Όπως και το TCP και UDP, έτσι και το SCTP παρέχει ένα **Socket API** για τις εφαρμογές. Το socket api (Berkeley-BSD) αποτελεί την «πόρτα» μέσω της οποίας μια εφαρμογή «συνομιλεί» με το λογισμικό πρωτοκόλλων. Σκοπός του είναι η γενική επικοινωνία μεταξύ διεργασιών (που τρέχουν στον ίδιο ή σε διαφορετικούς υπολογιστές). Ουσιαστικά το socket api είναι μια βιβλιοθήκη συναρτήσεων τις οποίες χρησιμοποιεί μια εφαρμογή για να έχει πρόσβαση στο λογισμικό πρωτοκόλλου.



Σχήμα 35

Όπως βλέπουμε στο σχήμα 35 το Λ.Σ. έχει υλοποιημένη την στοίβα TCP/IP στον πυρήνα του. Έτσι ο προγραμματισμός των εφαρμογών που λειτουργούν πάνω από TCP/IP (αλλά και πάνω από SCTP) γίνεται μέσω του Socket API.

2.5 Το API του πρωτοκόλλου SCTP

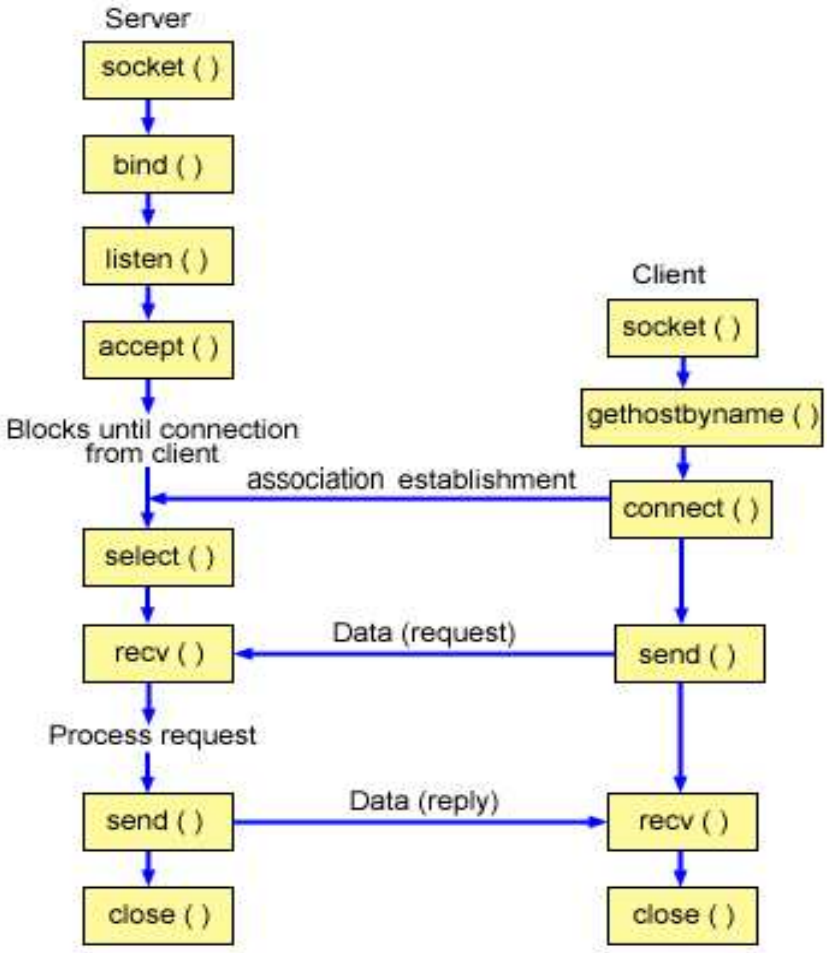
Το **SCTP API** υποστηρίζει δύο διεπαφές:

1. **Ένα σε πολλά**
   Ένα socket SCTP τύπου ένα- σε -πολλά είναι ικανό να ελέγχει πολλές συσχετίσεις SCTP. Αυτό είναι παρόμοιο με το *UDP socket* το οποίο μπορεί να επικοινωνήσει με πολλούς δέκτες. Εδώ, σε κάθε μία από αυτές τις συσχετίσεις δίνεται ένας μοναδικός αριθμός ταυτότητας (association id). 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 είναι παρόμοια με το *TCP socket* (που είναι πρωτόκολλο με σύνδεση). Εδώ έχουμε μόνο μία συσχέτιση. Ο σημαντικότερος λόγος που καθορίστηκε αυτή η διεπαφή είναι για να επιτρέψει σε εφαρμογές γραμμένες πάνω από πρωτόκολλα με σύνδεση (κυρίως εφαρμογές γραμμένες σε TCP) να τροποποιηθούν εύκολα σε SCTP.

Για την υλοποίηση της εφαρμογής μας χρησιμοποιήσαμε τον δεύτερο τρόπο μιας και είναι ο πιό εύκολος και δεν μειονεκτεί σε σχέση με τον πρώτο.
Σύμφωνα με την δεύτερη διεπαφή λοιπόν (1-προς-1) ο server δημιουργεί ένα socket και ορίζει μια πόρτα σ'αυτό. Έτσι χρησιμοποιεί αυτό το socket για να δέχεται τις συνδέσεις του client. Όπως ο server, έτσι και ο client δημιουργεί κι αυτός ένα socket και μ'αυτό συνδέεται στον server. Έτσι λοιπόν ο client και ο server χρησιμοποιούν socket file descriptors για να στέλνουν και να λαμβάνουν μηνύματα. Η διαδικασία αυτή όπως είπαμε είναι παρόμοια με μια tcp σύνδεση.

Στο σχήμα 36 βλέπουμε τη διαδικασία αυτή.



Σχήμα 36

Η διαδικασία είναι παρόμοια με αυτή του Socket API. Η διεργασία που δημιουργεί το Socket το χειρίζεται μέσω ενός θετικού ακεραίου – **socket descriptor** (όπως ένα file descriptor).

Όπως βλέπουμε από το σχήμα 36, ο **Server** ακολουθεί τις παρακάτω κλήσεις του συστήματος:

α/α	**Κλήση**	**Περιγραφή**
1	socket()	Η κλήση αυτή δημιουργεί ένα socket file descriptor που αντιστοιχεί στον συγκεκριμένο SCTP κόμβο.
2	bind()	Με την κλήση bind() ορίζεται αρχική διεύθυνση (address και port) που συσχετίζεται με το συγκεκριμένο SCTP κόμβο.

3	listen()	Η κλήση listen() ετοιμάζει ένα SCTP κόμβο να δεχθεί κάποια σύνδεση (association).
4	accept()	Με την κλήση αυτή γίνεται αποδοχή κάποιας αίτησης σύνδεσης η διεργασία μπλοκάρει τον κόμβο μέχρι μια δημιουργηθεί μία νέα συσχέτιση όπου και επιστέφεται ένας νέος socket descriptor.
5	read() / write() *ή* send() / recv() *ή* sendmsg() / recvmsg()	Οι κλήσεις αυτές χρησιμοποιούνται από τον server για να λάβει/στείλει μηνύματα
6	close()	Με την κλήση close() τερματίζεται μια συσχέτιση από τον server.

Με τον ίδιο τρόπο και ο **Client** ακολουθεί τις παρακάτω κλήσεις του συστήματος:

α/α	**Κλήση**	**Περιγραφή**
1	socket()	Η κλήση αυτή δημιουργεί ένα socket file descriptor που αντιστοιχεί στον συγκεκριμένο SCTP κόμβο.
2	connect()	Με την κλήση connect() ξεκινάει μια συσχέτιση μεταξύ client-server.
3	read() / write() *ή* send() / recv() *ή* sendmsg() / recvmsg()	Οι κλήσεις αυτές χρησιμοποιούνται από τον client για να λάβει/στείλει μηνύματα
4	close()	Με την κλήση close() τερματίζεται μια συσχέτιση από τον client.

3. INETD.

3.1 Περιγραφή του inetd

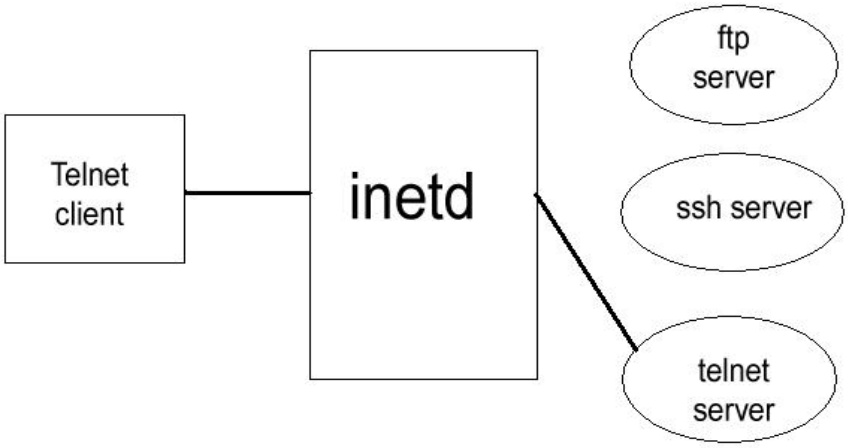
Πριν ξεκινήσουμε την περιγραφή της εφαρμογής inetd, είναι σημαντικό να καταλάβουμε την λειτουργία των *daemons* γενικότερα.

3.1.1 Γενικά για τους Daemons

Ένας **daemon** (δαίμονας) είναι ένα πρόγραμμα το οποίο τρέχει στο background ενός λειτουργικού συστήματος Unix. Πιο συγκεκριμένα, ένας daemon είναι ένας server ο οποίος εξυπηρετεί τους διάφορους clients προσφέροντάς τους δικτυακές υπηρεσίες (internet services) όπως για παράδειγμα e-mail ή file serving. Οι daemons εκτελούνται κατά την έναρξη (boot time) του λειτουργικού συστήματος. Στα Unix συστήματα υπάρχει ένας μεγάλος αριθμός από daemons. Το όνομα του κάθε daemon τελειώνει με το γράμμα “d” όπως για παράδειγμα ο “httpd”, ο “ftpd”, ο “sshd” κ.α. Η λειτουργία λοιπόν ενός Unix Daemon είναι παρόμοια με μια υπηρεσία (service) των λειτουργικών συστημάτων Microsoft Windows.

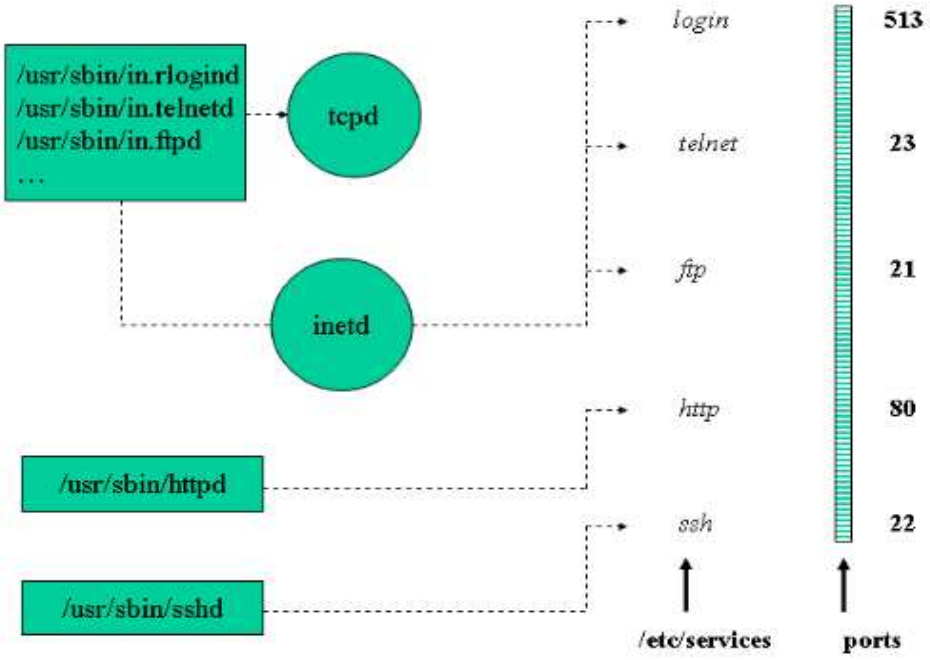
3.1.2 Ο inetd Daemon

Ένας από τους σημαντικότερους Unix daemons είναι ο **inetd daemon** ή αλλιώς Internet super-server. Ο inetd ακούει σε πολλαπλά ports για εισερχόμενες αιτήσεις σύνδεσης. Όταν λαμβάνει μία αίτηση, καλεί τον ανάλογο server. Μ’αυτόν τον τρόπο, ο inetd εκτελεί τον κάθε daemon μόνο όταν χρειάζεται. Για παράδειγμα ας σκεφτούμε την υπηρεσία telnet. Ο daemon για την υπηρεσία αυτή είναι ο telnetd (*/usr/sbin/in.telnetd*). Αντί να εκτελείται συνέχεια ο telnetd, αφήνουμε τη δουλειά αυτή να την κάνει ο inetd ο οποίος ακούει συνέχεια όπως είπαμε σε πολλαπλά ports. Έτσι όταν «ακούσει» κάποια αίτηση στο Port 23 (default telnet port), θα καταλάβει ότι πρόκειται για την υπηρεσία telnet και αυτόματα θα καλέσει τον telnetd (σχήμα 37). Το inetd είναι ένας “super-server”. Η χρήση ενός super-server λοιπόν επιτρέπει τους υπόλοιπους servers να εκτελούνται μόνο όταν χρειάζεται και να τερματίζονται όταν έχουν εκτελέσει τα αιτήματα που τους δόθηκαν.



Σχήμα 37

Με τον ίδιο τρόπο λοιπόν το inetd καλεί τους διάφορους daemons σύμφωνα με τον αριθμό του κάθε port από το οποίο ακούει κάποια αίτηση. Για να πραγματοποιηθεί αυτή η διαδικασία, ο inetd συμβουλεύεται δύο αρχεία, το **/etc/inetd.conf** (configuration file), το οποίο περιέχει τα ονόματα των διαφόρων υπηρεσιών και τα ονόματα των αντίστοιχων daemons που παρέχουν την υπηρεσία, και το **/etc/services**, το οποίο περιέχει τα ονόματα των διάφορων υπηρεσιών καθώς και τα αντίστοιχα ports στα οποία εκτελείται η κάθε υπηρεσία (σχήμα 38).



Σχήμα 38

Έτσι στο προηγούμενο παράδειγμα, ο inetd daemon θα λειτουργήσει ως εξής:

- Ακούει στην πόρτα 23 την αίτηση του client.
- Ψάχνει στο αρχείο /etc/services αν υπάρχει υπηρεσία που να εκτελείται στην πόρτα 23. Βρίσκει ότι η υπηρεσία που εκτελείται στην πόρτα αυτή είναι το telnet.
- Ζητάει από το /etc/inetd.conf το όνομα του daemon (ή του προγράμματος) που εκτελεί την υπηρεσία αυτή. Το inetd.conf απαντάει με το όνομα του daemon ο οποίος είναι ο in.telnetd.
- Εκτελεί τον in.telnetd και περιμένει για επόμενες αιτήσεις σύνδεσης.

Στο σχήμα 39 βλέπουμε τα περιεχόμενα του αρχείου **/etc/inetd.conf** :

```
# inetd services
# service       type    protocol  wait    user        server                  cmdline

ftp             stream  tcp       nowait  root        /usr/sbin/ftpd          in.ftpd -l
telnet          stream  tcp       nowait  root        /usr/sbin/telnetd       in.telnetd
finger          stream  tcp       nowait  bin         /usr/sbin/fingerd       in.fingerd
tftp            dgram   udp       wait    nobody      /usr/sbin/tftpd         in.tftpd
login           stream  tcp       nowait  root        /usr/sbin/rlogind       in.rlogind
MyServer1       stream  tcp       nowait  root                  /home/geobros/MyServer1
```

Σχήμα 39

Κάθε γραμμή του αρχείου αυτού αποτελείται από τα εξής πεδία: **service, type, protocol, wait, user, server, cmdline**.

