

# *TCP*

(Transmission Control  
Protocol)

Αβραάμ Κεβρεκίδης  
Μανώλης Μελάκης  
Μάριος Ιακώβου

## 3.5 Προσανατολισμένη προς τη Σύνδεση Μεταφορά. TCP: (Πρωτόκολλο Ελέγχου Μετάδοσης).

### 3.5.1. Σύνδεση TCP

Το TCP υποστηρίζει **multiplexing** (πολύπλεξη), **demultiplexing** (απόπλεξη) και **error detection** (ανίχνευση σφαλμάτων) (αλλά όχι την αποκατάστασή τους) με ακριβώς τον ίδιο τρόπο με το UDP. Εντούτοις, το TCP και UDP διαφέρουν σε πολλά σημεία. Η πιο θεμελιώδης διαφορά είναι ότι το UDP είναι χωρίς σύνδεση (connection less) ενώ το TCP είναι προσανατολισμένο προς την σύνδεση. Το UDP είναι χωρίς σύνδεση επειδή στέλνει τα δεδομένα χωρίς πάντα να επιτυγχάνεται μια σύνδεση. Το TCP είναι προσανατολισμένο προς την σύνδεση, επειδή προτού καταφέρει να αρχίσει η μία διαδικασία εφαρμογής να στέλνει τα δεδομένα σε μια άλλη, πρέπει οι δύο διαδικασίες να 'συμφωνήσουν' (**handshake**) η μία με την άλλη. Δηλαδή πρέπει να στείλουν μερικά προκαταρκτικά πακέτα δεδομένων η μία στην άλλη για να καθοριστούν οι παράμετροι της επιτευχθείσας μεταφοράς δεδομένων. Ως τμήμα της επίτευξης σύνδεσης του TCP και οι δύο πλευρές της σύνδεσης θα αρχικοποιήσουν πολλές μεταβλητές TCP που συσχετίζονται με την TCP σύνδεση.

Η TCP σύνδεση δεν είναι ένα συνεχές κύκλωμα TDM ή FDM όπως ένα δίκτυο μεταγόμενων κυκλωμάτων. Ούτε είναι ένα εικονικό κύκλωμα, επειδή η κατάσταση της σύνδεσης βασίζεται αποκλειστικά στα δύο τελικά συστήματα. Επειδή το TCP πρωτόκολλο τρέχει μόνο στα ακραία συστήματα (αποστολέα & αποδέκτη) και όχι σε ενδιάμεσα στοιχεία δικτύων (δρομολογητές και γέφυρες), τα ενδιάμεσα στοιχεία δικτύων δεν διατηρούν την κατάσταση σύνδεσης TCP. Στην πραγματικότητα, οι ενδιάμεσοι δρομολογητές αγνοούν παντελώς και εξ' ολοκλήρου τις TCP συνδέσεις: 'βλέπουν' μόνο τα datagrams (αυτοδύναμα πακέτα).

Μία TCP σύνδεση υποστηρίζει 'αμοιβαία' (full duplex) μεταφορά δεδομένων. Αυτό γίνεται γιατί τα δεδομένα σε επίπεδο εφαρμογής μπορούν να μεταφερθούν ταυτόχρονα και στις δύο κατευθύνσεις μεταξύ των hosts - αν υπάρχει σύνδεση TCP μεταξύ της πρώτης διαδικασίας σ' ένα host και της δεύτερης διαδικασίας σ' ένα άλλο host, τότε τα δεδομένα επιπέδου εφαρμογής μπορούν να μεταφερθούν από το A στο B και από το B στο A ταυτόχρονα. Η TCP σύνδεση πραγματοποιείται πάντα από σημείο σε σημείο (point to point) για παράδειγμα μεταξύ ενός συγκεκριμένου αποστολέα και ενός συγκεκριμένου δέκτη. Το multicasting (μεταφορά δεδομένων από έναν αποστολέα σε πολλούς δέκτες με μία εκπομπή δεδομένων) δεν είναι εφικτό με το TCP. Με το TCP μπορεί το πολύ να έχουμε 2 hosts και όχι περισσότερους..

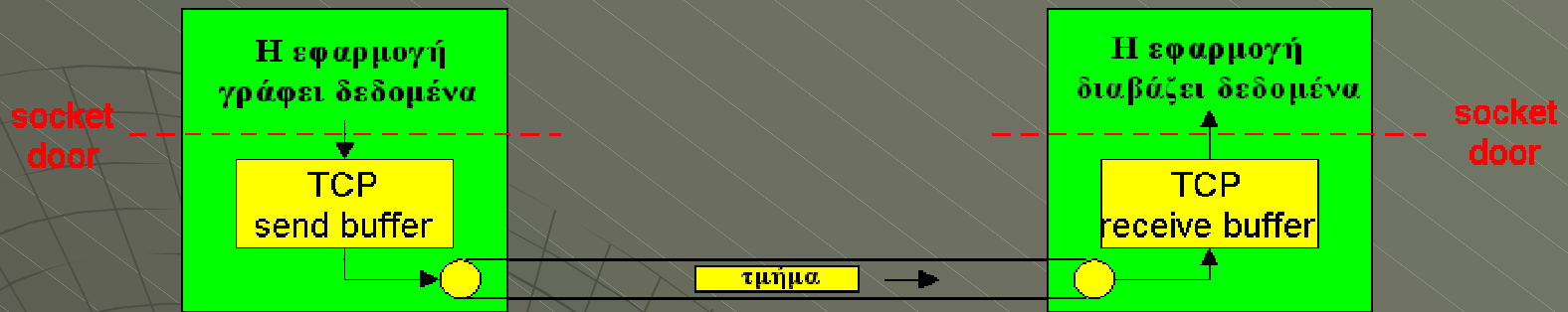
Ας ρίξουμε μια ματιά στο πώς επιτυγχάνεται μία σύνδεση TCP. Ας υποθέσουμε ότι μία διαδικασία τρέχει στον ένα host και ο host αυτός θέλει να επιτύχει σύνδεση με μία διαδικασία ενός άλλου host. Σημειώνουμε ότι ο host που πραγματοποιεί την σύνδεση ονομάζεται **πελάτης** (client host), ενώ ο άλλος host ονομάζεται server host. Η διαδικασία εφαρμογής πελάτη (client application-process) πρώτα ενημερώνει τον πελάτη TCP ότι θέλει να πραγματοποιήσει μια σύνδεση σε μια διαδικασία του server. Κάτι παρόμοιο συμβαίνει στην java με την εντολή:

```
Socket clientsocket=new socket ("hostname", "port number");
```

Το TCP στο πελάτη τότε προβαίνει στην επίτευξη μιας σύνδεσης TCP, με το TCP στον server. Θα συζητήσουμε με λεπτομέρειες την διαδικασία πραγματοποίησης της σύνδεσης στο τέλος αυτής της ενότητας. Για την ώρα αρκεί να ξέρουμε ότι ο πελάτης στέλνει ένα ειδικό πακέτο TCP · ο server ανταποκρίνεται με ένα δεύτερο ειδικό πακέτο TCP και τελικά ο πελάτης ανταποκρίνεται και πάλι με ένα τρίτο ειδικό πακέτο. Τα δύο πρώτα πακέτα δεν περιέχουν "payload", δηλαδή δεν περιέχουν δεδομένα επιπέδου εφαρμογής. Το τρίτο απ' αυτά τα πακέτα μπορεί να μεταφέρει ένα 'payload'. Επειδή τα τρία πακέτα στέλνονται μεταξύ των 2 hosts, αυτή η διαδικασία πραγματοποίησης της σύνδεσης συχνά αναφέρεται ως **τριμερής χειραψία** (three-way handshake).

Εφόσον πραγματοποιηθεί η σύνδεση TCP, οι δύο διαδικασίες εφαρμογών μπορούν να στείλουν δεδομένα η μία στην άλλη· επειδή το TCP είναι **'full duplex'** μπορούν να στείλουν δεδομένα ταυτόχρονα. Ας θεωρήσουμε την αποστολή δεδομένων από τον πελάτη στον server. Ο πελάτης περνάει μία ακολουθία δεδομένων από την υποδοχή (που είναι η πόρτα της διαδικασίας). Από την στιγμή που τα δεδομένα περνούν από την πόρτα τώρα περνάνε στα χέρια του TCP που τρέχει στον πελάτη. Όπως φαίνεται στο σχήμα 3.5-1 το TCP κατευθύνει αυτά τα δεδομένα στο **'send buffer'** (προσωρινός χώρος) της σύνδεσης, ο οποίος είναι ένας χώρος που δεν χρησιμοποιείται κατά την διάρκεια της αρχικής τριπλής χειραψίας (three-way handshake). Κατά διαστήματα το TCP θα **'αρπάξει'** τα μεγάλα πακέτα δεδομένων από τον προσωρινό χώρο αποστολής. Το μέγιστο πλήθος δεδομένων που μπορεί να επιλεγεί και να τοποθετηθεί σ' ένα πακέτο περιορίζεται από το **'Maximum Segment Size'** (Μέγιστο Μέγεθος Πακέτου) (MSS). Το MSS εξαρτάται από την εφαρμογή του TCP (που καθορίζεται από το λειτουργικό σύστημα) και μπορεί συχνά να διαμορφωθεί· συνηθισμένες τιμές είναι 1500 bytes, 536 bytes και 512 bytes (αυτά τα μεγέθη πακέτων επιλέγονται συχνά προκειμένου να επιτευχθεί ο κατακερματισμός των δεδομένων)

Σημειώσατε ότι το MSS είναι το μέγιστο πλήθος δεδομένων επιπέδου εφαρμογής στο πακέτο και όχι το μέγιστο μέγεθος του πακέτου TCP, συμπεριλαμβανομένων και των επικεφαλίδων.