- Το πεδίο ***service*** δηλώνει το όνομα του service. Για τα non-RPC services χρησιμοποιείται το αρχείο */etc/services* από το οποίο ταυτοποιείται το service, για τα RPC services χρησιμοποιείται το αρχείο */etc/rpc*.
- Το πεδίο ***type*** καθορίζει τον τύπο του socket του συγκεκριμένου service. Αυτό μπορεί να είναι είτε stream (για connection-oriented πρωτόκολλα) είτε dgram (για datagram πρωτόκολλα). Έτσι όταν έχουμε κάποιο service που λειτουργεί με tcp πρωτόκολλο, το πεδίο αυτό πρέπει να είναι “stream”, αλλιώς αν λειτουργεί με udp θα πρέπει να είναι “dgram”. Οι τιμές που παίρνει αυτό το πεδίο είναι οι: *stream/dgram/raw/rdm/seqpacket.*
- Το πεδίο ***protocol*** δηλώνει το πρωτόκολλο με το οποίο λειτουργεί το service. Το inetd υποστηρίζει δύο πρωτόκολλα, το TCP και το UDP. Το πρωτόκολλα πρέπει να υπάρχουν στο αρχείο *etc/protocols* .
- Το πεδίο ***wait*** αναφέρεται μόνο στα services τύπου dgram. Μπορεί να δηλωθεί ως *wait* ή *nowait*. Όταν δηλώνεται ως wait τότε το inetd εκτελεί

μόνο έναν server όταν «ακούσει» το συγκεκριμένο port. Έχουμε δηλαδή single-thread λειτουργία. Όταν δηλώνεται ως nowait τότε ο inetd συνεχίζει να ακούει στο συγκεκριμένο port ακόμα και μετά την εκτέλεση του server. Έχουμε δηλαδή multi-thread λειτουργία.

- Το πεδίο ***user*** δηλώνει το Login ID του χρήστη στον οποίο ανήκει η διεργασία (ή το service) που εκτελείται. Συνήθως εδώ ορίζουμε τον root εκτός και αν το service απαιτεί άλλο account.
- Το πεδίο ***server*** δηλώνει την ακριβή τοποθεσία του server που θα εκτελεστεί (π.χ. /usr/sbin/telnetd).
- Το πεδίο ***cmdline*** δηλώνει την εντολή που θα εκτελεστεί για να εκτελεστεί ο server που θέλουμε, καθώς και διάφορα άλλα ορίσματα που μπορούμε να περάσουμε στην εντολή αυτή (π.χ. in.ftpd –l).

Στο σχήμα 40 βλέπουμε τα περιεχόμενα του αρχείου **/etc/services** :

```
# service                port/protocol

echo                     7/tcp
echo                     7/udp
ftp-data                 20/tcp
ftp                      21/tcp
ssh                      22/tcp
ssh                      22/udp
telnet                   23/tcp
MyServer1                3333/tcp
```

Σχήμα 40

Κάθε γραμμή του αρχείου αυτού αποτελείται από τα εξής πεδία: **service** και **port/protocol**.

- Το πεδίο ***service*** δηλώνει το όνομα το service και
- Το πεδίο ***port/protocol*** δηλώνει τον αριθμό της πόρτας και το πρωτόκολλο με το οποίο λειτουργεί το service αντίστοιχα.

Τόσο το **/etc/inetd.conf** όσο και το **/etc/services** διαβάζονται από το Inetd κατά την έναρξή του, όταν δηλαδή φορτώνεται ο daemon.

3.1.3 Το πακέτο openbsd-inetd

Για να έχουμε πρόσβαση στον κώδικα του inetd πρέπει καταρχάς να κατεβάσουμε το κατάλληλο πακέτο του inetd το οποίο περιέχει τον πηγαίο κώδικα του inetd (πηγαίο πακέτο). Αυτό γίνεται με δύο τρόπους, είτε μέσω του προγράμματος διαχείρισης πακέτων της linux διανομής μας (π.χ. synaptic, aptitude,apt-get) όπως για παράδειγμα μέσω της εντολής apt-get, όπου ανοίγοντας την κονσόλα και πληκτρολογώντας την εντολή:

sudo apt-get source openbsd-inetd-######## *(εδώ βάζουμε την έκδοση που θέλουμε)*

κατεβάζουμε το source code (πηγαίο κώδικα) του openbsd inetd, είτε πηγαίνοντας στον κατάλληλο διαδικτυακό τόπο (π.χ. στη σελίδα με τα πακέτα της debian: http://packages.debian.org) και κατεβάζοντας όποια έκδοση του πακέτου θέλουμε. Η έκδοση του inetd που χρησιμοποιήσαμε ήταν η ***openbsd-inetd-0.20080125-1*** η οποία είναι και η πιο πρόσφατη unstable έκδοση του inetd (http://packages.debian.org/unstable/net/openbsd-inetd).
Η τελευταία stable έκδοση του openbsd-inetd είναι η ***0.20050402***.
(http://packages.debian.org/etch/openbsd-inetd).

Αφού κατεβάσουμε το πακέτο με το source code, παρατηρούμε ότι αυτό αποτελείται από 3 αρχεία:

openbsd-inetd_0.20080125-1.dsc
openbsd-inetd_0.20080125.orig.tar.gz
openbsd-inetd_0.20080125-1.diff.gz

Το αρχείο με την κατάληξη ***.dsc*** περιέχει πληροφορίες του πακέτου πηγαίου κώδικα.
Το αρχείο με την κατάληξη ***.diff*** περιέχει τις διαφορές του κώδικα της τωρινής έκδοσης σε σχέση με τον κώδικα της προηγούμενης. Αν αυτές οι αλλαγές εφαρμοστούν στον κώδικα της προηγούμενης έκδοσης τότε θα μας δώσουν την ικανότητα να φτιάξουμε ένα πακέτο debian.
Τέλος, το αρχείο με την κατάληξη ***.orig.tar.gz*** περιέχει τον κώδικα της έκδοσης του πακέτου που κάναμε download (θα πρέπει πρώτα να αποσυμπιέσουμε το αρχείο).

3.2 Ανάλυση Υπάρχοντος Κώδικα inetd

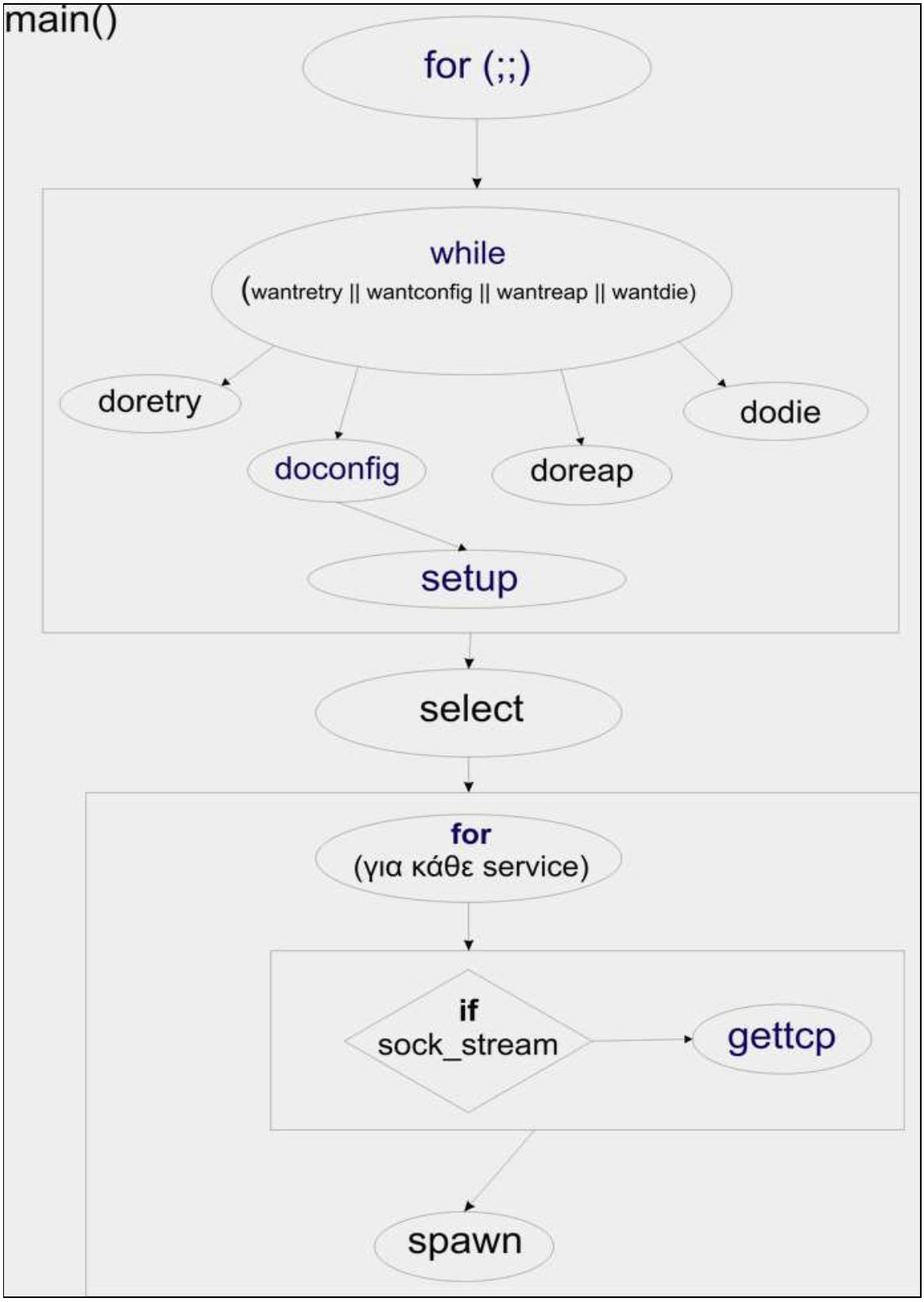
Όπως είπαμε και προηγουμένως, ο κώδικας του inetd βρίσκεται στο αρχείο **openbsd-inetd_0.20080125.orig.tar.gz**. Αποσυμπιέζοντας λοιπόν το αρχείο αυτό, έχουμε πρόσβαση στον κώδικα του inetd.

Ο κώδικας του inetd είναι γραμμένος σε γλώσσα **C**. Το inetd χρησιμοποιεί το udp api για τις udp υπηρεσίες και tcp api για τις tcp υπηρεσίες. Θα δούμε τις σημαντικότερες συναρτήσεις του inetd και θα περιγράψουμε το τι ακριβώς κάνουν ώστε να καταλάβουμε την λογική σύμφωνα με την οποία δουλεύει ο super-server.

Οι σημαντικότερες συναρτήσεις του κώδικα του inetd είναι οι εξής:

```
void doconfig(void);
void dodie(void);
void spawn(struct servtab *, int);
int  gettcp(struct servtab *);
void setup(struct servtab *);
```

Το Inetd εκτελεί αυτές τις μεθόδους σύμφωνα με το σχήμα 41.


main()
for (;;)
while
(wantretry || wantconfig || wantreap || wantdie)
doretry
doconfig
doreap
dodie
setup
select
for
(για κάθε service)
if
sock_stream
gettcp
spawn

Σχήμα 41

Όπως βλέπουμε και από το σχήμα 41, το inetd εκτελεί έναν ατέρμονα βρόγχο for και έτσι όλοι οι έλεγχοι που βλέπουμε στο σχήμα εκτελούνται συνέχεια.

Αρχικά όμως ας ξεκινήσουμε να περιγράφουμε τι ακριβώς κάνει η κάθε μέθοδος.

Η μέθοδος **doconfig** είναι η μέθοδος η οποία διαβάζει κάθε γραμμή του αρχείου */etc/inetd.conf* (ένα service δηλαδή) και αποθηκεύει σε μεταβλητές το κάθε πεδίο της κάθε γραμμής. Κατ'αυτόν τον τρόπο γνωστοποιούνται τα *service(name), type, protocol, wait, user, server, cmdline* (μέσω διάφορων ελέγχων) οπότε ξέρουμε έτσι τον τύπο, το πρωτόκολλο και τον server που εκτελείται κάθε φορά που καλείται ένα service. Αυτό θα βοηθήσει και την εξέλιξη του προγράμματος, αφού για διαφορετικές τιμές στα παραπάνω πεδία εκτελούνται διαφορετικές συναρτήσεις. Για κάθε service (γραμμή στο inetd.conf) λοιπόν η μέθοδος αυτή δημιουργεί έναν ξεχωριστό δείκτη. Η μέθοδος αυτή με τη σειρά της καλεί τη μέθοδο *setup* την λειτουργία της οποίας θα περιγράψουμε παρακάτω.

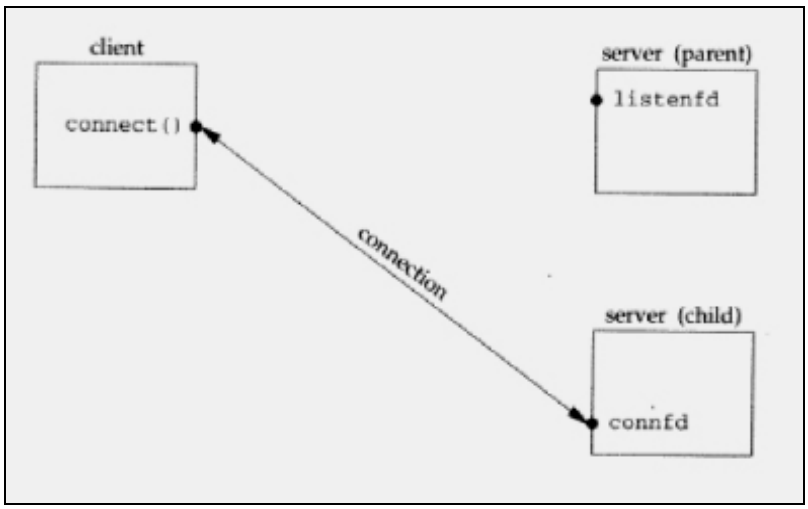
Η μέθοδος **setup** ορίζει έναν *socket file descriptor* που αντιστοιχεί σε έναν TCP κόμβο (οπού εδώ κόμβος είναι το κάθε service του inetd,conf). Στη συνέχεια αφού το socket αυτό περνάει από διάφορους ελέγχους, ορίζεται η διεύθυνση (address και port) που συσχετίζεται με το συγκεκριμένο socket μέσω της *bind*. Αφού ορίζεται η διεύθυνση ακολουθεί η κλήση *listen* η οποία ετοιμάζει τον server να δεχθεί κάποια σύνδεση από έναν client. Ουσιαστικά, αυτό που κάνει η setup είναι να κάνει χρήση του API της στοίβας TCP/IP , του socket API δηλαδή που περιγράψαμε στην παράγραφο 2.4 , ώστε να προετοιμάσει τον server όπου στη συγκεκριμένη περίπτωση ο server είναι το service (που διαβάζει η μέθοδος config από το αρχείο inetd.conf).

Η μέθοδος **gettcp** ουσιαστικά κάνει το *accept*, δηλαδή αποδέχεται κάποιο αίτημα (από τον client) για σύνδεση με τον server(με το κατάλληλο service δηλαδή).

Η μέθοδος **dodie** τερματίζει την σύνδεση μέσω της *close* η οποία τερματίζει τον socket file descriptor.

Η μέθοδος **spawn** είναι η μέθοδος στην οποία γίνονται οι περισσότεροι έλεγχοι. Εδώ ελέγχεται η λειτουργία του Inetd ανάλογα με τον τύπο του service (π.χ. stream ή dgram), ενημερώνεται το log αρχείο για την εξέλιξη του προγράμματος, κ.α. Η σημαντικότερη λειτουργία της spawn είναι ότι εδώ δημιουργούνται οι διεργασίες. Οι διεργασίες στο Unix δημιουργούνται με τη μέθοδο fork. Έτσι κι εδώ, γίνεται κλήση της fork με τον εξής τρόπο:
Pid = fork()
Η διεργασία που καλεί την fork ονομάζεται γονέας(parent). Η νέα διεργασία ονομάζεται παιδί(child). Ουσιαστικά αυτό που επιτυγχάνεται με την fork είναι ότι ο

client επικοινωνεί με έναν «αφοσιωμένο» αντίγραφο του server, ενώ ο «αρχικός» server μπορεί να δέχεται νέες κλήσεις σύνδεσης (accept) . Για την διεργασία-γονέα η pid είναι η ταυτότητα (ID) της διεργασίας-παιδιού ενώ για την διεργασία-παιδί το pid είναι μηδέν. Το αποτέλεσμα της fork φαίνεται στο σχήμα 42.



Σχήμα 42

3.2.1 Δοκιμή λειτουργίας του inetd με TCP server

Ας δούμε πώς δουλεύει στην πράξη το inetd. Θα κατασκευάσουμε έναν απλό TCP server και έναν απλό TCP client. Με την εκτέλεσή τους, ο client θα συνδέεται με τον server και στη συνέχεια οι δύο σταθμοί θα ανταλλάσσουν κάποιο μήνυμα.

Ο server θα τρέχει μέσω του inetd. Αυτό θα μας βοηθήσει να καταλάβουμε καλύτερα τον τρόπο λειτουργίας των δικτυακών εφαρμογών άρα και του ίδιου του inetd.

Αρχικά ας κατασκευάσουμε τον server. Ο κώδικας του server είναι ο παρακάτω (σχήμα 43):

```
#include <unistd.h>   /* fork(), write(), read(), close() */

const char minima[]="\nto mhnyma pou pire o server einai to:\n";
char buffer2[4096] =""; //orizw to buffer

const char MESSAGE[] = "\ntcp server_test!!!\n"; //orizw to
mhnyma
```

```
int main(){
      write(1, MESSAGE, strlen(MESSAGE));
      read(0, buffer2, sizeof(buffer2));
}
```

Σχήμα 43

Όπως βλέπουμε, δεν χρειάζεται να ορίσω το socket, να κάνω bind , listen και accept μιας και αυτά θα τα κάνει ο ίδιος ο inetd. Έτσι αντί για socket descriptor εδώ χρησιμοποιούμε απλό file descriptor, οπότε αρκεί να χρησιμοποιήσουμε τα stdin και stdout. Για είσοδο-stdin χρησιμοποιούμε τον file descriptor 0, ενώ για την έξοδο-stdout το 1. Έτσι στα read και write , στην θέση του file descriptor γράφουμε τα 0 και 1 αντίστοιχα.

Σώζουμε λοιπόν το αρχείο σαν MyServer_tcp.c και στη συνέχεια το κάνουμε compile με τον gcc δίνοντας την παρακάτω εντολή στην κονσόλα:

```
$gcc –c MyServer_tcp.c –o MyServer_tcp
```

Στη συνέχεια ορίζουμε το port και το service στο αρχείο **/etc/services** ως:

```
MyServer_tcp          3335/tcp
```

Τέλος ορίζουμε το service στο αρχείο **/etc/inetd.conf** ως:

```
MyServer_tcp  stream  tcp    nowait nobody      /home/geobros/MyServer_tcp
```

Στη συνέχεια γράφουμε τον κώδικα για τον tcp client. Αρχικά φορτώνουμε τις απαραίτητες βιβλιοθήκες, όπως sys/socket.h, netinet/in.h κτλ. Η socket.h είναι η βιβλιοθήκη για τις λειτουργίες του BSD Socket ενώ η netinet/in.h είναι η βιβλιοθήκη για την υποστήριξη της οικογένειας των πρωτοκόλλων AF_INET. Έπειτα, ορίζουμε το socket με την εντολή:

```
int clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Αυτό θα επιστρέψει μια ακέραια τιμή αν εκτελεστεί σωστά, ή το -1 αν αποτύχει.

Στη συνέχεια γνωστοποιούμε την διεύθυνση του server στον οποίο θα συνδεθεί ο client με την εντολή:

```
gethostbyaddr(remoteHost, strlen(remoteHost), AF_INET);
```

Ορίζουμε την οικογένεια πρωτοκόλλων καθώς και το port, στο οποίο ακούει ο server με τον οποίο θέλουμε να συνδεθούμε, με τις παρακάτω εντολές:

```
serverName.sin_family = AF_INET;
serverName.sin_port = htons(remotePort);
```

Τώρα είμαστε έτοιμοι να συνδεθούμε με τον server, απλά με την κλήση της:

```
status = connect(clientSocket, (struct sockaddr*) &serverName,
sizeof(serverName));
```

όπου servername η τιμή που επέστρεψε η gethostbyaddr.

Απο εδώ και πέρα αρχίζει η μεταφορά των δεδομένων μεταξύ client-server. Η αποστολή δεδομένων γίνεται με την κλήση της:

```
write(clientSocket, MESSAGE, strlen(MESSAGE));
```

ενώ η λήψη γίνεται με την κλήση της:

```
read(clientSocket, buffer, sizeof(buffer));
```

Τέλος, για τον τερματισμό της σύνδεσης καλούμε την:

```
close(clientSocket);
```

Όπως και με τον server, δίνουμε την εντολή:

$gcc –c MyClient_tcp.c –o MyClient_tcp

για το compile.

Πριν τρέξουμε τον MyClient_tcp για τη σύνδεση του με τον MyServer_tcp, πρέπει να επανακινήσουμε τον inetd daemon, μιας και ενημερώσαμε τα αρχεία /etc/services και /etc/inetd.conf.

Δίνουμε λοιπόν στην κονσόλα την εντολή:

```
$sudo /etc/init.d/openbsd-inetd restart
```

Τρέχουμε λοιπόν τον client και ορίζουμε τη διεύθυνση του server (εδώ τρέχει τοπικά οπότε είναι 127.0.0.1[loop-back interface] και το Port το οποίο όπως ορίσαμε κ στο inetd.conf είναι το 3335.

Το αποτέλεσμα της σύνδεσης είναι (σχήμα 44):

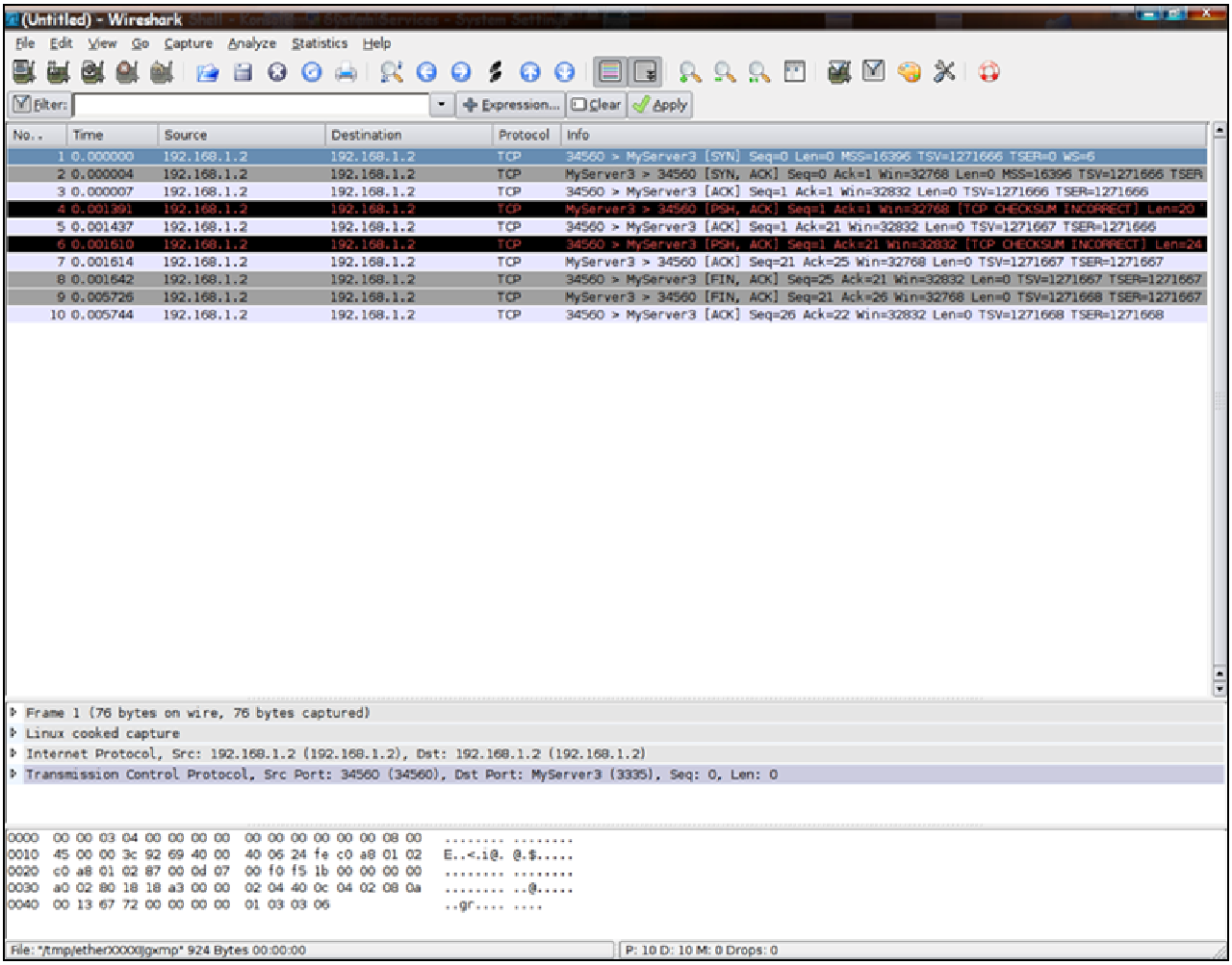
```
geobros@geobros:~/Temp$ ./MyClient_tcp 127.0.0.1 3335
clientName: 127.0.0.1
message:
tcp server_test!!!
 received from server 127.0.0.1

sending packets to the server 127.0.0.1
message:
client_to_server test!
 send to server
```

Σχήμα 44

Στο σχήμα 44 βλέπουμε τα αποτελέσματα. Ο client συνδέεται με τον server με tcp σύνδεση και αφού λάβει από αυτόν το μήνυμα “tcp server test!!”, στέλνει στον server το μήνυμα “client to server test!”.

Για να επιβεβαιώσουμε ότι η μεταφορά έγινε με tcp σύνδεση, τρέχουμε το πρόγραμμα **wireshark/ethereal** το οποίο μας δείχνει την κίνηση και το είδος των πακέτων που μεταφέρονται στο Interface μας (σχήμα 45):



Σχήμα 45

Βλέπουμε λοιπόν μια TCP σύνδεση (SYN - 3way handshake) μια μεταφορά δεδομένων (PSH / ACK - αποστολή και λήψη) και έναν τερματισμό της σύνδεσης (FIN – shutdown).

3.3 Τροποποίηση του κώδικα του inetd

Στην παράγραφο 2 μελετήσαμε το πρωτόκολλο SCTP και είδαμε τα πλεονεκτήματά του σε σχέση με το TCP. Όπως ήδη γνωρίζουμε, ο inetd υποστηρίζει μόνο tcp και udp συνδέσεις. Θα τροποποιήσουμε λοιπόν τον κώδικα του inetd ώστε αυτός να μπορεί να εξυπηρετεί και sctp συνδέσεις.

Με την ανάλυση του κώδικα του inetd που περιγράψαμε στην παράγραφο 3,2 μπορούμε να βοηθηθούμε και να καταλάβουμε ποιες μέθοδοι είναι οι σημαντικότεροι καθώς και ποια τμήματα κώδικα θα πρέπει να τροποποιήσουμε. Το inetd όπως είδαμε χρησιμοποιεί το socket API για να επιτύχει τη σύνδεση κάποιου client με κάποιον tcp ή udp server. Θα τροποποιήσουμε λοιπόν την εφαρμογή κάνοντας χρήση και του SCTP API, ώστε να δώσουμε την δυνατότητα στο Inetd να εξυπηρετεί και sctp συνδέσεις.

Αρχικά, μιας και θα κάνουμε χρήση της sctp βιβλιοθήκης, δηλώνουμε τη βιβλιοθήκη αυτή στον κώδικα του Inetd (αρχείο inetd.c).

Αφού έγινε προσεκτική μελέτη του κώδικα του inetd (παράγραφος 3,2) , έγιναν πιο κατανοητά τα σημεία στα οποία γίνεται η χρήση του socket API. Συγκεκριμένα, είδαμε ότι για κάθε service, η μέθοδος **doconfig** γνωστοποιεί το όνομα,τον τύπο, το πρωτόκολλο, τη διεύθυνση και το port του. Στη συνέχεια αν ο τύπος του service είναι sock_stream και όχι sock_dgram, καλείται η μέθοδος **setup** στην οποία γίνεται και η κλήση του socket. Εδώ λοιπόν, προσθέτουμε έναν επιπλέον έλεγχο, όπου ελέγχουμε το string του πεδίου του πρωτοκόλλου που μας έγινε γνωστό από την doconfig. Εδώ γίνεται και η χρήση της μεθόδου **getprotobyname** η οποία μας επιστρέφει το όνομα του πρωτοκόλλου σύμφωνα με το αρχείο **/etc/protocols** το οποίο περιέχει τα ονόματα και τους αντίστοιχους αριθμούς των πρωτοκόλλων που υποστηρίζει το λειτουργικό μας σύστημα. Έτσι ουσιαστικά ελέγχουμε αν το service θα λειτουργεί πάνω στο tcp ή στο sctp πρωτόκολλο. Σε κάθε περίπτωση δημιουργείται το κατάλληλο socket του server. Το socket, σε κάθε περίπτωση, είτε είναι τύπου tcp είτε sctp, θα κάνει bind το κατάλληλο port του service και στη συνέχεια μέσω της listen θα «ακούει» για αιτήσεις sctp συσχετίσεων ή tcp συνδέσεων στο port αυτό. Έτσι, όταν στο αρχείο inetd.conf δηλώνεται service το οποίο λειτουργεί με sctp πρωτόκολλο, ο κώδικας θα ελέγχει αν το πρωτόκολλο αυτό υποστηρίζεται από το λειτουργικό σύστημα και θα πραγματοποιεί τις κλήσεις συστήματος μέσω του SCTP API.

Με την τροποποίηση του κώδικα του inetd, τροποποιήθηκαν και διάφορα άλλα απαραίτητα αρχεία, όπως για παράδειγμα το αρχείο makefile στο οποίο δηλώθηκε το κατάλληλο flag ώστε να γίνεται χρήση της βιβλιοθήκης sctp κατά το compiling (-lsctp).

3.3.1 Δοκιμή λειτουργίας του inetd με SCTP server

Με τον ίδιο τρόπο που περιγράψαμε την κατασκευή του tcp server και του tcp client, θα κατασκευάσουμε έναν sctp server (ο οποίος θα λειτουργεί μέσω του inetd) αλλά και τον αντίστοιχο sctp client. Η λογική είναι η ίδια μ'αυτή της παραγράφου 3,2,1. Θα γίνει χρήση του SCTP API.

Ο κώδικας του sctp server είναι ίδιος μ'αυτόν του tcp server, το μόνο πράγμα που θα αλλάξουμε εδώ είναι το ότι θα ορίσουμε ένα sctp port για τον server αυτόν στο αρχείο /etc/services , όπως επίσης το ότι θα ορίσουμε για πρωτόκολλο το SCTP αντί του tcp στο αρχείο /etc/inetd.conf

/etc/services

```
MyServer_sctp        4001/sctp
```

/etc/inetd.conf

```
MyServer_sctp  stream  sctp    nowait nobody  /home/geobros/MyServer_sctp
```

Το γεγονός αυτό λοιπόν, θα κάνει τον inetd να δεί το service σαν sctp και έτσι να ορίσει το κατάλληλο socket αυτό. Μέσω αυτού του sctp server λοιπόν θα δούμε στην πράξη τον τρόπο με τον οποίο ο inetd εκτός από udp και tcp συνδέσεις τώρα θα εξυπηρετεί και sctp συνδέσεις.

Για τον sctp client, ο κώδικας θα είναι ίδιος με αυτόν του tcp client με την διαφορά ότι τώρα η κλήση του socket θα είναι η εξής:

```
int clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
```

Επίσης, για την αποστολή και λήψη των δεδομένων χρησιμοποιήσαμε τις μεθόδους του SCTP api **sctp_sendmsg()** και **sctp_recvmsg()** αντίστοιχα.

Για το compile δίνουμε την εντολή:

```
$gcc –c MyClient_sctp.c –o MyClient_sctp  -lsctp
```

Το –lsctp που προσθέσαμε στο τέλος δηλώνει ότι στον κώδικα χρησιμοποιήσαμε την βιβλιοθήκη του sctp πρωτοκόλλου.

Όμοια με πριν, τρέχουμε τον client και παίρνουμε τα αποτελέσματα του σχήματος 46.

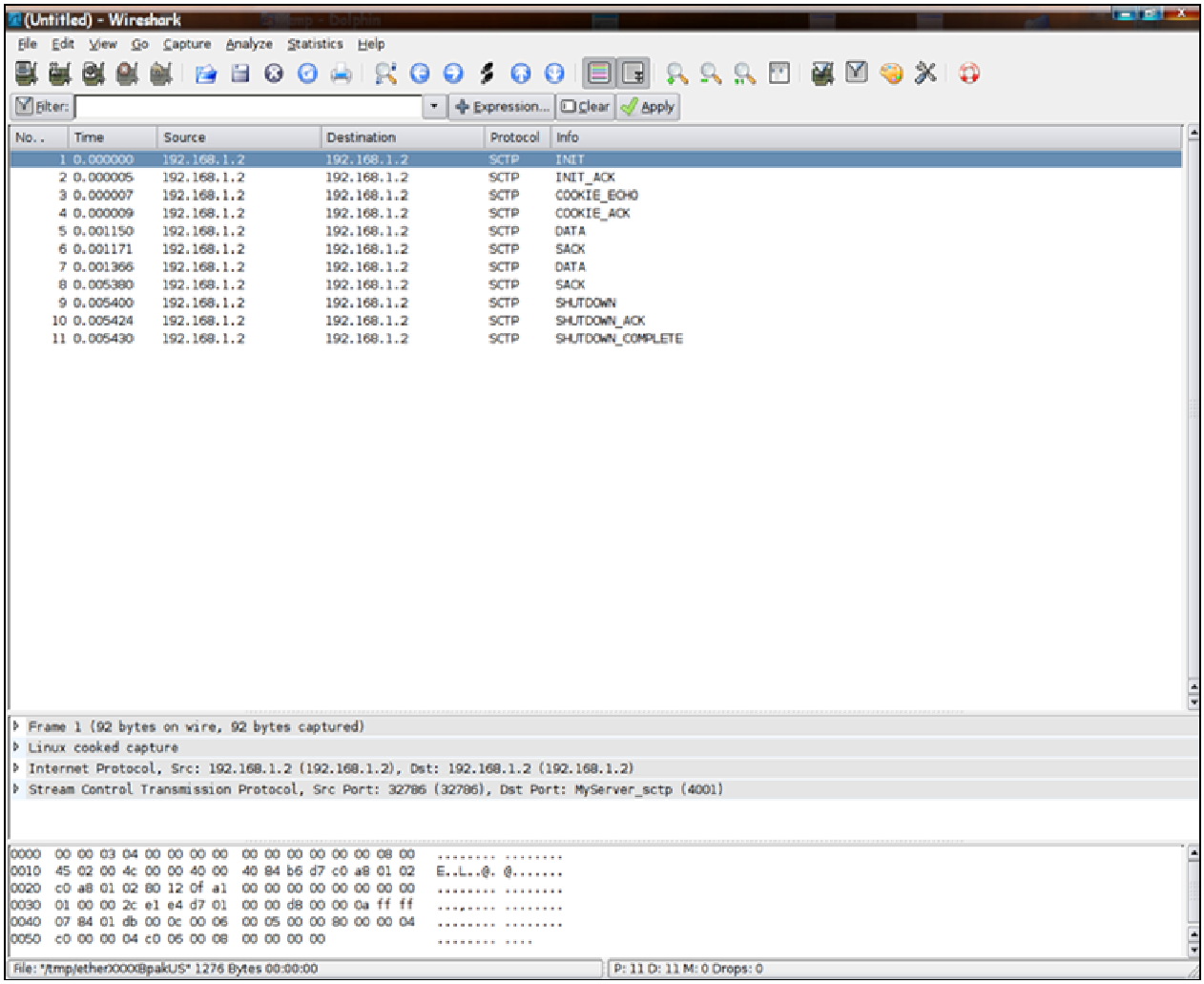
```
geobros@geobros:~/Temp$ ./MyClient_sctp 127.0.0.1 4001
clientName: 127.0.0.1
message:
sctp server_test!!!
 received from server 127.0.0.1

sending packets to the server 127.0.0.1
message:
client_to_server test!
 send to server
```

Σχήμα 46

Στο σχήμα 46 βλέπουμε τα αποτελέσματα. Ο client συνδέεται με τον server με sctp σύνδεση και αφού λάβει από τον server το μήνυμα “sctp server test!!”, στέλνει σ’αυτόν το μήνυμα “client to server test!”.

Για να επιβεβαιώσουμε ότι η μεταφορά έγινε με sctp σύνδεση, τρέχουμε το πρόγραμμα wireshark/ethereal το οποίο μας δείχνει την κίνηση και το είδος των πακέτων που μεταφέρονται στο Interface μας (σχήμα 47):



Σχήμα 47

Βλέπουμε λοιπόν μια SCTP συσχέτιση (INIT / COOKIE - 4way handshake) μια μεταφορά δεδομένων (DATA / SACK - αποστολή και λήψη) και τον τερματισμό της συσχέτισης (SHUTDOWN).

Με αυτόν τον τρόπο λοιπόν, διαπιστώνουμε ότι η τροποποιημένη έκδοση του inetd λειτουργεί, έτσι λοιπόν βλέπουμε και στην πράξη ότι το inetd εξυπηρετεί τόσο τις udp και tcp συνδέσεις, όσο και τις sctp συνδέσεις.

4. FTP.PROXY και WU-FTPD.

4.1 Περιγραφή του FTP πρωτοκόλλου

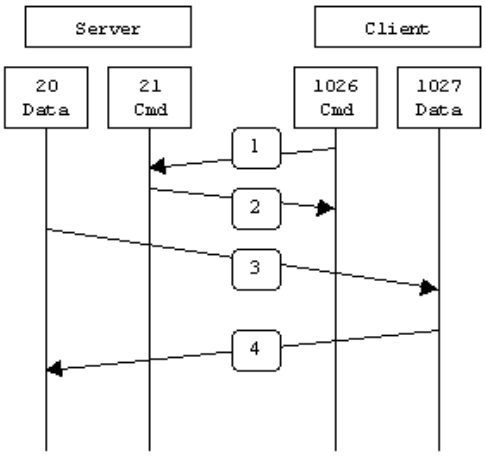
Πριν περιγράψουμε το ftp.proxy, είναι σημαντικό να ξέρουμε τι είναι το FTP. Το **FTP (File Transfer Protocol)** είναι λοιπόν το πρωτόκολλο μεταφοράς αρχείων, το οποίο χρησιμοποιούν διάφορες εφαρμογές για την μεταφορά των αρχείων. Η υπηρεσία της μεταφοράς αρχείων βασίζεται στο πρωτόκολλο επικοινωνίας TCP/IP. Έτσι όταν ένας ftp πελάτης θέλει να στείλει και να λάβει αρχεία από έναν ftp εξυπηρετητή, τότε δημιουργείται μια tcp σύνδεση μεταξύ τους όπου τα δεδομένα μεταφέρονται μέσω του FTP πρωτοκόλλου.

Το FTP πρωτόκολλο , για τη σύνδεση των δύο άκρων (client και server) ορίζει ένα **FTP Session (σύνοδος)**. Ένα FTP session αποτελείται από μία **σύνδεση ελέγχου (control connection)** και από μία ή περισσότερες **συνδέσεις δεδομένων (data connection)**.

- Η σύνδεση ελέγχου χρησιμοποιείται για τη μεταφορά των ftp εντολών και απαντήσεων (σε ASCII format).
- Για τη μεταφορά ενός αρχείου (ή ενός καταλόγου) χρησιμοποιείται μία σύνδεση δεδομένων. Έτσι ο αριθμός των συνδέσεων δεδομένων σε μια FTP σύνοδο είναι ίσος με τον αριθμό των μεταφορών αρχείων που πραγματοποιήθηκαν.

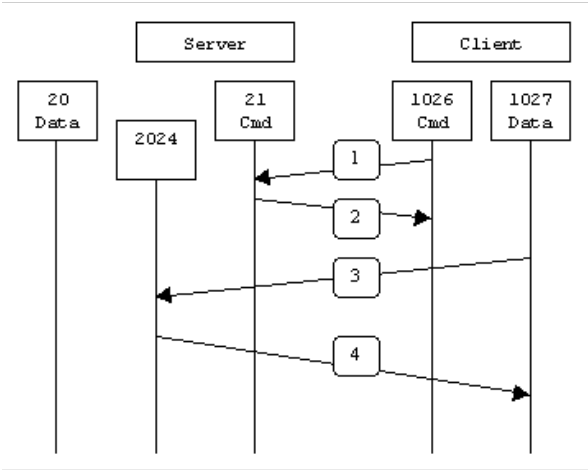
Η κάθε σύνδεση δεδομένων λειτουργεί με δύο τρόπους, είτε ως ενεργητική **(active)** είτε ων **παθητική (passive)**.

- Στην ενεργητική λειτουργία (active mode) ο FTP client συνδέεται από μια τυχαία θύρα (N>1023) στη θύρα 21 (command port) του ftp server. Έπειτα ο client (με την εντολή PORT) στέλνει τον αριθμό της θύρας (N+1), στην οποία επιθυμεί να "ακούει" (listen) για εισερχόμενες συνδέσεις. Με τη σειρά του ο FTP server δημιουργεί μια σύνδεση από την θύρα 20 (data port) στην ανοιχτή θύρα (N+1) του client για τη μεταφορά των δεδομένων (σχήμα 48). Κάθε φορά που ο client ζητάει δεδομένα, δημιουργείται κατά παρόμοιο τρόπο μια σύνδεση δεδομένων και η διαδικασία επαναλαμβάνεται.



Σχήμα 48

- Στην παθητική λειτουργία (passive mode) ο FTP client ζητά από τον server να διαλέξει μια τυχαία θύρα (P>1023), στην οποία θα "ακούει" (listen) για την σύνδεση δεδομένων (data connection). Ο server ενημερώνει τον client για την θύρα την οποία έχει διαλέξει και ο client συνδέεται σε αυτή για τη μεταφορά των δεδομένων (σχήμα 49). Η μεταφορά ολοκληρώνεται όπως και στην ενεργητική λειτουργία.



Σχήμα 49

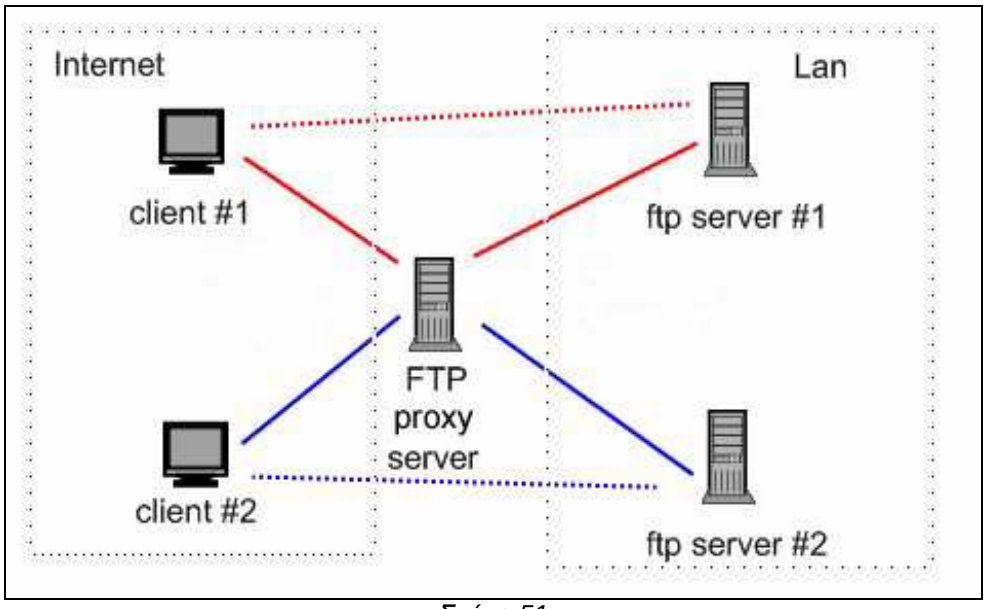
Στο σχήμα 50 βλέπουμε έναν ενδεικτικό διάλογο μεταξύ ftp client και ftp server:

```
geobros@geobros:~$ ftp 192.168.1.2 5000
Connected to 192.168.1.2.
220 geobros FTP server (Version wu-2.6.2(1) Fri Sep 12 18:05:37 EEST 2008) ready
.
Name (192.168.1.2:geobros): geobros
331 Password required for geobros.
Password:
230 User geobros logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> passive
Passive mode on.
ftp> cd ftptest/
250 CWD command successful.
ftp> ls
227 Entering Passive Mode (192,168,1,2,204,223)
150 Opening ASCII mode data connection for /bin/ls.
total 19344
-rw-r--r-- 1 geobros geobros    52549 2008-02-26 22:38 1.jpg
-rw-r--r-- 1 geobros geobros   239060 2008-09-15 02:24 aaaaaa!
-rw-r--r-- 1 geobros geobros 18874942 2008-10-20 14:01 Archive.tar.gz
-rw-r--r-- 1 geobros geobros   598016 2008-03-02 16:35 parousiasi.ppt
226 Transfer complete.
ftp> get 1
1      1.jpg
ftp> get 1.jpg
local: 1.jpg remote: 1.jpg
227 Entering Passive Mode (192,168,1,2,122,182)
150 Opening BINARY mode data connection for 1.jpg (52549 bytes).
226 Transfer complete.
52549 bytes received in 0.00 secs (314830.6 kB/s)
ftp> quit
221-You have transferred 52549 bytes in 1 files.
221-Total traffic for this session was 53427 bytes in 2 transfers.
221-Thank you for using the FTP service on geobros.
221 Goodbye.
```

Σχήμα 50

4.2 Περιγραφή του FTP PROXY

Σ'αυτό το κεφάλαιο θα περιγράψουμε την λειτουργία του **FTP PROXY SERVER**. Με τον όρο proxy εννοούμε την διαδικασία της μεσολάβησης. Συγκεκριμένα ένας proxy server(διακομιστής μεσολάβησης ή ενδιάμεσος διακομιστής) παρεμβάλλεται μεταξύ των πελατών και των εξυπηρετητών, και όταν λάβει κάποια αίτηση από ένας συγκεκριμένο πελάτη, μεταδίδει την αίτηση στον ανάλογο διακομιστή. Έτσι ο πελάτης αντί να επικοινωνήσει απευθείας με το server, επικοινωνεί με τον proxy, στέλνει τις αιτήσεις σ'αυτόν και στη συνέχεια ο proxy απευθύνεται στον server όπου ζητάει τα αποτελέσματα της αίτησης του client. Στη συνέχεια επιστρέφει στον client τα αποτελέσματα. Ο proxy server είναι και μέρος ενός firewall και μπορεί να αποτρέπει τους χάκερς να χρησιμοποιήσουν το Διαδίκτυο για να αποκτήσουν πρόσβαση σε υπολογιστές ενός ιδιωτικού δικτύου. Ο FTP PROXY λειτουργεί με αυτή τη λογική για ftp συνδέσεις. Ένας FTP Proxy φαίνεται στο σχήμα 51.



Σχήμα 51

Η εφαρμογή ftp proxy που αναλύουμε παρακάτω, είναι το open-source πρόγραμμα **ftp.proxy.1.2.3** το οποίο βρίσκεται στην διεύθυνση: http://www.ftpproxy.org/

Το ftp.proxy λειτουργεί σε Λ.Συστήματα unix και καλείται μέσω του inetd daemon

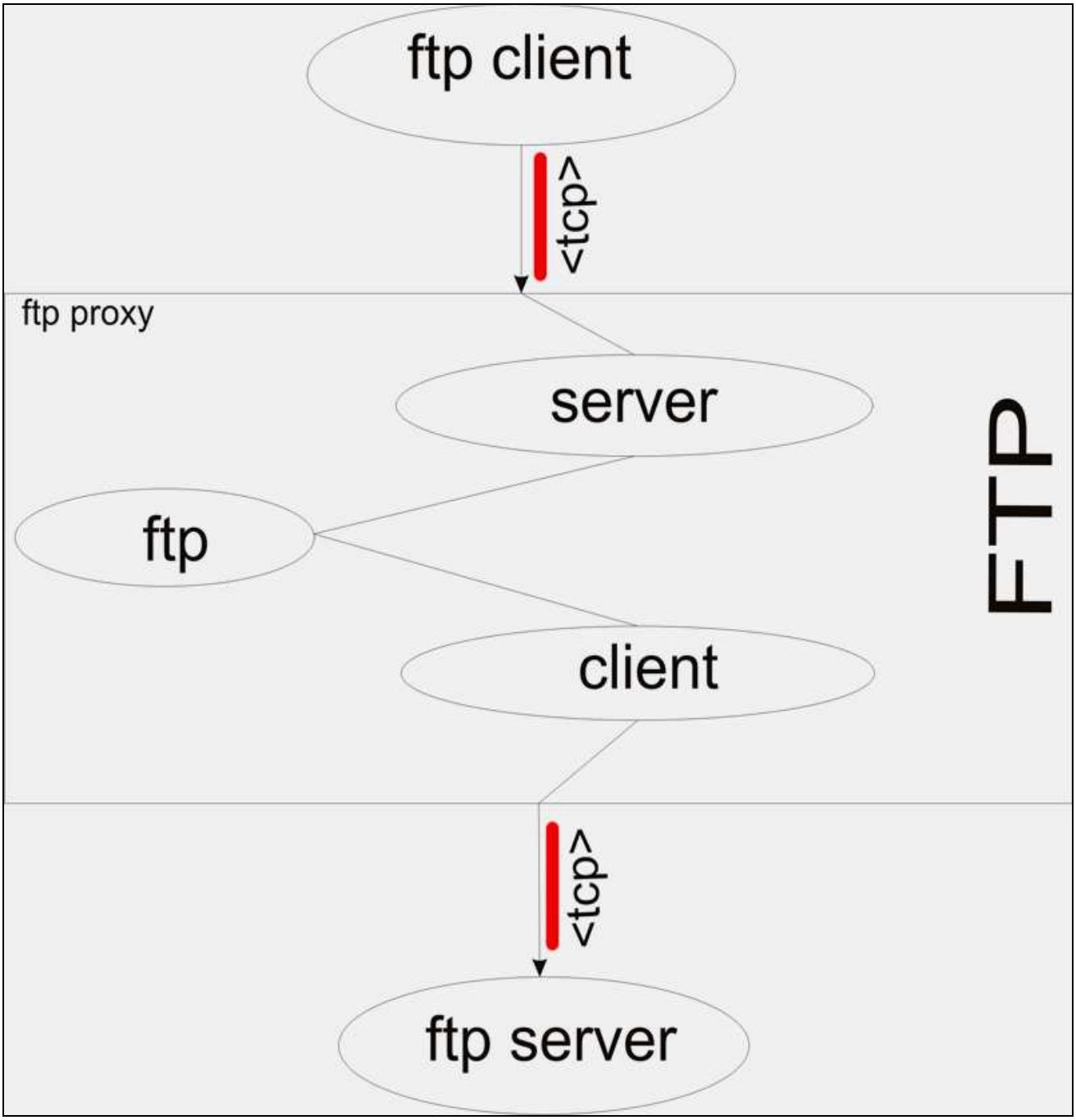
4.2.1 Ανάλυση Υπάρχοντος Κώδικα ftp.proxy server

Ο κώδικας του ftp proxy.1.2.3 κάνει χρήση του Socket API και λειτουργεί ως εξής: Ορίζονται ένας server και ένας client, ο server για τη σύνδεση με τον ftp client και ο client για τη σύνδεση με τον ftp server. Έτσι έχουμε δύο sockets, ένα για την επικοινωνία του ftp proxy με τον ftp client και ένα για την επικοινωνία του ftp proxy με τον ftp server.

Με τη μέθοδο bind_to_port ορίζεται ο server ο οποίος θα ακούει τις αιτήσεις του ftp client. Ορίζεται το socket σε TCP σύνδεση, ακολουθεί η κλήση συστήματος bind και listen όπου από εκεί και μετά αυτός ο server θα ακούει τις αιτήσεις ftp του ftp client. Όμοια με τη μέθοδο openip ορίζεται ο client ο οποίος θα συνδέεται με τον ftp server. Ορίζεται το socket (σε TCP σύνδεση), ορίζεται το Port και το protocol family του client, στη συνέχεια έχουμε την κλήση της bind (στο Port που ορίστηκε πριν), και τέλος μέσω της κλήσης συστήματος connect συνδέεται ο client αυτός με τον ftp server.

Αυτό που γίνεται ουσιαστικά, είναι σύνδεση του ftp client με τον server της bind_to_port που περιγράψαμε πριν (μέσω της accept) η μεταφορά των ftp αιτημάτων (και εντολών) από τον ftp client στον server του ftp.proxy , και στη συνέχεια η μεταφορά τους στον client (της μεθόδου openip) ο οποίος θα τα μεταφέρει με την σειρά του στον ftp server.

Ο λειτουργία του κώδικα του ftp proxy φαίνεται στο σχήμα 52.


ftp client
<tcp>
ftp proxy
server
ftp
client
FTP
<tcp>
ftp server

Σχήμα 52

4.2.2 Τροποποίηση Κώδικα ftp.proxy

Όπως και με το inetd, έτσι και εδώ θα τροποποιήσουμε τον κώδικα του ftp proxy ώστε αυτός να δέχεται tcp σύνδεση και να τη μετατρέπει σε sctp σύνδεση.

Από την προηγούμενη ανάλυση κώδικα βοηθηθήκαμε ώστε να καταλάβουμε καλύτερα το πώς ακριβώς δουλεύει ο ftp.proxy1,2,3.

Το σημαντικότερο σημείο του κώδικα για την μετατροπή της σύνδεσης από tcp σε sctp είναι ο client που δημιουργείται από τη μέθοδο **openip** για την σύνδεση του ftp.proxy με τον ftp server. Στη μέθοδο αυτήν λοιπόν ορίζεται το socket και ο τύπος του socket για τον client. Δεδομένου ότι στην επόμενη παράγραφο θα τροποποιήσουμε έναν ftp server (τον wu-ftpd) ώστε να λειτουργεί με το sctp πρωτόκολλο, μπορούμε να τροποποιήσουμε τον client αυτόν ώστε να λειτουργεί κι αυτός με το sctp πρωτόκολλο (αντί του tcp) έτσι ώστε η σύνδεση του (και επομένως η σύνδεση του ftp.proxy) με τον ftp server να είναι μια sctp συσχέτιση. Θα χρειαστεί να κάνουμε χρήση του SCTP API ώστε να υποστηρίζει το ftp.proxy το sctp πρωτόκολλο. Αφού καλέσουμε λοιπόν την κατάλληλη βιβλιοθήκη (sctp.h) τροποποιούμε τον τύπο του socket του client της μεθόδου openip ορίζοντας το socket σε sctp πρωτόκολλο.

```
socketd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
```

Με την εντολή αυτή όπως είπαμε ορίζουμε το socket (υποδοχή). Συγκεκριμένα ορίζουμε την οικογένεια των πρωτοκόλλων που θα ανήκει η υποδοχή μας, το είδος της υπηρεσίας που θα παρέχεται μέσω της παρούσας υποδοχής, και το πρωτόκολλο που θα χρησιμοποιείται από την υποδοχή. Εδώ, χρησιμοποιούμε την οικογένεια πρωτοκόλλων AF_INET (Address Format, Internet). Η οικογένεια αυτή ορίζει το internet domain και έτσι για κάθε socket ορίζεται μια IP διεύθυνση και ένα PORT. Άλλες οικογένειες πρωτοκόλλων είναι η PF_INET (Packet Format, Internet - για τη στοίβα TCP/IP) και η AF_UNIX (Address Format, Unix - Unix Domain διευθύνσεις). Το είδος της υπηρεσίας που θα παρέχεται εδώ είναι η SOCK_STREAM (εγκατάσταση κυκλώματος). Τα πρωτόκολλα TCP και SCTP όπως είπαμε απαιτούν τη σύνδεση των δύο κόμβων πριν αρχίσει η μεταφορά δεδομένων μεταξύ τους. Αντίθετα, το πρωτόκολλο UDP δεν απαιτεί κάτι τέτοιο. Στην περίπτωση του UDP η υπηρεσία θα ήταν η “SOCK_DGRAM”. Εδώ, εφόσον έχουμε SCTP πρωτόκολλο, η υπηρεσία SOCK_STREAM είναι ανάλογη με αυτήν του TCP. Οπότε μιλάμε για σύνδεση 1 προς 1. Για τη σύνδεση ένα προς πολλά (που αναφέραμε στη παράγραφο 2,5) χρησιμοποιούμε την υπηρεσία “SOCK_SEQPACKET”. Τέλος με τον τύπο IPPROTO_SCTP ορίζουμε το SCTP πρωτόκολλο στην υποδοχή μας.

Τέλος, αν λάβουμε υπ'όψη μας την περίπτωση όπου μια διεργασία κλείσει μια θύρα, τότε αυτή δε μπορεί να ξαναχρησιμοποιηθεί (να ξανασυνδεθεί - rebind) για κάποιο χρονικό διάστημα (συνήθως 2 sec), εξαιτίας κάποιων λειτουργιών του socket που παραμένουν στον πυρήνα για κάποιο λόγο. Αυτό το πρόβλημα μπορεί να λυθεί με την παράμετρο "*setsockopt (socketd, SOL_SOCKET, SO_REUSEADDR, (int *) &one, sizeof (one))*".

```
setsockopt (socketd, SOL_SOCKET, SO_REUSEADDR, (int *)
&one, sizeof (one));
```

Με την τροποποίηση του κώδικα του ftp.proxy, τροποποιήθηκαν και διάφορα άλλα απαραίτητα αρχεία, όπως για παράδειγμα το αρχείο makefile στο οποίο δηλώθηκε το κατάλληλο flag ώστε να γίνεται χρήση της βιβλιοθήκης sctp κατά το compiling (-lsctp).

4.3 Περιγραφή του wu-ftpd

Ο **wu-ftpd** είναι ένας open-source unix ftp server. Βασίζεται (όπως όλοι οι ftp servers) πάνω στο TCP πρωτόκολλο.

4.3.1 Ανάλυση Υπάρχοντος Κώδικα wu-ftpd server

Ο κώδικας του wu-ftpd, αρχικά , μέσω του socket api ορίζει δύο tcp sockets. Ένα socket τη σύνδεση ελέγχου (control connection) όπου όπως είπαμε θα χρησιμοποιείται από το FTP για τη μεταφορά των εντολών ανάμεσα στον client και στον server, και ένα socket για τη σύνδεση δεδομένων (data connection) όπου χρησιμοποιείται για την μεταφορά των δεδομένων.

Αφού οριστεί λοιπόν το κατάλληλο port μέσω της bind για κάθε socket, καλείται η listen με την οποία ο server περιμένει αιτήσεις για συνδέσεις ελέγχου και δεδομένων από κάποιον client. Όταν λοιπόν ο server «ακούσει» κάποια αίτηση στο port του socket το οποίο ορίζει τη σύνδεση ελέγχου, συνδέεται με τον αντίστοιχο client με σκοπό τη μεταφορά ftp εντολών, ενώ αν ο server «ακούσει» κάποια αίτηση από το port του socket δεδομένων, τότε ορίζει μια σύνδεση δεδομένων με τον αντίστοιχο client στην οποία μεταφέρονται δεδομένα (αρχεία ή κατάλογοι). Ο ftp server, κάθε φορά που ακούει μια νέα αίτηση για μεταφορά δεδομένων εκτελεί τη μέθοδο **Select()**. Η μέθοδος αυτή «περιμένει» μέχρι ένας socket descriptor να είναι διαθέσιμος για εγγραφή ή ανάγνωση (readable/writable). Η δημιουργία των socket αυτών γίνεται στο αρχείο ftpd.c.

4.3.2 Τροποποίηση Κώδικα wu-ftpd server

Αφού φορτώσουμε την sctp βιβλιοθήκη για το sctp api, ορίζουμε τόσο το socket για το control connection όσο και το socket για το data connection να χρησιμοποιούν το πρωτόκολλο SCTP. Εδώ γνωρίζουμε εκ των προτέρων ότι με το να ορίσουμε το πρωτόκολλο μεταφοράς SCTP και στο data connection αλλά και στο control connection αυτομάτως παρέχουμε επιπλέον ασφάλεια στις συνδέσεις αυτές μιας και δεν υπάρχει πλέον ο κίνδυνος για DoS Attacks.

Για την τροποποίηση λοιπόν του server ώστε να δέχεται sctp συνδέσεις, γράφουμε τον παρακάτω κώδικα στο τμήμα όπου ορίζεται η σύνδεση δεδομένων καί η σύνδεση ελέγχου:

```
s = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
//change to sctp socket
    if (s < 0)
     goto bad;
    if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR,
              (char *) &on, sizeof(on)) < 0)
     goto bad;
```

Τις εντολές αυτές τις περιγράψαμε στην προηγούμενη παράγραφο.

Με την τροποποίηση του κώδικα του wu-ftpd, τροποποιήθηκαν και διάφορα άλλα απαραίτητα αρχεία, όπως για παράδειγμα το αρχείο makefile στο οποίο δηλώθηκε το κατάλληλο flag ώστε να γίνεται χρήση της βιβλιοθήκης sctp κατά το compiling (-lsctp).

4.4 Εκτέλεση εφαρμογών / απλές μετρήσεις

4.4.1 Οι εφαρμογές που χρησιμοποιήθηκαν

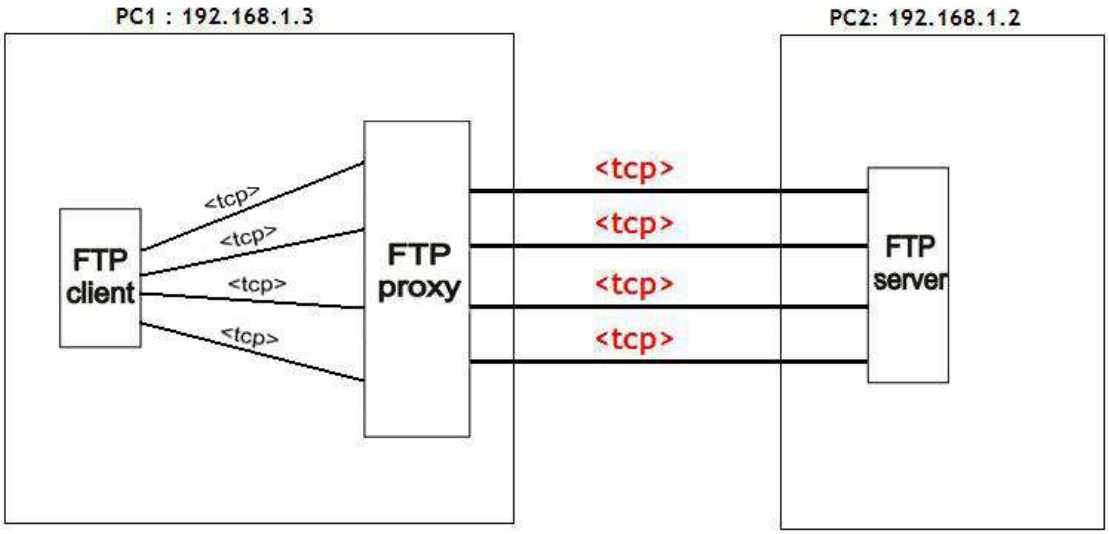
Προχωράμε στο πειραματικό μέρος της μελέτης μας και ρυθμίζουμε κατάλληλα το σύστημά μας, εκτελώντας τις εφαρμογές:

gftp	Linux ftp client με γραφικό περιβάλλον (open-source)
ftp.proxy	Ο ftp proxy που περιγράψαμε στην παράγραφο 4,2
wu-ftpd	Ο ftp server που περιγράψαμε στην παράγραφο 4,3

Θα τρέξουμε τις εφαρμογές αυτές για να συγκρίνουμε τα δύο πρωτόκολλα στην πράξη. Για το TCP πρωτόκολλο θα τρέξουμε τις κανονικές εφαρμογές ενώ για το SCTP τις τροποποιημένες εφαρμογές που περιγράψαμε στις 2 προηγούμενες παραγράφους. Και στις δύο περιπτώσεις, η σύνδεση του ftp client με τον ftp proxy θα γίνεται με TCP πρωτόκολλο. Για τον λόγο αυτό, αλλά και για να έχουμε όσο το δυνατών έγκυρα αποτελέσματα, χωρίς απώλειες (κυρίως για το πρωτόκολλο SCTP) Θα τρέξουμε τον ftp proxy στο ίδιο μηχάνημα που βρίσκεται και το πρόγραμμα του ftp client.

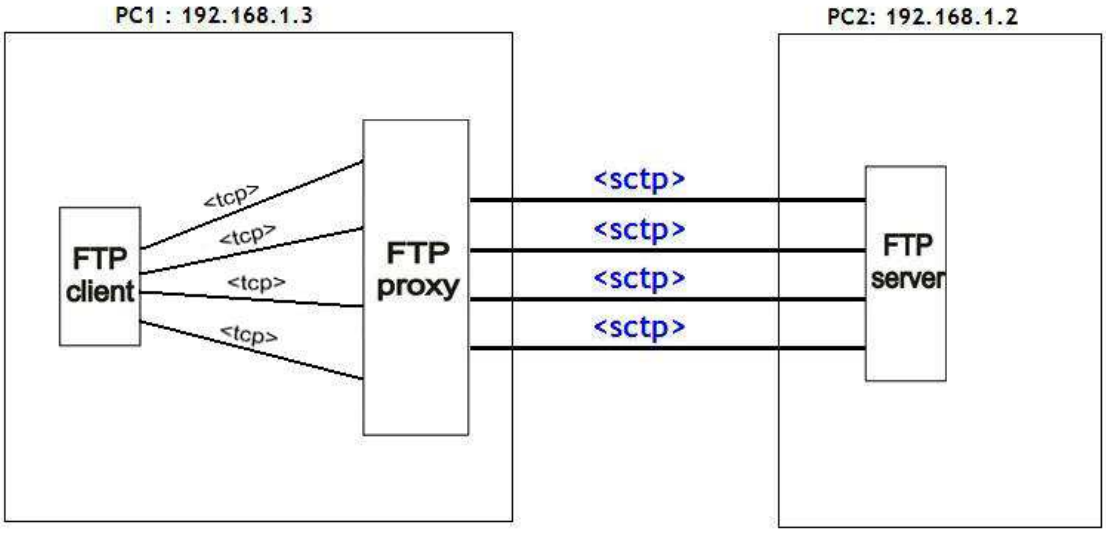
Το (τοπικό) δίκτυο στο οποίο κάναμε τις μετρήσεις αποτελείται από 2 υπολογιστές και ένα switch. Ο ftp proxy και ο ftp client τρέχουν στο PC με IP 192.168.1.3 ενώ ο ftp server στο PC me IP 192.168.1.2. Ουσιαστικά ο gftp και ο ftp.proxy «τρέχουν» στο ίδιο μηχάνημα.

Έτσι για το πρωτόκολλο TCP έχουμε το σχήμα 53.



Σχήμα 53

Ενώ για το πρωτόκολλο SCTP έχουμε το σχήμα 53.



Σχήμα 54

4.4.2 Δοκιμή λειτουργίας ftp.proxy και wu-ftpd μέσω inetd, με TCP και SCTP πρωτόκολλο

Αρχικά για το πρωτόκολλο TCP, εγκαθιστούμε στο σύστημα μας την κανονική-tcp έκδοση των ftp.proxy και wu-ftpd και ορίζουμε τα σχετικά ports και services αρχείο **/etc/services** :

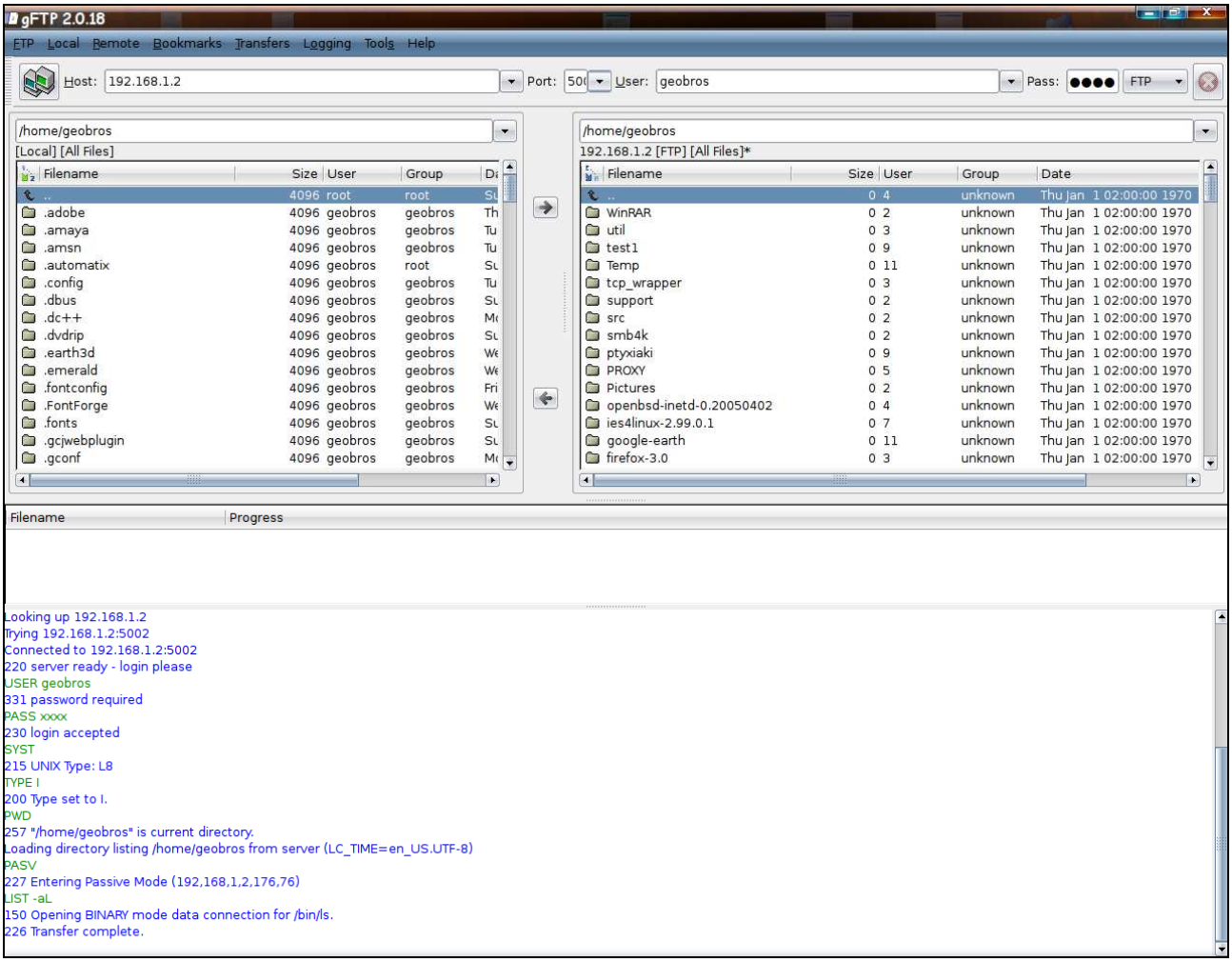
```
ftpproxy        5002/tcp
wu-ftpd         5000/tcp
```

Τέλος ορίζουμε τα service στο αρχείο **/etc/inetd.conf** ως:

```
ftpproxy  stream  tcp  nowait  nobody  /usr/local/sbin/ftp.proxy -b 192.168.1.2:5000
wu-ftpd   stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/wu-ftpd –l
```

Αφού κάνουμε restart τον inetd server, τρέχουμε την εφαρμογή gftp ορίζοντας την ip και το port της ftp σύνδεσης με αυτό του ftp proxy. Αυτός με την σειρά του θα συνδεθεί με τον ftp server μας στην διεύθυνση που ορίσαμε στο αρχείο inetd.conf.

Τρέχουμε λοιπόν τον gftp και έτσι επιτυγχάνεται η σύνδεσή μας (TCP) με τον ftp server, όπως βλέπουμε και από το σχήμα 55.



Σχήμα 55

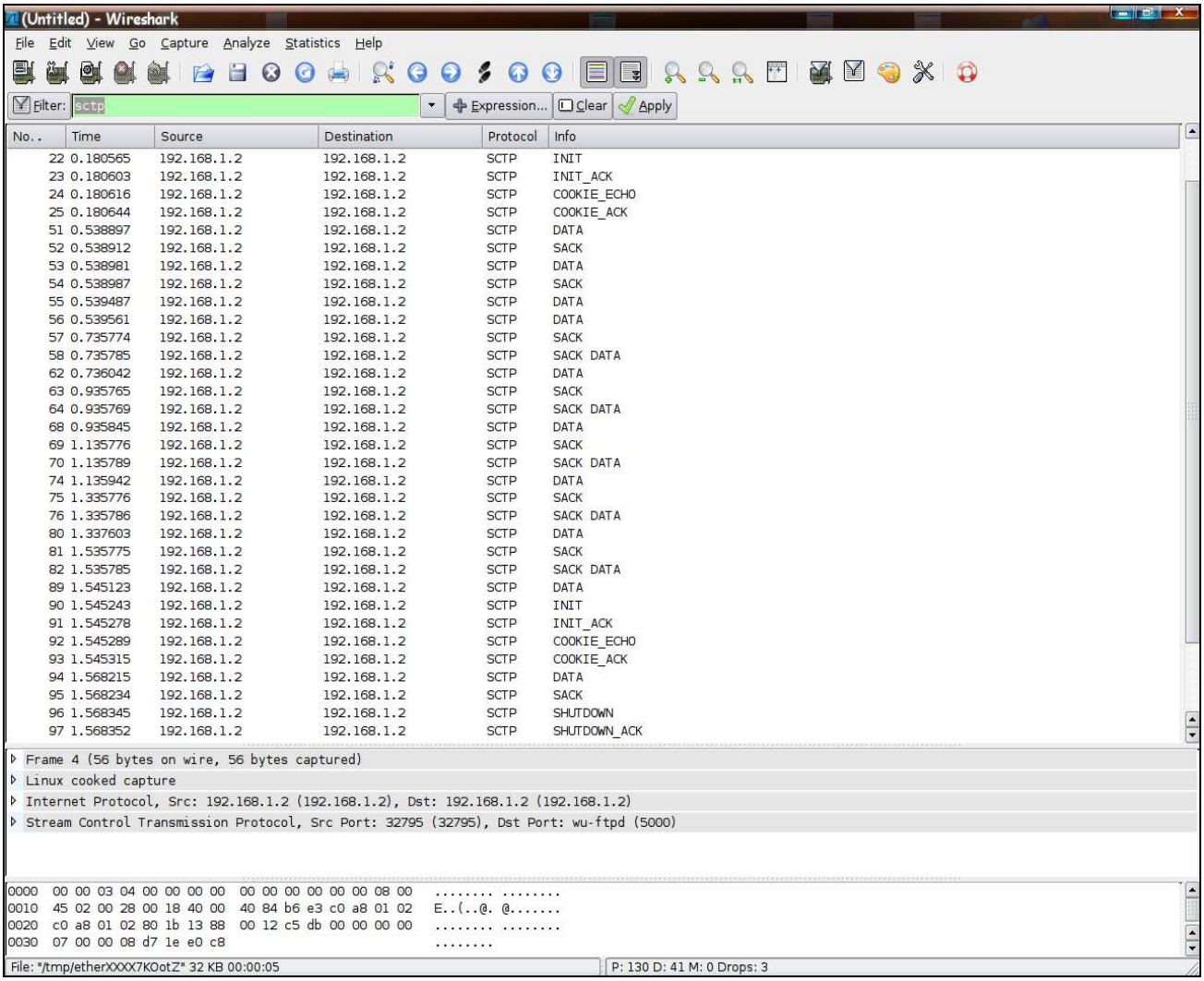
Για την σύνδεση με το πρωτόκολλο SCTP, εγκαθιστούμε τις τροποποιημένες εφαρμογές που περιγράψαμε πριν και ρυθμίζουμε κατάλληλα το αρχείο **/etc/services** :

```
ftpproxy        5002/tcp
wu-ftpd         5000/sctp
```

και το αρχείο **/etc/inetd.conf** :