Σχήμα 3.5-1 (buffers αποστολής και λήψης του TCP)

Το TCP τοποθετεί κάθε μεγάλο (chunk) κομμάτι των δεδομένων του πελάτη μαζί με την TCP επικεφαλίδα και ως εκ τούτου σχηματίζει πακέτα TCP. Τα πακέτα περνούν στο επίπεδο του δικτύου όπου ομαδοποιούνται χωριστά σε IP αυτοδύναμα πακέτα δεδομένων στρώματος δικτύου. Τα πακέτα αυτά στέλνονται τότε στο δίκτυο. Όταν το TCP παραλαμβάνει ένα πακέτο στο άλλο άκρο, το πακέτο δεδομένων τοποθετείται στο χώρο λήψης (receive buffer) της TCP σύνδεσης. Η εφαρμογή διαβάζει την ροή των δεδομένων από αυτό το χώρο. Κάθε πλευρά της σύνδεσης έχει το δικό της χώρο αποστολής και λήψης. Οι χώροι αποστολής και λήψης ροής δεδομένων φαίνονται στο σχήμα 3.5-1.

Απ' αυτήν την συζήτηση βλέπουμε ότι μια σύνδεση TCP αποτελείται από τους προσωρινούς χώρους, μεταβλητές και σύνδεση υποδοχών προς μια διαδικασία ενός host και ένα άλλο σύνολο από χώρους, μεταβλητές και σύνδεση υποδοχών σε μια άλλη διαδικασία ενός άλλου host. Όπως αναφέρθηκε προηγουμένως, κανένας χώρος ή μεταβλητή δεν δεσμεύεται στη σύνδεση στα στοιχεία δικτύου (routers, bridges και repeaters) μεταξύ των hosts.

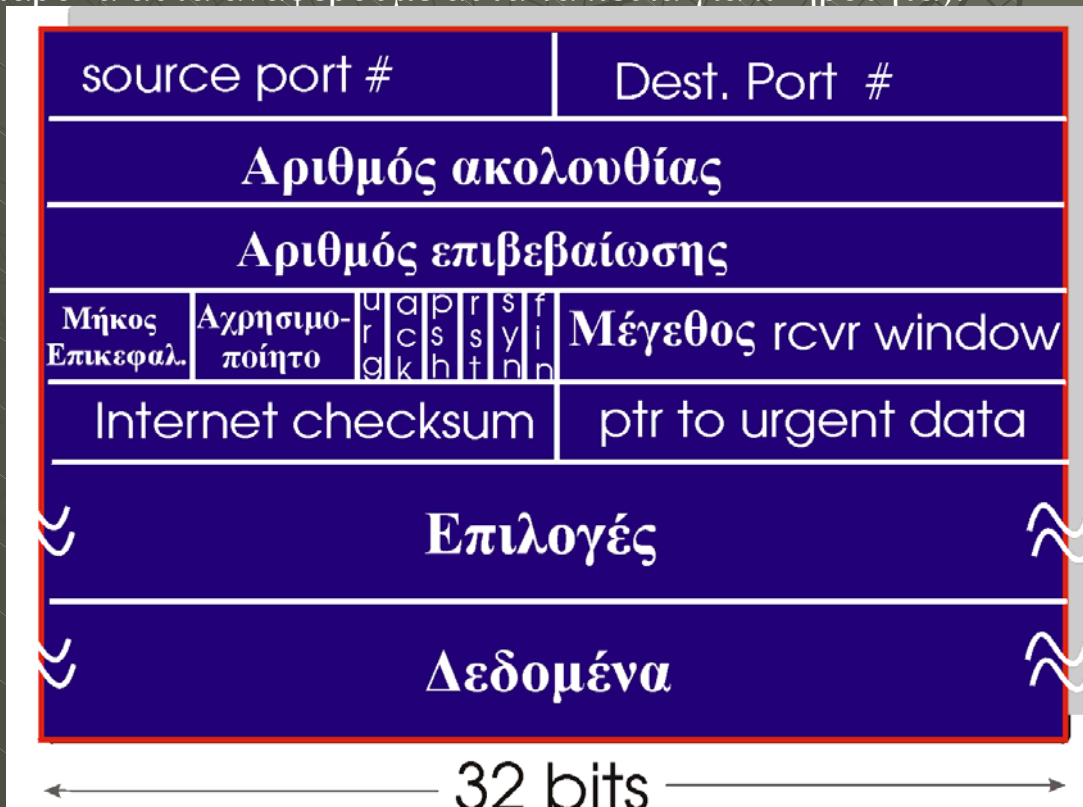
### 3.5.2. Δομή Πακέτου TCP

Έχοντας ρίξει μια σύντομη ματιά στην σύνδεση TCP ας εξετάσουμε την δομή πακέτου TCP. Ένα πακέτο TCP αποτελείται από πεδία επικεφαλίδων και ένα πεδίο δεδομένων. Το πεδίο δεδομένων περιέχει ένα μεγάλο τμήμα δεδομένων εφαρμογής. Όπως αναφέρθηκε παραπάνω, το MSS περιορίζει το μέγιστο μέγεθος του πεδίου δεδομένων ενός πακέτου. Όταν το TCP στέλνει ένα μεγάλο αρχείο, όπως μια κωδικοποιημένη εικόνα που είναι τμήμα μιας Web page, σπάει τυπικά το αρχείο σε μεγάλα κομμάτια μεγέθους MSS (εκτός από το τελευταίο κομμάτι, που συχνά είναι μικρότερο από το MSS). Εντούτοις, οι interactive εφαρμογές συχνά μεταφέρουν κομμάτια δεδομένων μικρότερα από ένα MSS· για παράδειγμα με εφαρμογές απομακρυσμένης άδειας εισόδου όπως το Telnet, το πεδίο δεδομένων στο πακέτο TCP είναι συχνά μόνο ένα byte. Επειδή η επικεφαλίδα TCP είναι συνήθως 20 bytes (12 bytes μεγαλύτερη από την επικεφαλίδα UDP), τα πακέτα που μπορούν να σταλούν από το Telnet είναι μόνο 21 bytes.

Το σχήμα 3.5-2 δείχνει την δομή ενός πακέτου TCP. Όπως και στο UDP η επικεφαλίδα περιλαμβάνει τους αριθμούς των ports προέλευσης και προορισμού, τα οποία χρησιμοποιούνται για multiplexing/demultiplexing δεδομένων από και προς το άνω στρώμα εφαρμογών. Επίσης, όπως και στο UDP, η επικεφαλίδα περιλαμβάνει checksum (άθροισμα/σύνολο ελέγχων) πεδίο. Μία επικεφαλίδα πακέτου TCP επίσης περιλαμβάνει τα ακόλουθα πεδία:

- Το πεδίο αριθμού ακολουθίας μεγέθους 32-bit και το πεδίο αριθμού επιβεβαίωσης επίσης 32-bit, χρησιμοποιούνται από τον αποστολέα και τον δέκτη TCP στην εφαρμογή μιας υπηρεσίας αξιόπιστης μεταφοράς δεδομένων όπως αναφέρεται και πιο κάτω.
- Το πεδίο μεγέθους παραθύρου που είναι 16-bit χρησιμοποιείται για έλεγχο ροής. Θα δούμε εν συντομία ότι χρησιμοποιείται για την κατάδειξη του αριθμού των bytes που ένας παραλήπτης είναι διατεθειμένος να δεχτεί.
- Το πεδίο μήκους μεγέθους 4 bits προσδιορίζει το μήκος της επικεφαλίδας TCP σε 32-bit λέξεις. Η επικεφαλίδα TCP μπορεί να είναι μεταβλητού μήκους το οποίο οφείλεται στο πεδίο επιλογών TCP και αναφέρεται πιο κάτω (ουσιαστικά το πεδίο επιλογών είναι άδειο έτσι ώστε το μήκος της τυπικής επικεφαλίδας TCP να είναι 20 bytes).
- Το προαιρετικό και μεταβλητό μήκος πεδίου επιλογών χρησιμοποιείται όταν ένας αποστολέας και ένας δέκτης καθορίζουν από κοινού το MSS ή χρησιμοποιείται ως ένας παράγοντας παραθύρου για χρήση δικτύων υψηλής ταχύτητας. Επίσης μια timestamping (σφραγίδα με ημερομηνία και ώρα) επιλογή καθορίζεται.