```
ftpproxy  stream  tcp   nowait  nobody  /usr/local/sbin/ftp.proxy -b 192.168.1.2:5000
wu-ftpd   stream  sctp  nowait  root    /usr/sbin/wu-ftpd –l
```

Τρέχουμε τον gftp client, και συνδεόμαστε άμεσα με τον ftp proxy όπου όπως είπαμε αυτός θα μας συνδέσει με τον ftp server(wu-ftpd) με sctp σύνδεση. Αφού λοιπόν συνδεθούμε, τρέχουμε το wireshark όπου και διαπιστώνουμε ότι η σύνδεση του ftp.proxy με τον wu-ftpd γίνεται με το πρωτόκολλο sctp (σχήμα 56).



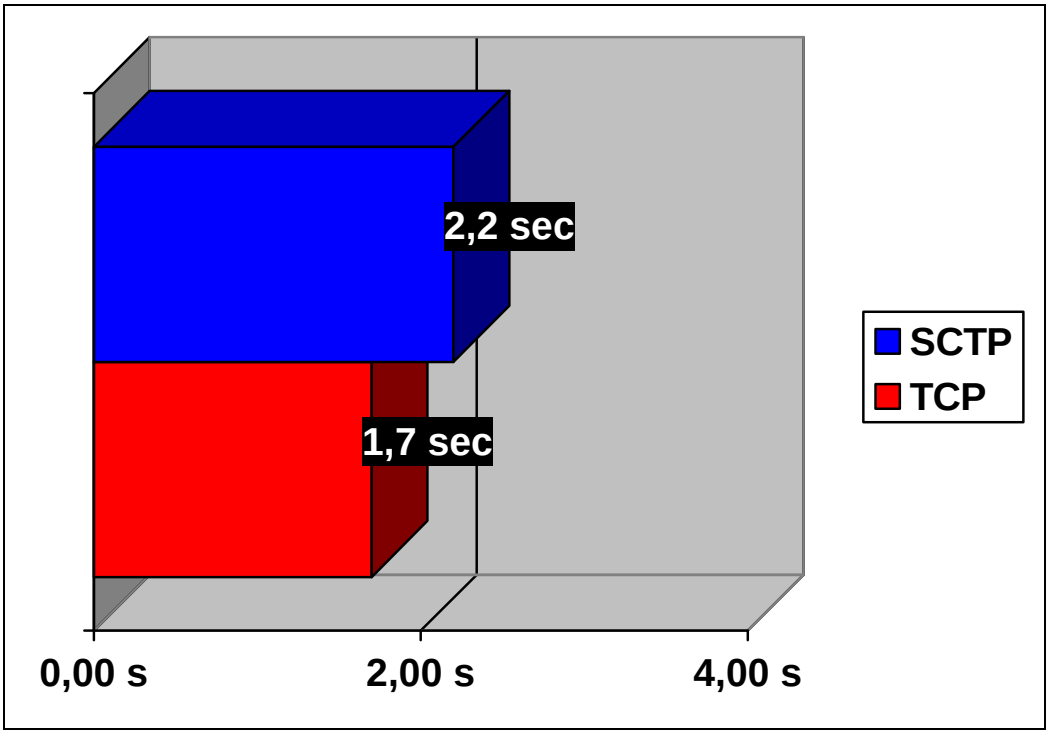
Σχήμα 56

4.4.3 Παραδείγματα

*Σε όλα τα παραδείγματα μετρήθηκαν οι χρόνοι (σε δευτερόλεπτα) με το πρόγραμμα **Wireshark**.*

Παράδειγμα 1 : ΣΥΝΔΕΣΗ

Στο παράδειγμα αυτό μετρήσαμε τον χρόνο που χρειάζεται για να συνδεθούμε με τον ftp server (σχήμα 57).

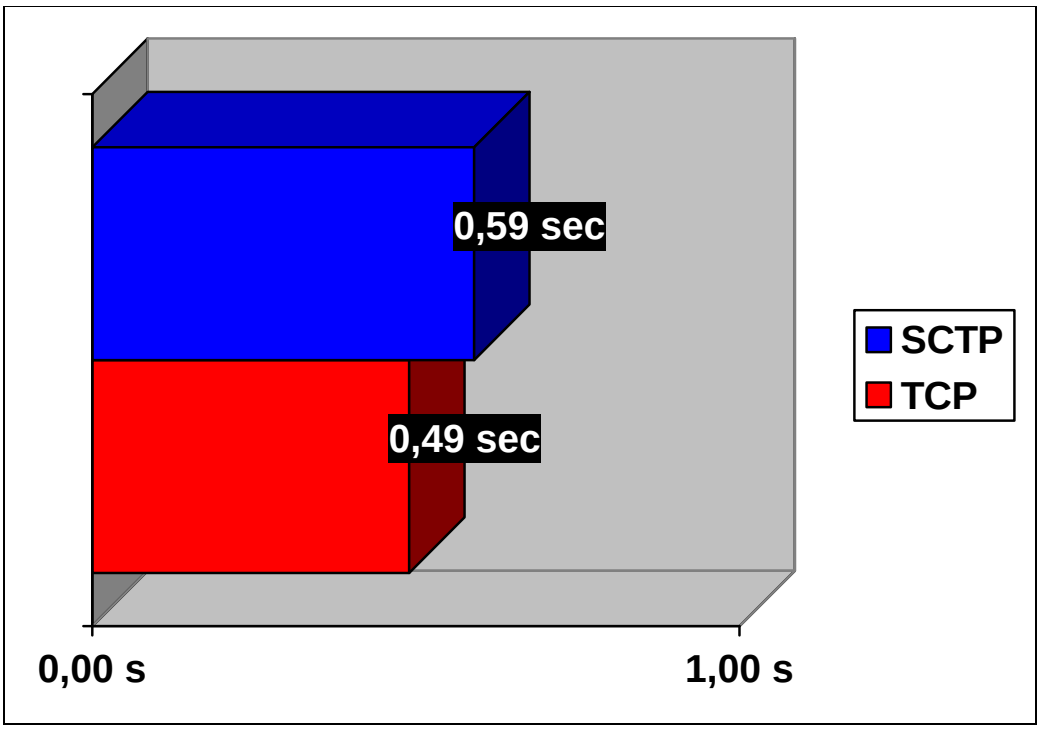


Σχήμα 57

Όπως βλέπουμε σχήμα 57 ο χρόνος σύνδεσης του ftp client με τον ftp server (μέσω του ftp proxy) είναι μικρότερος όταν χρησιμοποιείται το TCP πρωτόκολλο.

Παράδειγμα 2 : μεταφορά αρχείου μικρού μεγέθους

Στο παράδειγμα αυτό μετρήσαμε τον χρόνο που χρειάζεται για τη μεταφορά ενός αρχείου μεγέθους 2,8Kbytes από τον server στον client (σχήμα 58).

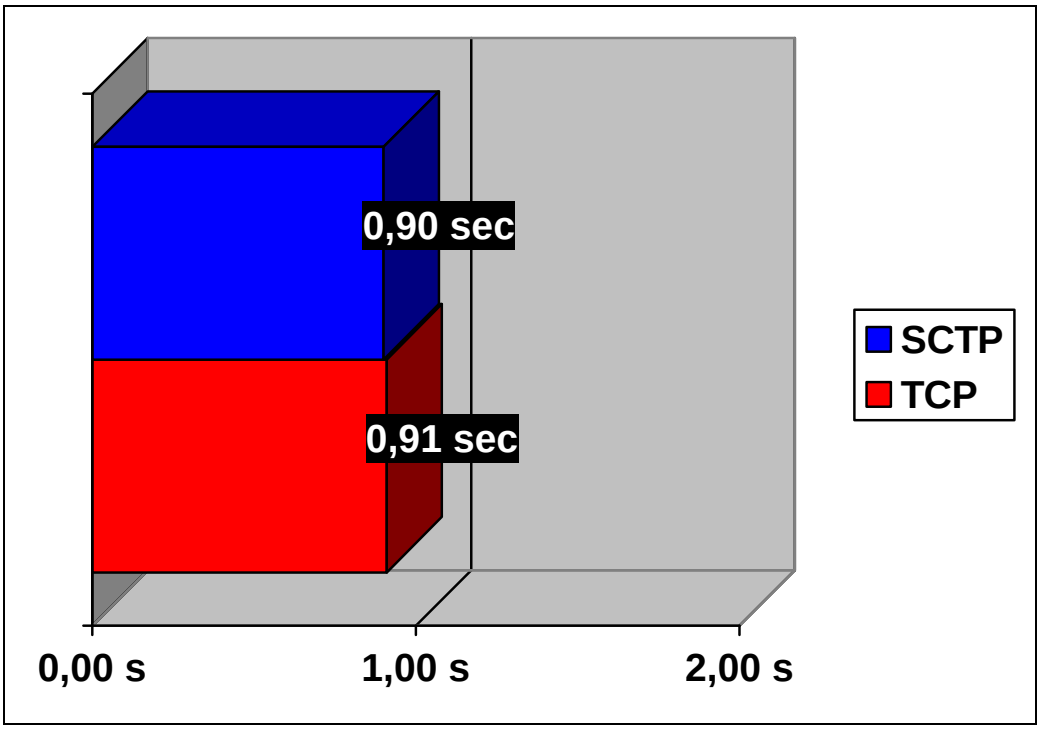


Σχήμα 58

Από το σχήμα 58 παρατηρούμε ότι ο χρόνος για τη μεταφορά ενός αρχείου μεγέθους 2,8KB από τον ftp server στον ftp client (μέσω του ftp proxy) είναι μικρότερος όταν χρησιμοποιείται το TCP πρωτόκολλο.

Παράδειγμα 3 : μεταφορά αρχείου μεσαίου μεγέθους

Στο παράδειγμα αυτό μετρήσαμε τον χρόνο που χρειάζεται για τη μεταφορά ενός αρχείου μεγέθους 9Mbytes από τον server στον client (σχήμα 59).

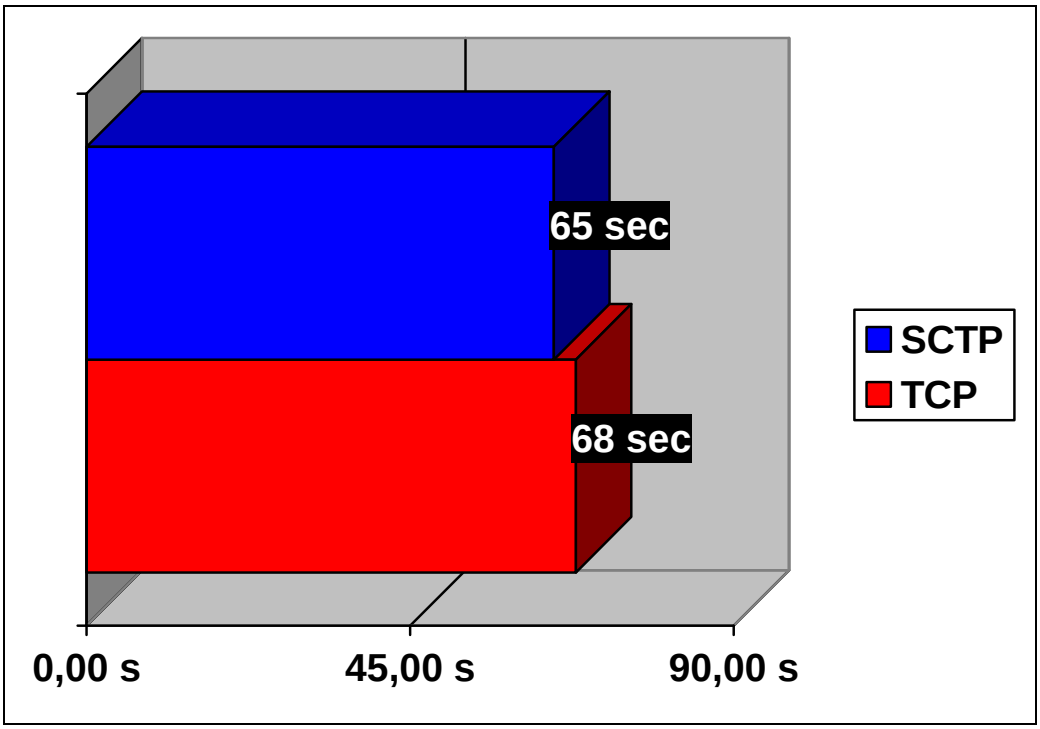


Σχήμα 59

Όπως βλέπουμε και από το σχήμα 59, και εδώ, ο χρόνος για τη μεταφορά ενός αρχείου μεγέθους 9MB από τον ftp server στον ftp client (μέσω του ftp proxy) είναι ίδιος και για τα δύο πρωτόκολλα.

Παράδειγμα 4 : μεταφορά αρχείου μεγάλου μεγέθους

Στο παράδειγμα αυτό μετρήσαμε τον χρόνο που χρειάζεται για τη μεταφορά ενός αρχείου μεγέθους 741MBytes από τον server στον client (σχήμα 60).



Σχήμα 60

Παρατηρούμε ότι για τα μεγάλα αρχεία μεγέθους 741MB, ο χρόνος μεταφοράς (από τον ftp server στον ftp client) είναι μικρότερος όταν χρησιμοποιείται το SCTP πρωτόκολλο.

4.4.3 Διάφορες παρατηρήσεις σχετικά με τα αποτελέσματα των παραδειγμάτων

- Στο παράδειγμα 1 παρατηρήσαμε ότι ο χρόνος σύνδεσης του client με τον ftp server είναι μεγαλύτερος όταν χρησιμοποιείται το πρωτόκολλο SCTP. Αυτό οφείλεται στο γεγονός ότι η σύνδεση στο TCP πρωτόκολλο γίνεται με τη μέθοδο 3-way handshake ενώ η συσχέτιση στο SCTP γίνεται με τη μέθοδο του 4-way handshake. Αυτό σημαίνει ότι ανάμεσα στους κόμβους έχουμε την ανταλλαγή τεσσάρων μηνυμάτων (INIT, INIT_ACK, COOKIE_ECHO, COOKIE_ACK) αντί τριών στο tcp (SYN, SYN_ACK, ACK).

- Στο παράδειγμα 2 παρατηρήσαμε για τη μεταφορά ενός αρχείου πολύ μικρού μεγέθους (2,8KB), το TCP πρωτόκολλο είναι πιο αποτελεσματικό. Η εξήγηση γι'αυτό είναι η ίδια με το προηγούμενο παράδειγμα. Έτσι το SCTP πρωτόκολλο θα κάνει περισσότερο χρόνο να εγκαθιδρύσει μια σύνδεση (4-way handshake) με τον απέναντι κόμβο (ftp server). Μόλις οι δύο κόμβοι συνδεθούν, για μικρά αρχεία (μικρός αριθμός από PDU) ο χρόνος αποστολής/λήψης δεν είναι ιδιαίτερα μεγάλος. Μόλις η μεταφορά ολοκληρωθεί, οι δύο κόμβοι τερματίζουν την σύνδεση. Ο τερματισμός για το πρωτόκολλο SCTP γίνεται κι'αυτός με 4-way handshake (αντί της 3-way handshake του TCP). Άρα λοιπόν, συγκρίνοντας τους χρόνους αυτούς με τους χρόνους του προηγούμενου παραδείγματος παρατηρούμε ότι στην πραγματικότητα η «καλύτερη» επίδοση του TCP δεν σχετίζεται με την μεταφορά των δεδομένων (του 2,8KB αρχείου) αλλά με τον μεγαλύτερο χρόνο που απαιτεί το SCTP για να ολοκληρώσει μια σύνδεση μέσω του 4-way handshake. Λαμβάνοντας υπ'όψη μας και τα επόμενα παραδείγματα, θα διαπιστώσουμε ότι όσο το μέγεθος των αρχείων αυξάνεται, τόσο λιγότερη σημασία θα έχει η διαδικασία της σύνδεσης των κόμβων.

- Στο παράδειγμα 3 βλέπουμε ότι οι χρόνοι μεταφοράς και για τα δύο πρωτόκολλα είναι ίδιοι. Τα δύο πρωτόκολλα αποδίδουν το ίδιο για τη μεταφορά αρχείων τέτοιου μεγέθους.

- Στο παράδειγμα 4 βλέπουμε την υπεροχή του SCTP πρωτοκόλλου για τη μεταφορά μεγάλων αρχείων. Εδώ για τη μεταφορά αρχείου μεγέθους 741MB παρατηρούμε ότι το SCTP έχει καλύτερη επίδοση, και καταλαβαίνουμε ότι για τη μεταφορά δεδομένων τέτοιου μεγέθους οι χρόνοι σύνδεσης δε χρειάζεται να μας απασχολούν. Αυτό οφείλεται στο γεγονός ότι το SCTP χρησιμοποιεί πιο επιθετικό congestion control (παράγραφος 2,1,4,1) από το TCP. Στην περίπτωση λοιπόν που μεταφέρουμε μεγάλο όγκο δεδομένων, οι πιθανότητες για

απώλειες/επανεκπομπές των πακέτων αυξάνονται. Η καλύτερη λειτουργία του ελέγχου συμφόρησης του SCTP από αυτή του TCP οφείλεται κυρίως στις διαφορετικές υλοποιήσεις των μηχανισμών της γρήγορης αναμετάδοσης και της γρήγορης αποκατάστασης.

4.5 Προοπτικές επιπλέον ανάπτυξης

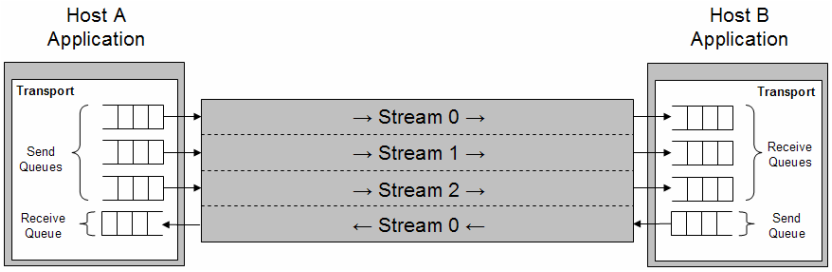
Η εφαρμογές μας αποτελούν τη βάση για την ανάπτυξη ενός συστήματος με επιπλέον δυνατότητες. Κάποιες απ' αυτές θα αναφέρουμε στην παράγραφο αυτή.

4.5.1 Multistreaming

Θα μπορούσαμε να παρέχουμε στις εφαρμογές μας (ftp.proxy και wu-ftpd) τη δυνατότητα υποστήριξης πολλών ροών δεδομένων.

Η τροποποίηση των εφαρμογών μπορεί να γίνει με τον εξής τρόπο:

Μπορούμε να τροποποιήσουμε τον ftp proxy και να ορίσουμε ένα ενιαίο sctp socket, τόσο για την σύνδεση ελέγχου όσο και για τη σύνδεση δεδομένων. Μ' αυτόν τον τρόπο θα έχουμε ένα ενιαίο association. Έπειτα ο proxy θα ανταλλάζει πληροφορίες ελέγχου από το stream 0, και πληροφορίες δεδομένων από το stream 1. Με τον ίδιο τρόπο μπορούμε να τροποποιήσουμε και τον ftp server ώστε να υπάρχει ένα ενιαίο socket όπου μέσα από ένα association (με τον proxy) θα ανταλλάζονται οι έλεγχοι από το stream 0 και τα δεδομένα από το stream 1. (σχήμα 61)



Σχήμα 61

Έτσι στην περίπτωση στην οποία θέλουμε να μεταφέρουμε πολλά αρχεία από τον server, ο ftp server θα λαμβάνει τις αιτήσεις (εντολές ελέγχου) από τον ftp proxy από το stream 0 και στη συνέχεια θα στέλνει τα δεδομένα στον proxy από το stream 1. Αυτή η διαδικασία θα γίνεται για κάθε αρχείο. Έτσι λοιπόν, ενώ ο ftp client (ο οποίος λειτουργεί πάνω στο TCP πρωτόκολλο) θα δημιουργεί μια μόνιμη σύνδεση ελέγχου και πολλές συνδέσεις δεδομένων (τόσες όσες και τα

αρχεία που πρέπει να μεταφερθούν) με τον ftp proxy, ο proxy θα συνδέεται με τον ftp server χρησιμοποιώντας μια ενιαία σύνδεση (association) , τόσο για τη μεταφορά των πληροφοριών ελέγχου όσο και των δεδομένων.

Ο τρόπος καθορισμού πολλαπλών ροών δεδομένων για το socket γίνεται σύμφωνα με το σχήμα 62.