• Το **flag field** περιέχει 6 bits. Το ACK bit χρησιμοποιείται για να υποδεικνύει ότι η τιμή που μεταφέρεται στο πεδίο αναγνώρισης είναι έγκυρη. Τα RST, SYN και FIN bits χρησιμοποιούνται για την ρύθμιση και για τον τερματισμό της σύνδεσης, όπως θα αναφέρουμε και στο τέλος της ενότητας. Όταν το PSH bit καθορίζεται, αυτό είναι μια ένδειξη ότι ο δέκτης πρέπει να μεταφέρει αμέσως τα δεδομένα στο επόμενο στρώμα. Τελικά η URG χρησιμοποιείται για να υποδείξει ότι υπάρχουν δεδομένα σ' αυτό το πακέτο των οποίων η πλευρά αποστολής του άνω στρώματος τα χαρακτηρίζει όλα ως επείγοντα. Η διεύθυνση του τελευταίου byte αυτού του πακέτου επειγόντων δεδομένων καταδεικνύεται από τον επείγοντα δέκτη δεδομένων που έχει μέγεθος 16 bit. Το TCP πρέπει να ενημερώσει την λαμβάνουσα πλευρά ολόκληρου του άνω στρώματος όταν υπάρχουν επείγοντα δεδομένα και να τα “σημαδεύσει” στο τέλος τους (στην πράξη το PSH, URG και ο δείκτης επειγόντων δεδομένων δεν χρησιμοποιούνται, παρόλα αυτά αναφέρουμε αυτά τα πεδία για πληρότητα).

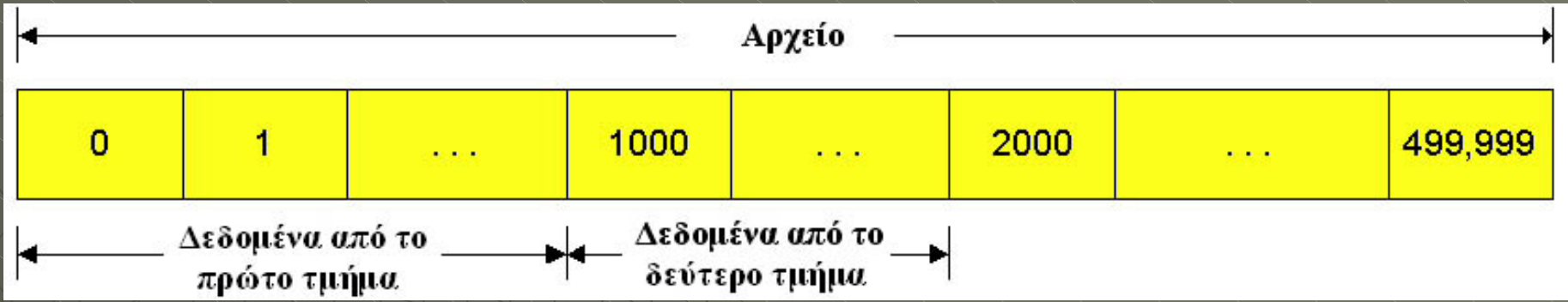


Σχήμα 3.5-2 (δομή πακέτων TCP)

### 3.5.3. Αριθμοί ακολουθίας και επιβεβαίωσης

Δύο από τα πιο σημαντικά πεδία στο τμήμα επικεφαλίδας TCP είναι το πεδίο αριθμού ακολουθίας και το πεδίο αριθμού επιβεβαίωσης. Αυτά τα πεδία είναι ένα πολύ σημαντικό μέρος της αξιόπιστης υπηρεσίας μεταφοράς δεδομένων του TCP. Όμως πριν αναφέρουμε πώς χρησιμοποιούνται αυτά τα πεδία για παροχή αξιόπιστης μεταφοράς δεδομένων ας εξηγήσουμε πρώτα τι ακριβώς τοποθετεί το TCP σ' αυτά τα πεδία.

Το TCP “βλέπει” τα δεδομένα σαν μία ροή bytes τα οποία βρίσκονται σε σειρά, αλλά είναι χωρίς δομή. Η χρήση των αριθμών ακολουθίας από το TCP δείχνει ότι αυτή η μορφή της σειράς των αριθμών είναι μεγαλύτερη της ροής μεταφοράς bytes και όχι μεγαλύτερη της σειράς μεταφερόμενων τμημάτων. Ο αριθμός ακολουθίας ενός πακέτου είναι ο αριθμός ροής byte του πρώτου byte ενός πακέτου. Ας δούμε ένα παράδειγμα :Υποθέτουμε ότι η διαδικασία στο host A θέλει να στείλει μια ροή δεδομένων σε μια διαδικασία στον host B μέσω της σύνδεσης TCP. Το TCP στο host A θα αριθμήσει “σιωπηρά” (στο παρασκήνιο) κάθε byte στην ροή δεδομένων. Ας υποθέσουμε ότι η ροή δεδομένων αποτελείται από ένα αρχείο 500.000 bytes από τα οποία τα 1000 bytes είναι MSS και όπου το πρώτο byte της ροής δεδομένων είναι ίσο με το 0. Όπως φαίνεται στο σχήμα 3.5-3 , το TCP χωρίζει τη ροή δεδομένων σε 500 πακέτα . Στο πρώτο πακέτο καταχωρείται ο αριθμός ακολουθίας 0, στο δεύτερο πακέτο ο αριθμός ακολουθίας 1000, στο τρίτο ο αριθμός ακολουθίας 2000 κ.ο.κ. Κάθε αριθμός ακολουθίας εισάγεται στο πεδίο αριθμού ακολουθίας της επικεφαλίδας του κατάλληλου πακέτου TCP.



Σχήμα 3.5-3 (χωρίζοντας τα δεδομένα ενός αρχείου σε πακέτα TCP)

Ας εξετάσουμε τους αριθμούς επιβεβαίωσης. Είναι λίγο πιο περίπλοκο από τους αριθμούς ακολουθίας. Γνωρίζουμε ότι το TCP είναι “full duplex” έτσι ώστε ο host A μπορεί να παραλαμβάνει δεδομένα από τον host B ενώ στέλνει δεδομένα στον host B (καθότι είναι μέρη της ίδιας σύνδεσης TCP). Καθένα από τα πακέτα που λαμβάνονται από τον host B έχουν ένα αριθμό ακολουθίας για τα δεδομένα που πηγάζουν από το B στο A. Ο αριθμός επιβεβαίωσης που βάζει ο host A στα πακέτα του είναι ο αριθμός ακολουθίας του επόμενου byte που ο host A αναμένει από τον host B. Είναι καλό να κοιτάξουμε μερικά παραδείγματα για να καταλάβουμε τι συμβαίνει. Ας υποθέσουμε ότι ο host A έχει λάβει όλα τα bytes από το 0-535 από τον host B και ας υποθέσουμε ότι πρόκειται να στείλει ένα πακέτο στον host B. Με άλλα λόγια ο host A περιμένει το byte 536 και όλα τα bytes που ακολουθούν στην ροή δεδομένων του host B. Έτσι ο host A τοποθετεί ‘536’ στο πεδίο αριθμού επιβεβαίωσης του πακέτου που στέλνει στο B.

Ένα άλλο παράδειγμα: Ας υποθέσουμε ότι ο host A έχει λάβει ένα πακέτο από τον host B που περιέχει τα bytes από το 0 έως το 535 και ένα άλλο τμήμα που περιέχει από το 900 μέχρι το 1000. Για κάποιον λόγο ο host A δεν έχει ακόμα παραλάβει τα bytes 536 μέχρι 899. Σ' αυτό το παράδειγμα ο host A ακόμα περιμένει το byte 536 (και πέρα) έτσι ώστε να αναδιαμορφώσει την ροή δεδομένων του B. Κατά συνέπεια το επόμενο πακέτο του A στο B θα περιέχει το '536' στο πεδίο αριθμού επιβεβαίωσης. Επειδή το TCP αναγνωρίζει μόνο bytes μέχρι το πρώτο ελλιπών byte της ροής, το TCP παρέχει αθροιστικές επιβεβαιώσεις (cumulative acknowledgements).