```
struct sctp_initmsg initmsg;
struct    sctp_event_subscribe  evnts;  //events  struct  for
setsockopt - set events
struct    sctp_sndrcvinfo  *sri;
memset( &initmsg, 0, sizeof(initmsg) );
     initmsg.sinit_num_ostreams = 2;
     initmsg.sinit_max_instreams = 2;
     initmsg.sinit_max_attempts = 4;
bzero(&evnts, sizeof(evnts));
    evnts.sctp_data_io_event = 1;
    evnts.sctp_association_event=1;

s = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
//change to sctp socket
    if (s < 0)
     goto bad;
    if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR,
              (char *) &on, sizeof(on)) < 0)
     goto bad;
     setsockopt(s,IPPROTO_SCTP,          SCTP_EVENTS,&evnts,
sizeof(evnts));
     setsockopt(s,  IPPROTO_SCTP,  SCTP_INITMSG,  &initmsg,
sizeof(initmsg));
//set 2 streams for sctp association (stream 0-1)!
```

Σχήμα 62

Με την παράμετρο "*initmsg.sinit_num_ostreams = 2*" ορίζουμε 2 εξερχόμενες ροές δεδομένων ενώ με την παράμετρο "*initmsg.sinit_max_instreams = 2*" ορίζουμε 2 εισερχόμενες ροές δεδομένων. Έτσι σε κάθε κατεύθυνση (αποστολή/λήψη) θα έχουμε 2 streams. Η παράμετρος "initmsg.sinit_max_attempts = 4" ορίζει τον μέγιστο αριθμό των αναμεταδόσεων (retransmissions) σε μια συσχέτιση (εδώ θα έχουμε μέχρι 4 αναμεταδώσεις). Με την παράμετρο "*setsockopt(socketd, IPPROTO_SCTP, SCTP_INITMSG, &initmsg, sizeof(initmsg))*" θέτουμε τις παραπάνω παραμέτρους στο συγκεκριμένο socket. Κατά τη διαδικασία της εγκαθίδρυσης του association λοιπόν, οι δύο σταθμοί (proxy και server) θα ανταλλάζουν αυτές τις τιμές. Από το σημείο αυτό λοιπόν, μέσω της κλήσης *sctp_sendmsg* θα επιλέγουμε από ποιό stream χρειάζεται να σταλθούν οι πληροφορίες (για τις πληροφορίες ελέγχου θα αναφερόμαστε στο stream 0 ενώ για την αποστολή δεδομένων στο stream 1).

4.5.2 Multihoming

Μπορούμε να εξελίξουμε τις εφαρμογές μας παρέχοντας υποστήριξη της λειτουργίας Multihoming. Αυτό είναι πολύ απλό, μπορούμε με μικρές μετατροπές του κώδικα του ftp client και του ftp server να παρέχουμε τη λειτουργία αυτή (μέσω του SCTP πρωτοκόλλου). Τροποποιώντας τον κώδικα του server, αποθηκεύουμε απλά σε έναν πίνακα όλες τις διευθύνσεις των interface του και τις κάνουμε bind με την κλήση **sctp_bindx()**. Έπειτα τροποποιούμε και τον κώδικα του client στον οποίο αντικαθιστώντας την κλήση connect από την **sctp_connectx()** παρέχουμε στον client τη δυνατότητα να διαβάσει μια λίστα με όλες τις διευθύνσεις του server. Με τον τρόπο αυτό, ο client θα συνδεθεί με τον server χρησιμοποιώντας τις εναλλακτικές διευθύνσεις σε περίπτωση που η κύρια διεύθυνση αυτού (interface) αποτύχει.
Ένα παράδειγμα κώδικα για τον server είναι το παρακάτω (σχήμα 63).

```
int main(int argc, char *argv[]) {
      int sockfd;
      int n;
      struct sockaddr_in addr, *addresses;
      int addr_size = sizeof(struct sockaddr_in);
      int port;
      if (argc < 2) {
          fprintf(stderr, "Usage %s client-addresses...\n", argv[0]);
          exit(1);
      }
      /* ΟΡΙΖΩ ΤΟ SOCKET */
      sockfd = socket(AF_INET, SOCK_STREAM,IPPROTO_SCTP);
      for ( n = 2; n < argc; n++) {
              addr.sin_family = AF_INET;
              addr.sin_addr.s_addr = inet_addr(argv[n]);
              addr.sin_port = port;
              memcpy(addresses + (n-2), &addr, addr_size);
      }
      /* ΓΙΝΟΝΤΑΙ BIND ΟΛΕΣ ΟΙ ΔΙΕΥΘΥΝΣΕΙΣ*/
      if (sctp_bindx(sockfd, (struct sockaddr *) addresses, addr_count,
      SCTP_BINDX_ADD_ADDR) == -1) {
              perror("sctp bindx");
              exit(2);
      }
      /* Η ΛΙΣΤΑ ΤΩΝ ΔΙΕΥΘΥΝΣΕΩΝ (ΠΟΥ ΕΓΙΝΑΝ BIND) ΤΟΥ SERVER */
      addr_count      =      sctp_getladdrs(sockfd,      0,      (struct
sockaddr**)&addresses);
      for (n = 0; n < addr_count; n++) {
              memcpy(&addr, addresses+n, addr_size);
              printf("addr %s, port %d\n",
                     inet_ntoa(addr.sin_addr.s_addr),
                     addr.sin_port);
      }
      close(sockfd);
      exit(0);
```

```
}
```

Σχήμα 63

5. Συμπεράσματα.

5.1 Συμπεράσματα

Σήμερα, στο διαδίκτυο χρησιμοποιούνται πρωτόκολλα που έχουν γραφτεί πριν από πολλά χρόνια. Στο επίπεδο μεταφοράς, τα πρωτόκολλα που χρησιμοποιούνται είναι κυρίως το TCP και το UDP. Τα πρωτόκολλα αυτά υπάρχουν πάνω από 30 χρόνια και παρέχουν λειτουργικότητα και σταθερότητα στη μεταφορά δεδομένων στο διαδίκτυο αλλά και στα δίκτυα υπολογιστών γενικότερα. Τα πρωτόκολλα αυτά εξελίχθηκαν και πρόσθεσαν πολλές τεχνολογίες μέσα σ'αυτά τα χρόνια προκειμένου να ανταποκριθούν στις ανάγκες και τις απαιτήσεις των χρηστών. Το πρωτόκολλο SCTP είναι ένα νέο πρωτόκολλο, άγνωστο ακόμα προς τους χρήστες του διαδικτύου. Αρχικά σχεδιάστηκε για να παρέχει υπηρεσίες για μεταφοράς τηλεφωνικής σηματοδοσίας. Στην πορεία προστέθηκαν διάφορες νέες τεχνολογίες και έτσι σήμερα καταλήξαμε να έχουμε ένα νέο ολοκληρωμένο πρωτόκολλο μεταφοράς. Το SCTP παρέχει όλες τις δυνατότητες του TCP καθώς και επιπλέον δυνατότητες που το κάνουν πιο αποτελεσματικό από το δεύτερο.

Στο πρακτικό μέρος της πτυχιακής, χρησιμοποιήσαμε το πρωτόκολλο αυτό τροποποιώντας διάφορες εφαρμογές. Συγκεκριμένα, εκτός από τον inetd daemon, τροποποιήσαμε και προγράμματα μεταφοράς αρχείων, όπως έναν ftp proxy(ftp.proxy1.2.3) και έναν ftp server(wu-ftpd), ώστε να χρησιμοποιείται στο επίπεδο μεταφοράς το SCTP πρωτόκολλο. Η απόδοση του πρωτοκόλλου δείχνει να είναι καλύτερη του TCP για τη μεταφορά αρχείων μεγάλου μεγέθους.

Κάποια γενικά συμπεράσματα για το SCTP πρωτόκολλο:

- Ο προγραμματισμός στο πρωτόκολλο SCTP έχει πολλά κοινά χαρακτηριστικά με τον προγραμματισμό στο TCP. Λόγω του παρόμοιου socket interface του sctp με αυτό του tcp, είναι εύκολη η τροποποίηση των προγραμμάτων που χρησιμοποιούν το tcp πρωτόκολλο ώστε να χρησιμοποιούν το sctp πρωτόκολλο.
- Το SCTP πρωτόκολλο είναι πιο ασφαλές από το TCP. Το SCTP εξασφαλίζει την ανοχή του τόσο σε SYN Attacks λόγω του μηχανισμού των COOKIES όσο και σε Blind Masquerate Attacks (verification tag).
- Το SCTP είναι πολύ χρήσιμο στην περίπτωση που θέλουμε να μεταφέρουμε μαζικά πολλά αρχεία, οπού λόγω του Multistreaming θα

έχουμε πολλές ροές δεδομένων σε μία σύνδεση, σε αντίθεση με το TCP στο οποίο θα χρειαζόμασταν τόσες συνδέσεις όσα και τα αρχεία που θέλουμε να μεταφέρουμε.

- Με το SCTP και τη Multihoming λειτουργία όχι μόνο εξασφαλίζεται η μεταφορά των δεδομένων σε περίπτωση που κάποια από τις κάρτες δικτύου κάποιου κόμβου σταματήσει να λειτουργεί, αλλά εξασφαλίζεται και η γρήγορη μεταφορά των δεδομένων από το μονοπάτι που δεν έχει υποστεί συμφόρηση.
- Όταν η εφαρμογή που θέλουμε να υλοποιήσουμε απαιτεί μετάδοση δεδομένων σε σειρά και χωρίς λάθη, τότε μπορούμε να χρησιμοποιήσουμε το SCTP πρωτόκολλο.

5.2 Μελλοντικές εφαρμογές – Μελλοντική δουλειά

Στην εργασία αυτή μελετήσαμε το πρωτόκολλο SCTP. Είδαμε τα κοινά χαρακτηριστικά του πρωτοκόλλου αυτού με το πρωτόκολλο TCP, καθώς και τις διαφορές τους. Έπειτα τροποποιώντας συγκεκριμένες FTP εφαρμογές, είδαμε πώς δουλεύει το πρωτόκολλο αυτό στην πράξη. Συγκρίνοντας λοιπόν τη λειτουργία του FTP μέσω του TCP και του SCTP πρωτοκόλλου είδαμε πώς οι εφαρμογές αυτές (μεταφοράς αρχείων) επωφελούνται (ή μπορούν να επωφεληθούν) από το πρωτόκολλο αυτό.

Λαμβάνοντας υπόψη όλες τις λειτουργίες του SCTP πρωτοκόλλου, όπως για παράδειγμα η λειτουργία multistreaming, η λειτουργία multihoming, η έννοια της συσχέτισης, η κατασκευή των chunk, ο μηχανισμός cookie που μας προστατεύει από Denial of Service επιθέσεις, συμπεραίνουμε ότι το μέλλον του SCTP πρωτοκόλλου προδιαγράφεται ελπιδοφόρο. Έτσι, μιας και το SCTP και μας παρέχει όλες τις υπηρεσίες των πρωτοκόλλων TCP και UDP αλλά και επιπλέον δυνατότητες, μπορούμε να ισχυριστούμε ότι μπορεί να γίνει ο αντικαταστάτης των πρωτοκόλλων αυτών από εφαρμογές που τελικά θα μας προσφέρουν καλύτερες ταχύτητες και μεγαλύτερη ασφάλεια σ'ολόκληρο το διαδίκτυο. Ήδη αυτή τη στιγμή γίνεται μια απόπειρα δημιουργίας RFC για τη λειτουργία του HTTP πάνω απο το SCTP πρωτόκολλο.

Βιβλιογραφία

[1] Douglas E. Comer, “Διαδίκτυα με TCP/IP – αρχές, πρωτόκολλα και αρχιτεκτονικές» , Κλειδάριθμος , Τέταρτη Αμερικανική έκδοση , 2005.

[2] Cisco Network Academy Program, “CCNA 1 and 2 - companion guide”, Ciscopress , Third edition , 2005.

[3] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and, V. Pax-son,
“Stream Control Transmission Protocol”, RFC 2960, 2000.
http://portal.acm.org/citation.cfm?id=RFC2960

[4] Iván Arias Rodríguez, “Stream Control Transmission Protocol - The design of a new reliable transport protocol for IP networks”, HELSINKI UNIVERSITY OF TECHNOLOGY , ABSTRACT OF MASTER'S THESIS , 2002.

[5] Άρης Αλεξόπουλος , Γιώργος Λαγογιάννης , “Τηλεπικοινωνίες και Δίκτυα Υπολογιστών” , Έκτη έκδοση , 2003.

[6] Ivan Griffin, Dr. John Nelson, “*Linux Network Programming”*, Linux Journal, 1998. http://www.linuxjournal.com/article/2333

[7] Wikipedia, “SCTP review” , Wikimedia Foundation, Inc.
http://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol

[8] Jan Newmarch, “Introduction to Stream Control Transmission Protocol”, Linux Journal, 2007. http://www.linuxjournal.com/article/9748

[9] IBM, “SCTP Socket API”, IBM Corporation.
http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.commadmn/doc/commadmndita/sctp_socketapis.htm

[10] Cisco, “TCP/IP” , Cisco Systems Inc., 1996.
http://cio.cisco.com/warp/public/535/4.html

[11] E. Kohler, M. Handley, S. Floyd , “Datagram Congestion Control Protocol (DCCP)”, RFC4340 , 2006. http://www.rfc-editor.org/rfc/rfc4340.txt

[12] Jukka Manner, “Protocol Software Engineering: Linux IP networking”, University of Helsinki, 2007. www.cs.helsinki.fi/u/jmanner/courses/pse/linux-net-1.pdf

[13] LWN.net, “Linux gets DCCP” , 2005. http://lwn.net/Articles/149756/

[14] J. Postel, “Transmission Control Protocol”, RFC 793, September 1981. http://tools.ietf.org/html/rfc793

[15] J. Postel, “User Datagram Protocol”, RFC 768, August 1980. http://tools.ietf.org/html/rfc768

[16] A.S.Tanenbaum, “Computer Networks, 3rd edition, Prentice Hall, 1996

[17] Jinwook Seo, Youngju Ahn, Minki No, Hyunchel Kim, Seongjin Ahn, Jinwook Chung, “Implement of FTP application using SCTP”, IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.12, December 2006. http://paper.ijcsns.org/07_book/200612/200612B22.pdf

[18] Rajesh Rajamani, Sumit Kumar, Nikhil Gupta, “SCTP versus TCP: Comparing the Performance of Transport Protocols for Web Traffic”, University of Wisconsin-Madison, July 2002. http://pages.cs.wisc.edu/~sumit/extlinks/sctp.pdf

[19] Δρ. Μπόζιος Ελευθέριος, "ΣΗΜΕΙΩΣΕΙΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΑΣΦΑΛΕΙΑΣ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ", ΑΤΕΙ Θεσσαλονίκης , Τμήμα Πληροφορικής, 2004. http://aetos.it.teithe.gr/~vaf/download_files/itsecnotes.pdf

[20] Wikipedia, “Datagram Congestion Control Protocol”, Wikimedia Foundation,Inc.
http://en.wikipedia.org/wiki/Datagram_Congestion_Control_Protocol

[21] Wikipedia, “TCP (Transmission Control Protocol - Πρωτόκολλο Ελέγχου Μεταφοράς)”, Wikimedia Foundation,Inc.
http://el.wikipedia.org/wiki/Transmission_Control_Protocol

[22] Randall Stewart, Michael Tuxen, Peter Lei, “SCTP: What is it, and how to use it?” , bsdcan.org, 2008.
http://www.bsdcan.org/2008/schedule/attachments/44_bsdcan_sctp.pdf

[23] Randall Stewart, Paul D. Amer, “Why is SCTP needed given TCP and UDP are widely available?”, The Internet Society (ISOC), September 2007.
http://www.isoc.org/briefings/017/

[24] IEC, “Stream Control Transmission Protocol (SCTP) “,International Engineering Consortium, 2007.
http://www.iec.org/online/tutorials/sctp/topic02.html

[25] Eddie Kohler, Mark Handley, Sally Floyd , “Designing DCCP: Congestion Control Without Reliability”, UCLA Computer Science Department, 2006. http://www.cs.ucla.edu/~kohler/pubs/kohler06designing.pdf

[26] Cisco, “How can SCTP and High-Speed networking go together?”, Cisco Systems Inc., 2004. http://acs.lbl.gov/DIDC/PFLDnet2004/talks/SCTP_HSnetwork.pdf

[27] Olaf Kirch, Terry Dawson, "Linux Network Administrators Guide: Chapter 12. ImportantNetwork Features", Linux Documentation Project Guides, March 2000. http://www.tldp.org/LDP/nag2/x-087-2-appl.inetd.html

[28] Σαββόπουλος Μιχάλης, Καζαρλής Σπύρος, Αδαμίδης Παναγιώτης, “Σημειώσεις εργαστηρίου: Λειτουργικά Συστήματα ΙΙ (Unix)”, ΑΤΕΙ Θεσσαλονίκης, Τμήμα Πληροφορικής

[29] Wikipedia, “Οδηγός χρήσης του Debian: Σύστημα διαχείρισης Πακέτων/Αναβάθμιση-ανανέωση έκδοσης”, Wikimedia Foundation,Inc.http://el.wikibooks.org/wiki/Οδηγός_**χρήσης**_του_Debian/Αναβάθμιση-ανανέωση_**έκδοσης**

[30] J. Postel, J. Reynolds, "FILE TRANSFER PROTOCOL (FTP)", RFC959, October 1985. http://www.ietf.org/rfc/rfc0959.txt

[31] L. Coene, Siemens, “Stream Control Transmission Protocol Applicability Statement”, RFC3257, 2002. http://www.ietf.org/rfc/rfc3257.txt

[32] Sourabh Ladha, Paul D. Amer, “Improving Multiple File Transfers Using SCTP Multistreaming”, Performance, Computing, and Communications, 2004 IEEE International Conference on. http://www.cis.udel.edu/~amer/PEL/poc/pdf/TR2003-06.FTP.over.SCTP.Ladha.pdf

[33] P. Natarajan, P. Amer, J. Leighton, F. Baker, "Using SCTP as a Transport Layer Protocol for HTTP", draft-natarajan-httpbis-sctp-00.txt , October 27, 2008. http://tools.ietf.org/html/draft-natarajan-httpbis-sctp-00