Επίσης, το τελευταίο παράδειγμα εμφανίζει σημαντικό αλλά λεπτό ζήτημα. Ο host A παραλαμβάνει το τρίτο πακέτο (900-1000 bytes) προτού παραλάβει το δεύτερο (536-899 bytes). Γι' αυτό το λόγο το τρίτο πακέτο παραλαμβάνεται σε λάθος σειρά. Το λεπτό ζήτημα είναι: Τι κάνει ένας host σε μια TCP σύνδεση όταν δέχεται πακέτα σε λάθος σειρά; Παρουσιάζει ενδιαφέρον ότι τα TCP RFCS δεν επιβάλλουν οποιουσδήποτε κανόνες εδώ και αφήνουν την απόφαση σ' αυτούς που προγραμματίζουν μια TCP εφαρμογή. Υπάρχουν βασικά 2 επιλογές:

(i) Ο αποδέκτης αμέσως απορρίπτει τα bytes που δεν είναι στην αναμενόμενη σειρά (out of order bytes) (GO-BACK-N)

ή,

(ii) Ο αποδέκτης κρατάει τα bytes που δεν είναι στην σειρά (out of order bytes) και περιμένει τα bytes που "αγνοούνται" (missing bytes) για να γεμίσουν τα κενά (SRP).

Φανερά η δεύτερη επιλογή είναι πιο αποδοτική ως αναφορά το εύρος του δικτύου (bandwidth), ενώ η πρώτη επιλογή απλουστεύει σημαντικά τον κώδικα TCP. Εν ολίγοις, απ' όλα αυτά που συζητήσαμε για το TCP, θα επικεντρωθούμε στην πρώτη εφαρμογή στην οποία υποθέτουμε ότι ο δέκτης TCP απορρίπτει τα πακέτα που δεν είναι σε σειρά (out of order segments).

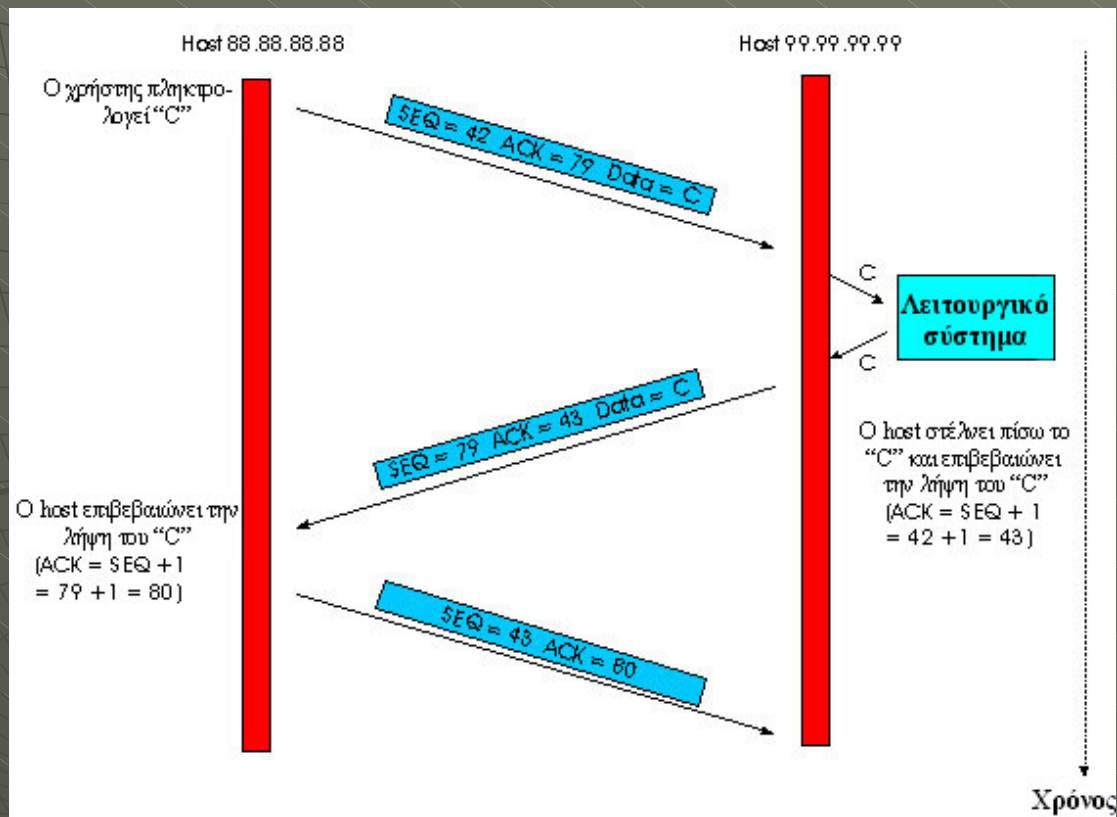
Στο σχήμα 3.5-3 υποθέτουμε ότι ο αρχικός αριθμός ακολουθίας είναι 0. Στην πραγματικότητα και οι δύο πλευρές μιας TCP σύνδεσης διαλέγουν τυχαία έναν αρχικό αριθμό ακολουθίας. Αυτό γίνεται για να ελαχιστοποιήσουμε την πιθανότητα σε μια πιο πρόσφατη σύνδεση μεταξύ 2 hosts να ληφθεί ως σωστό ένα πακέτο το οποίο είχε παραμείνει στο δίκτυο από μια προηγούμενη σύνδεση που είχε διακοπεί (και η παρούσα και η προηγούμενη σύνδεση τυγχάνει να έχουν τους ίδιους αριθμούς port).

#### ***3.5.4. Telnet: Μια προσεκτική μελέτη για τους αριθμούς ακολουθίας και επιβεβαίωσης.***

Το Telnet είναι ένα δημοφιλές πρωτόκολλο επιπέδου εφαρμογής που χρησιμοποιείται για την απομακρυσμένη άδεια εισόδου(remote login). Τρέχει βασισμένο στο TCP και είναι σχεδιασμένο να δουλεύει μεταξύ οποιουδήποτε ζευγαριού hosts. Σε αντίθεση με εφαρμογές μεταφοράς μεγάλου αριθμού δεδομένων, το Telnet είναι μια αμφίδρομη εφαρμογή. Θα αναφέρουμε ένα παράδειγμα του Telnet που παρουσιάζει τους αριθμούς ακολουθίας και επιβεβαίωσης του TCP.

Υποθέτουμε ότι ένας host, ο 88.88.88.88 πραγματοποιεί μια επικοινωνία Telnet με τον host 99.99.99.99. Επειδή ο host 88.88.88.88 πραγματοποιεί την επικοινωνία, χαρακτηρίζεται ως πελάτης (client) και ο host 99.99.99.99 ονομάζεται server. Κάθε χαρακτήρας που πληκτρολογείται από τον client θα σταλεί στον απομακρυσμένο host· ο απομακρυσμένος host θα στείλει πίσω ένα αντίγραφο του κάθε χαρακτήρα το οποίο θα εμφανιστεί στην οθόνη του χρήστη Telnet. Αυτός ο αντίλαλος (echo back) μεταξύ των hosts χρησιμοποιείται για να εξασφαλίσει ότι οι χαρακτήρες που εμφανίζονται από τον χρήστη Telnet έχουν ήδη παραληφθεί και επεξεργαστεί από το απομακρυσμένο site. Κάθε χαρακτήρας διαπερνά το δίκτυο δύο φορές· μία όταν ο χρήστης πατάει το πλήκτρο και μία όταν ο χαρακτήρας εμφανίζεται στην οθόνη του χρήστη.

Υποθέτουμε ότι ο χρήστης πληκτρολογεί το γράμμα “C” και κάνει διάλειμμα. Ας εξετάσουμε τα πακέτα TCP που στέλνονται μεταξύ του client και του server. Όπως φαίνεται στο σχέδιο 3.5-4 υποθέτουμε ότι οι αρχικοί αριθμοί ακολουθίας είναι 42 και 79 για τον client και τον server αντίστοιχα. Σημειώνουμε ότι ο αριθμός ακολουθίας ενός πακέτου είναι ο αριθμός ακολουθίας του πρώτου byte στο πεδίο δεδομένων. Γι’ αυτό το λόγο το πρώτο πακέτο που θα σταλεί από τον client θα έχει αριθμό ακολουθίας 42· το πρώτο πακέτο που θα σταλεί από τον server θα έχει αριθμό ακολουθίας 79. Ας σημειώσουμε ότι ο αριθμός επιβεβαίωσης είναι ο αριθμός ακολουθίας του επόμενου byte των δεδομένων που περιμένει ο host. Μετά την επίτευξη της TCP σύνδεσης, αλλά πριν σταλούν οποιαδήποτε δεδομένα, ο client περιμένει το byte 79 και ο server το byte 42.

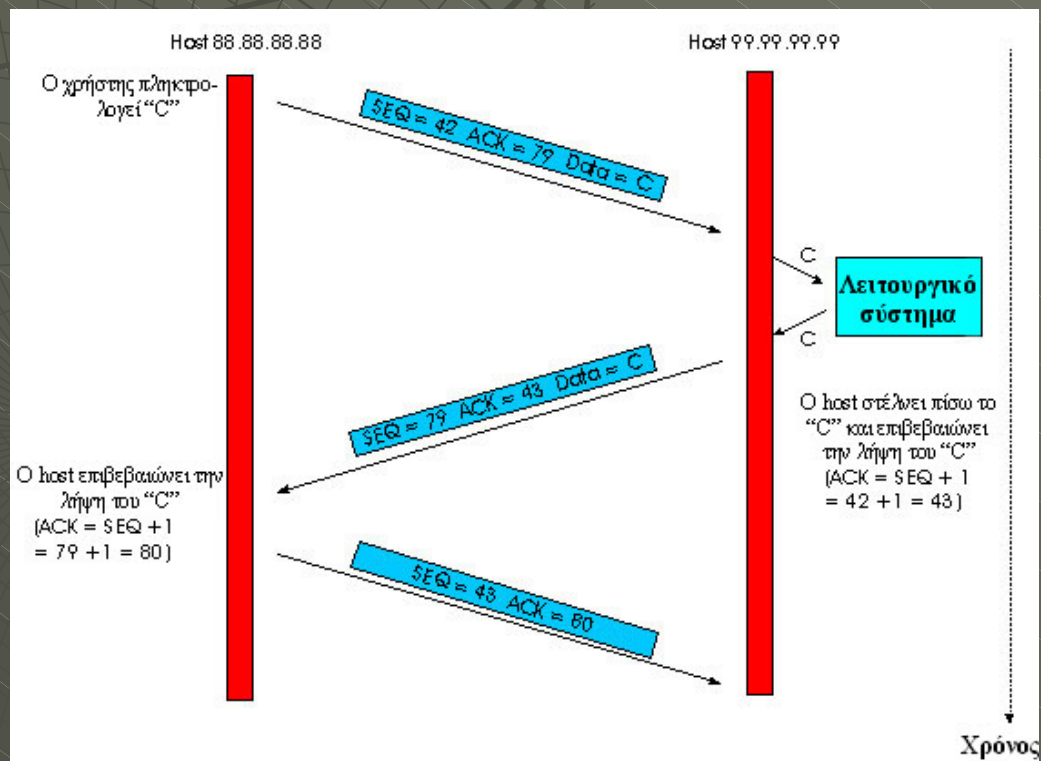


Σχήμα 3.5-4 (αριθμοί ακολουθίας και επιβεβαίωσης για μια απλή εφαρμογή Telnet μέσω του TCP)

Όπως φαίνεται στο σχήμα 3.5.4 στέλνονται 3 πακέτα.

- Το **πρώτο πακέτο** στέλνεται από τον client στον server και περιέχει την ASCII αναπαράσταση του χαρακτήρα “C” στο πεδίο δεδομένων. Το πρώτο πακέτο έχει επίσης το 42 στο πεδίο αριθμού ακολουθίας του όπως μόλις περιγράψαμε. Επίσης, επειδή ο client δεν έχει ακόμα δεχτεί δεδομένα από τον server, αυτό το πρώτο πακέτο θα έχει το 79 στο πεδίο αριθμού επιβεβαίωσης του.
- Το **δεύτερο πακέτο** στέλνεται από τον server στον client. Αυτό έχει διπλή χρησιμότητα: Πρώτον, παρέχει μια επιβεβαίωση για τα δεδομένα που έχει λάβει ο client. Τοποθετώντας το 43 στο πεδίο επιβεβαίωσης (acknowledgement field), ο server «επιβεβαιώνει» στον client ότι έχει λάβει επιτυχώς όλα τα δεδομένα μέχρι το byte 42 και τώρα περιμένει τα bytes 43 και άνω. Η δεύτερη χρησιμότητα αυτού του πακέτου είναι ο «αντίλαλος» (echo back) μεταξύ των 2 hosts του γράμματος “C”. Έτσι, το δεύτερο πακέτο έχει την αναπαράσταση ASCII του “C” στο πεδίο δεδομένων του. Το δεύτερο πακέτο έχει τον αριθμό ακολουθίας 79, που είναι ο αρχικός αριθμός ακολουθίας της ροής δεδομένων από τον server στον client αυτής της σύνδεσης TCP, εφόσον αυτό είναι το πρώτο byte δεδομένων που στέλνει ο server. Ας σημειωθεί ότι η επιβεβαίωση για τα δεδομένα από τον client στον server μεταφέρεται σ’ ένα πακέτο το οποίο μεταφέρει δεδομένα από τον server στον client. Αυτή η επιβεβαίωση είναι επισυνημμένη στο πακέτο δεδομένων από τον server στον client.

Το **τρίτο πακέτο** στέλνεται από τον client στον server. Ο μοναδικός του σκοπός είναι να επιβεβαιώσει τα δεδομένα που έλαβε από τον server. (Ας σημειωθεί ότι το δεύτερο πακέτο περιείχε δεδομένα – το γράμμα ‘C’ – από τον server στον χρήστη). Έχει κενό πεδίο δεδομένων (για παράδειγμα, η επιβεβαίωση δεν είναι επισυνημμένη με κανένα δεδομένο από τον client στον server). Το πακέτο έχει το 80 στο πεδίο αριθμού ακολουθίας επειδή ο client έχει λάβει μια ροή απο bytes μέχρι τον αριθμό ακολουθίας 79 και τώρα περιμένει τα bytes 80 και άνω. Μπορεί να φανεί περίεργο ότι αυτό το πακέτο έχει επίσης αριθμό ακολουθίας αφού δεν περιέχει δεδομένα. Αυτό συμβαίνει επειδή το TCP πρέπει οπωσδήποτε να έχει πεδίο αριθμού ακολουθίας.



Σχήμα 3.5-4 (αριθμοί ακολουθίας και επιβεβαίωσης για μια απλή εφαρμογή Telnet μέσω του TCP)

### 3.5.5 Αξιόπιστη Μεταφορά Δεδομένων

Αναφέρουμε ότι η υπηρεσία (σε επίπεδο δικτύου) του Internet (IP service) είναι αναξιόπιστη. Το IP δεν εγγυάται τη διανομή αυτοδύναμων πακέτων (datagrams), ούτε την κατά σειρά διανομή των datagram ούτε και την ακεραιότητα των δεδομένων τους. Με την υπηρεσία IP, τα datagrams μπορούν να «υπερχειλίσουν» τους χώρους προσωρινής αποθήκευσης των routers (overflow router buffers), να μην φτάσουν ποτέ στον προορισμό τους, τα datagrams να φτάσουν εκτός σειράς και τα bits στα datagrams μπορεί να αλλοιωθούν ( να αλλαχθεί κάποιο bit από 0 σε 1 και αντίστροφα). Επειδή τα πακέτα σε επίπεδο μεταφοράς ταξιδεύουν στο δίκτυο μέσα από τα datagrams του IP, μπορούν να επηρεαστούν από αυτά τα προβλήματα.

Το TCP υποστηρίζει μια υπηρεσία αξιόπιστης μεταφοράς δεδομένων που υπερκαλύπτει τις αναξιόπιστες υπηρεσίες του IP. Πολλα δημοφιλή πρωτόκολλα εφαρμογών – συμπεριλαμβανομένων των FTP, SMTP, NNTP, HTTP και Telnet – χρησιμοποιούν το TCP παρά το UDP, κυρίως επειδή το TCP παρέχει υπηρεσία αξιόπιστης μεταφοράς δεδομένων. Η αξιόπιστη υπηρεσία μεταφοράς δεδομένων διασφαλίζει ότι η ροή δεδομένων – όπου μέσω μιας διαδικασίας «διαβάζεται» από τον αποθηκευτικό χώρο λήψης του TCP – μένει ανεπηρέαστη, χωρίς κενά, χωρίς επαναλήψεις και σε ακολουθία. Για παράδειγμα, η ροή δεδομένων που διαβάζεται είναι ακριβώς ίδια με την ροή δεδομένων που είχε σταλεί από το ένα σύστημα στο άλλο άκρο της σύνδεσης. Σ' αυτή την υποενοότητα παρέχουμε μια άτυπη επισκόπηση για το πώς το TCP παρέχει την αξιόπιστη υπηρεσία μεταφοράς δεδομένων .

## *Αναμεταδόσεις (retransmissions)*

Η αναμετάδοση χαμένων και αλλοιωμένων δεδομένων είναι ζωτικής σημασίας για την αξιόπιστη υπηρεσία μεταφοράς δεδομένων. Το TCP παρέχει αυτή την υπηρεσία χρησιμοποιώντας θετικές επιβεβαιώσεις και χρονοδιακόπτες. Το TCP επιβεβαιώνει δεδομένα τα οποία είχαν ληφθεί ορθά και αναμεταδίδει πακέτα όταν τα πακέτα ή οι αντίστοιχες επιβεβαιώσεις τους υπολογίζεται ότι έχουν χαθεί ή αλλοιωθεί. Όπως και στην περίπτωση του πρωτοκόλλου αξιόπιστης μεταφοράς δεδομένων (rtd3.0) , το TCP δεν μπορεί να «πει» από μόνο του με σιγουριά αν ένα πακέτο ή η επιβεβαίωση του έχει χαθεί, αλλοιωθεί ή έχει καθυστερήσει υπερβολικά. Σε όλες τις περιπτώσεις η απάντηση του TCP είναι η ίδια: «αναμετέδωσε το εν λόγω πακέτο». Το TCP επίσης χρησιμοποιεί διοχέτευση (pipelining) που επιτρέπει στον αποστολέα να έχει πολλαπλές μεταδόσεις πακέτων που δεν έχουν ακόμα επιβεβαιωθεί σε οποιοδήποτε χρονικό σημείο. Η διοχέτευση μπορεί να βελτιώσει σημαντικά το throughput (ρυθμό απόδοσης) όταν η αναλογία της καθυστέρησης του μεγέθους πακέτου είναι μικρή. Ο συγκεκριμένος αριθμός ανεπιβεβαίωτων πακέτων που μπορεί να έχει ένας αποστολέας είναι καθορισμένος από τους μηχανισμούς ελέγχου ροής και ελέγχου συμφόρησης του TCP. Στο παρών σημείο απλά πρέπει να προσέξουμε ότι ο αποστολέας μπορεί να κάνει πολλαπλές εκπομπές ανεπιβεβαίωτων πακέτων.

```
/* Υποθέτουμε ότι ο αποστολέας δε δεσμεύεται από τον έλεγχο ροής ή συμφόρησης του TCP ,  
ότι τα δεδομένα από πάνω είναι μικρότερα σε μέγεθος από το MSS και ότι η μεταφορά δεδομένων είναι μόνο προς  
μια κατεύθυνση */
```

```
sendbase = initial_sequence number  
nextseqnum = initial_sequence number
```

```
loop (forever) {  
    switch(event)
```

```
    event: data received from application above  
        create TCP segment with sequence number nextseqnum  
        start timer for segment nextseqnum  
        pass segment to IP  
        nextseqnum = nextseqnum + length(data)
```

```
    event: timer timeout for segment with sequence number y  
        retransmit segment with sequence number y  
        compute new timeout interval for segment y  
        restart timer for sequence number y
```

```
    event: ACK received, with ACK field value of y  
        if (y > sendbase) { /* αθροιστικό ACK όλων των δεδομένων μέχρι και το y */  
            cancel all timers for segments with sequence numbers < y  
            sendbase = y  
        }
```

```
        else { /* ένα διπλότυπο ACK για ήδη επιβεβαιωμένο πακέτο */  
            increment number of duplicate ACKs received for y  
            if (number of duplicate ACKs received for y == 3) {  
                /* γρήγορη αναμετάδοση TCP */  
                resend segment with sequence number y  
                restart timer for segment y  
            }  
        }
```

```
    } /* τέλος του loop forever */
```

Σχήμα 3.5-5 : ένας απλοποιημένος αποστολέας TCP

Το σχήμα 3.5-5 δείχνει τα τρία κύρια γεγονότα που σχετίζονται με τη μετάδοση – αναμετάδοση σε ένα απλουστευμένο αποστολέα TCP. Ας μελετήσουμε μια σύνδεση TCP μεταξύ των host A και host B και ας επικεντρωθούμε στη ροή δεδομένων από τον host A στον host B. Στον host A-αποστολέα, το TCP είναι περασμένο σε δεδομένα επιπέδου εφαρμογής τα οποία χωρίζει σε πακέτα και μετά τα μεταφέρει στο IP. Η μεταφορά των δεδομένων από την εφαρμογή στο TCP και η ακόλουθη πλαισίωση και εκπομπή ενός πακέτου είναι το πρώτο σημαντικό γεγονός που πρέπει ένας αποστολέας TCP να χειριστεί. Κάθε φορά που το TCP «ελευθερώνει» ένα πακέτο στον IP, ενεργοποιείται ένα χρονόμετρο γι' αυτό. Αν αυτή η προθεσμία λήξει, ενεργοποιείται μια εντολή διακοπής στον host A. Το TCP «απαντάει» σ' αυτό το γεγονός (time-out event), αναμεταδίδοντας το πακέτο που προκάλεσε αυτή τη διακοπή (timeout) και αυτό είναι το δεύτερο κύριο γεγονός που πρέπει ένας TCP αποστολέας να διαχειριστεί.

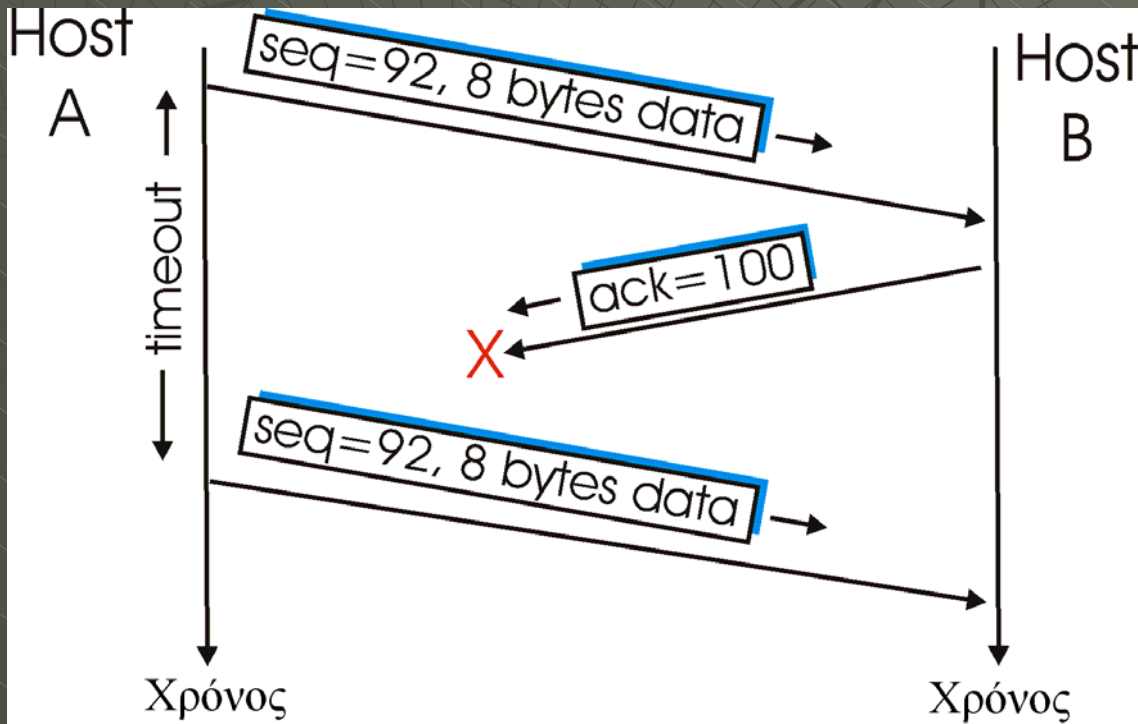
Το τρίτο σημαντικό γεγονός που πρέπει να διαχειριστεί ο αποστολέας TCP είναι η άφιξη ενός πακέτου επιβεβαίωσης (ACK) από τον παραλήπτη (πιο συγκεκριμένα ένα πακέτο που παρέχει μια έγκυρη τιμή πεδίου ACK). Εδώ ο αποστολέας TCP πρέπει να καθορίσει κατά πόσο το συγκεκριμένο ACK είναι το πρώτο ACK ενός πακέτου του οποίου ο αποστολέας δεν έχει ακόμη λάβει επιβεβαίωση και όχι διπλότυπη επιβεβαίωση (duplicate ACK) που επανεπιβεβαιώνει ένα πακέτο για το οποίο ο αποστολέας έχει λάβει νωρίτερα μια επιβεβαίωση. Στην περίπτωση της άφιξης της πρώτης επιβεβαίωσης (first time ACK), ο αποστολέας τώρα γνωρίζει ότι όλα τα δεδομένα μέχρι το byte που έχει επιβεβαιωθεί, έχουν ληφθεί ορθώς στον παραλήπτη. Έτσι ο αποστολέας μπορεί να ενημερώσει την μεταβλητή του TCP, η οποία ανιχνεύει τον αριθμό ακολουθίας του τελευταίου byte, το οποίο έχουν ληφθεί ορθά και με σωστή σειρά από παραλήπτη.

Για να κατανοήσουμε την αντίδραση του αποστολέα σε ένα διπλότυπο επιβεβαίωσης πρέπει να καταλάβουμε γιατί ο παραλήπτης στέλνει ένα duplicate ACK εξ' αρχής. Ο πίνακας 3.5.1 συνοψίζει την τακτική που ακολουθεί ο δέκτης TCP στη «δημιουργία» επιβεβαιώσεων. Όταν ο παραλήπτης TCP λάβει ένα πακέτο μ' έναν αριθμό ακολουθίας που είναι μεγαλύτερος από τον επόμενο αναμενόμενο και σε σειρά αριθμό ακολουθίας ανιχνεύει ένα κενό στην ροή των δεδομένων (για παράδειγμα ένα χαμένο πακέτο). Εφόσον το TCP δεν χρησιμοποιεί αρνητικές επιβεβαιώσεις, ο παραλήπτης δεν μπορεί να στείλει μια ακριβή αρνητική επιβεβαίωση πίσω στον αποστολέα. Αντίθετα, απλά επανεπιβεβαιώνει (για παράδειγμα παράγει ένα duplicate ACK) για το τελευταίο στην σειρά byte από τα δεδομένα που έχει ληφθεί. Αν ο αποστολέας TCP λάβει τρεις duplicate ACK για τα ίδια δεδομένα, αυτό το λαμβάνει ως ένδειξη ότι το πακέτο που ακολουθεί το πακέτο δηλαδή αυτό που έχει επιβεβαιωθεί τρεις φορές, έχει χαθεί. Σ' αυτήν την περίπτωση το TCP πραγματοποιεί μια γρήγορη αναμετάδοση (fast retransmit) του χαμένου πακέτου προτού ο χρόνος του πακέτου λήξει.

Γεγονός	Ενέργεια παραλήπτη TCP
<ul style="list-style-type: none"> <li>♦ Αφιξη πακέτου σε σωστή σειρά και με τον αναμενόμενο αριθμό ακολουθίας. Όλα τα δεδομένα μέχρι τον αναμενόμενο αριθμό ακολουθίας έχουν ήδη επιβεβαιωθεί. Χωρίς κενά στα ληφθέντα δεδομένα.</li> </ul>	<ul style="list-style-type: none"> <li>♦ Καθυστερημένο ACK. Αναμονή 500ms για άφιξη άλλου πακέτου σε σειρά. Αν το επόμενο σε σειρά πακέτο δε φτάσει σε αυτό το διάστημα στείλε ACK.</li> </ul>
<ul style="list-style-type: none"> <li>♦ Άφιξη πακέτου σε σωστή σειρά και με τον αναμενόμενο αριθμό ακολουθίας. Ένα άλλο πακέτο που βρίσκεται σε σωστή σειρά περιμένει αναμετάδοση του ACK.</li> </ul>	<ul style="list-style-type: none"> <li>♦ Στέλνει αμέσως ένα μοναδικό αθροιστικό ACK (cumulative ACK) επιβεβαιώνοντας και τα δύο πακέτα που βρίσκονται σε σειρά.</li> </ul>
<ul style="list-style-type: none"> <li>♦ Άφιξη πακέτου εκτός σειράς με μεγαλύτερο απ' τον αναμενόμενο αριθμό ακολουθίας. Ανιχνεύθηκαν κενά.</li> </ul>	<ul style="list-style-type: none"> <li>♦ Στέλνει αμέσως διπλότυπο ACK δηλώνοντας τον αριθμό ακολουθίας του επόμενου αναμενόμενου byte.</li> </ul>
<ul style="list-style-type: none"> <li>♦ Άφιξη πακέτου το οποίο εν μέρει ή ολοκληρωτικά γεμίζει τα κενά των ληφθέντων δεδομένων.</li> </ul>	<ul style="list-style-type: none"> <li>♦ Στέλνει αμέσως ACK δεδομένου ότι το πακέτο ξεκινά στην αρχή του κενού.</li> </ul>

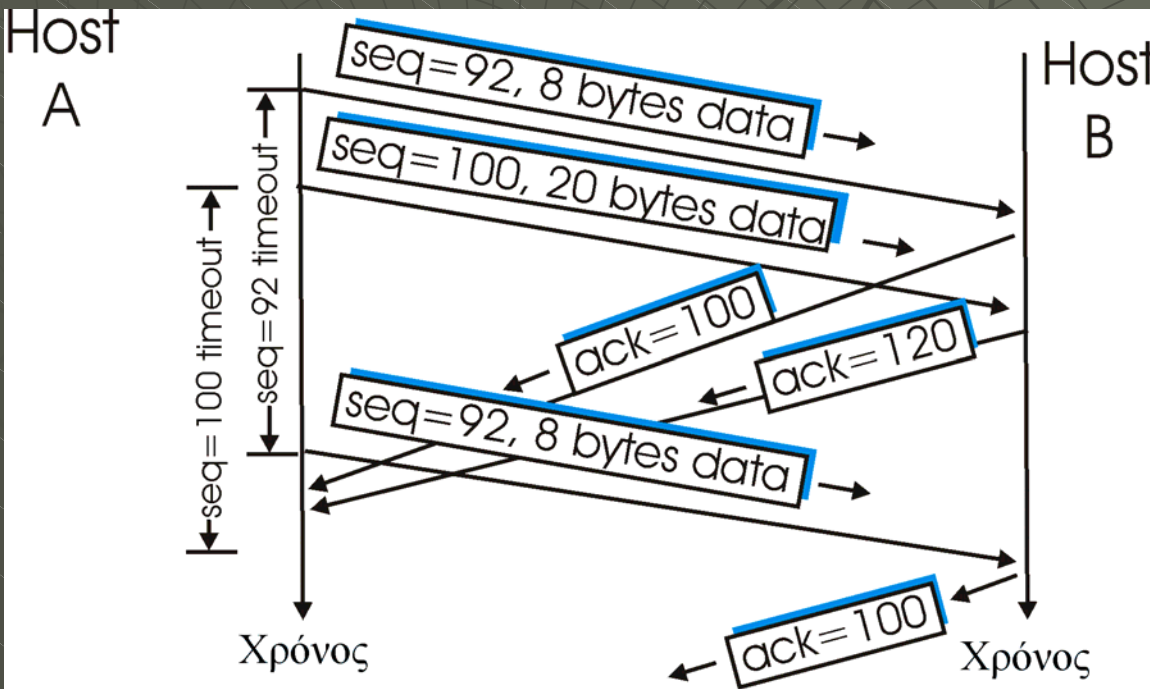
## Μερικά ενδιαφέροντα σενάρια

Τελειώνουμε αυτή τη συζήτηση παρουσιάζοντας μερικά απλά «σενάρια». Το σχήμα 3.5-6 απεικονίζει ένα σενάριο στο οποίο ο host A στέλνει ένα πακέτο στον host B. Ας υποθέσουμε ότι αυτό το πακέτο έχει αριθμό ακολουθίας 92 και περιέχει 8 bytes δεδομένων. Έχοντας στείλει αυτό το πακέτο ο host A αναμένει ένα πακέτο από τον B με αριθμό επιβεβαίωσης 100. Μολονότι το πακέτο του A παραλαμβάνεται από τον B, η επιβεβαίωση από τον B στον A χάνεται. Σε αυτήν τη περίπτωση ο χρόνος λήγει και ο host A αναμεταδίδει το ίδιο πακέτο. Φυσικά όταν ο host B λάβει την αναμετάδοση θα καταλάβει ότι τα bytes του πακέτου τα οποία έχουν ήδη αποθηκευτεί στον αποθηκευτικό χώρο λήψης επανεγράφονται (duplicate bytes). Γι' αυτό το λόγο το TCP στο host B θα απορρίψει τα bytes του πακέτου αναμετάδοσης.



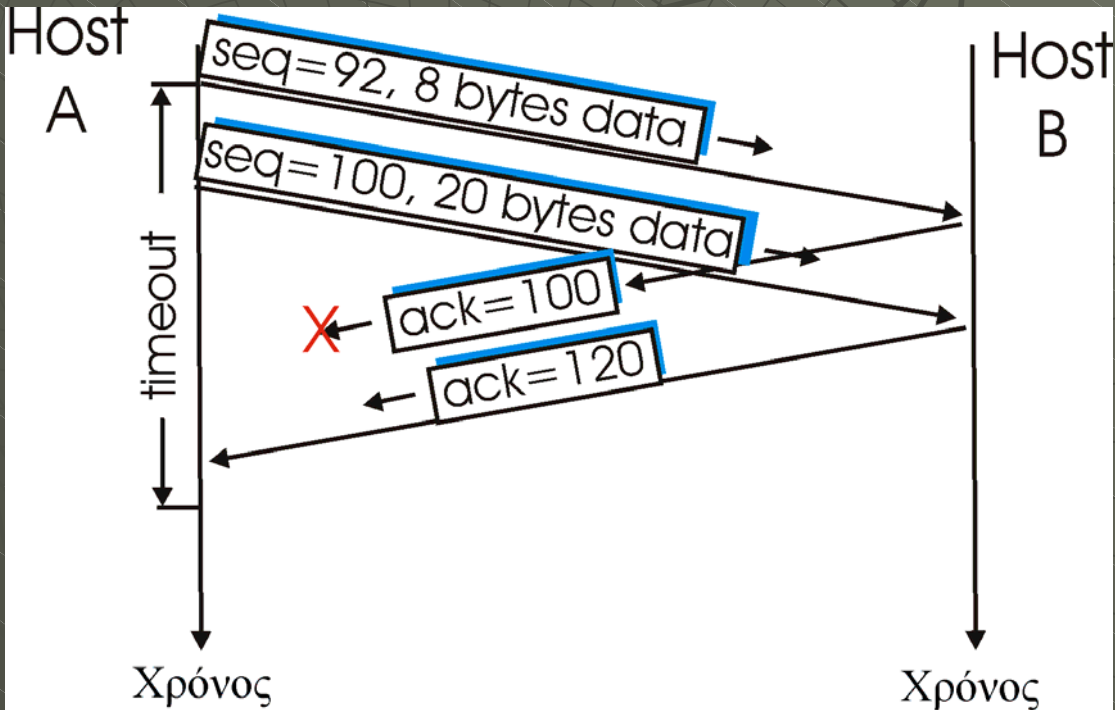
Σχήμα 3.5-6  
(αναμετάδοση λόγω χαμένης επιβεβαίωσης)

Στο δεύτερο σενάριο ο host A στέλνει δύο πακέτα συνεχόμενα. Το πρώτο έχει αριθμό ακολουθίας 92 και 8 bytes δεδομένων και το δεύτερο έχει αριθμό ακολουθίας 100 και 20 bytes δεδομένων. Υποθέτουμε ότι και τα δύο πακέτα φθάνουν άθικτα στο B και ο B στέλνει 2 χωριστές επιβεβαιώσεις για κάθε ένα από αυτά. Η πρώτη από αυτές τις επιβεβαιώσεις έχει αριθμό επιβεβαίωσης 100 και η δεύτερη 120. Ας υποθέσουμε ότι καμία από αυτές τις επιβεβαιώσεις δεν έχει φθάσει στον host A μέχρι το τερματισμό του χρόνου του πρώτου πακέτου. Όταν τελειώσει ο χρόνος, ο host A επαναστέλνει το πρώτο πακέτο με αριθμό ακολουθίας 92. Μπορούμε να αναρωτηθούμε τώρα, μήπως ο A επαναστέλνει και το δεύτερο πακέτο; Σύμφωνα με τους κανόνες που αναφέραμε παραπάνω ο host A επαναστέλνει το πακέτο μόνο εάν ο χρόνος τερματιστεί πριν την άφιξη μιας επιβεβαίωσης με αριθμό επιβεβαίωσης 120 ή μεγαλύτερη. Έτσι, όπως φαίνεται και στο σχήμα 3.5-7, αν η δεύτερη επιβεβαίωση δεν χαθεί και φθάσει πριν τον τερματισμό του χρόνου του δεύτερου πακέτου, ο A δεν επαναστέλνει το δεύτερο πακέτο.



Σχήμα 3.5-7  
 (Το πακέτο δεν επανεκπέμπεται επειδή η επιβεβαίωση έρχεται πριν το timeout)

Σ' ένα τρίτο και τελευταίο σενάριο, υποθέτουμε ότι ο host A στέλνει τα δύο πακέτα ακριβώς όπως και στο δεύτερο παράδειγμα. Η επιβεβαίωση του πρώτου «χάνεται» στο δίκτυο, αλλά ακριβώς προτού τερματιστεί ο χρόνος του, ο host A λαμβάνει μια επιβεβαίωση με αριθμό επιβεβαίωσης 120. Μ' αυτό τον τρόπο ο host A ενημερώνεται ότι ο host B έχει λάβει τα πάντα μέχρι το byte 119· έτσι ο host A δεν επαναστέλνει κανένα από τα δύο πακέτα. Αυτό το σενάριο παρουσιάζεται στο σχήμα 3.5-8



Σχήμα 3.5-8  
(Μια αθροιστική επιβεβαίωση αποφεύγει την επανεκπομπή του πρώτου πακέτου)

Στην προηγούμενη ενότητα χαρακτηρίσαμε το TCP ως “Go-Back-N Style Protocol” . Αυτό συμβαίνει γιατί οι επιβεβαιώσεις είναι αύξουσες και ορθώς ληφθείσες αλλά τα πακέτα (τα οποία δεν βρίσκονται σε ορθή σειρά) δεν επιβεβαιώνονται χωριστά από τον παραλήπτη. Συνεπώς, ο αποστολέας TCP χρειάζεται μόνο να διατηρήσει τον ελάχιστο αριθμό ακολουθίας ενός εκπεμπόμενου αλλά ανεπιβεβαίωτου byte (sendbase) και ο αριθμός ακολουθίας του επόμενου byte να σταλεί (next seqnum). Αλλά ο αναγνώστης πρέπει να λάβει υπόψη ότι παρολο που το μέρος αξιόπιστης μεταφοράς δεδομένων θυμίζει “Go-Back-N” δεν είναι κατ’ουδένα λόγο μια καθαρή εκτέλεση του “Go-Back-N”. Για να δούμε ότι υπάρχουν μερικές εμφανείς διαφορές μεταξύ του TCP και του “Go-Back-N”, σκεφθείτε τι συμβαίνει όταν ο αποστολέας στέλνει μια ακολουθία τμημάτων 1,2,...,N και όλα τα πακέτα φθάνουν με σειρά, χωρίς σφάλμα στον παραλήπτη. Περαιτέρω, ας υποθέσουμε ότι η επιβεβαίωση για το πακέτο  $n < N$  χάνεται, αλλά οι υπολειπόμενες  $n-1$  επιβεβαιώσεις φθάνουν στον αποστολέα πριν λήξει διαδοχικά η προθεσμία τους (timeouts). Σ’ αυτό το παράδειγμα, το “Go-Back-N” θα αναμετέδιδε όχι μόνο το πακέτο  $n$ , αλλά επίσης και όλες τις υποακολουθίες πακέτων  $n+1$ ,  $n+2$ , ...,  $N$ . Απεναντίας το TCP θα αναμετέδιδε ονομαστικά το πολύ ένα πακέτο, το πακέτο  $n$ . Επίσης το TCP δεν θα αναμετέδιδε για κανένα λόγο το πακέτο  $n$ , αν η επιβεβαίωση για το  $n+1$  έφθανε προτού λήξει ο χρόνος για το  $n$ .

Πρόσφατα υπήρξαν πολλές προτάσεις [[RFC 2018](#), [Fall 1996](#), [Mathis 1996](#)] για να επεκταθεί το όλο σχέδιο επιβεβαιώσεων του TCP και να μοιάζει περισσότερο σε ένα επιλεκτικό, επαναληπτικό πρωτόκολλο. Η κύρια ιδέα σ’ αυτές τις προτάσεις είναι να παρέχει στον αποστολέα ακριβείς πληροφορίες για το ποιά πακέτα έχουν ληφθεί από αυτόν και ποια αγνοούνται ακόμη από τον παραλήπτη.

### 3.5.6. Έλεγχος Ροής (Flow Control)

Το TCP υποστηρίζει μια υπηρεσία ελέγχου ροής στις λειτουργίες του, εξαλείφοντας την πιθανότητα υπερχειλίσισης του αποθηκευτικού χώρου λήψης από τον αποστολέα, δηλαδή ο έλεγχος ροής είναι μια υπηρεσία ταύτισης ταχύτητας, ταυτίζοντας τον ρυθμό με τον οποίο ο αποστολέας στέλνει με τον ρυθμό με τον οποίο η λειτουργία λήψης διαβάζει. Όπως σημειώθηκε προηγουμένως, ο αποστολέας TCP μπορεί επίσης να μπλοκαριστεί λόγω συμφόρησης μέσα στο δίκτυο IP· αυτή η μορφή ελέγχου του αποστολέα αναφέρεται ως έλεγχος συμφόρησης. Ενώ οι ενέργειες που γίνονται από τον έλεγχο ροής και συμφόρησης είναι παρόμοιες (μπλοκάρισμα του αποστολέα), είναι φανερό ότι έχουν δημιουργηθεί για εντελώς διαφορετικούς λόγους. Δυστυχώς πολλοί συγγραφείς χρησιμοποιούν τον όρο εναλλακτικά και ένας αναγνώστης που είναι γνώστης του θέματος πρέπει να είναι προσεκτικός για να μπορεί να ξεχωρίζει τις δύο περιπτώσεις. Θα αναφέρουμε τώρα πώς το TCP παρέχει την υπηρεσία ελέγχου ροής.

Το TCP παρέχει έλεγχο ροής με το να διατηρεί ο αποστολέας μια μεταβλητή που ονομάζεται «**παράθυρο λήψης**» (receive window). Ανεπίσημα το receive window χρησιμοποιείται για να δώσει στον αποστολέα μια ιδέα για το πόσος αποθηκευτικός χώρος είναι ελεύθερος στον παραλήπτη (free buffer space). Σε μια σύνδεση full duplex ο αποστολέας διατηρεί ένα ξεχωριστό παράθυρο λήψης στην κάθε πλευρά της σύνδεσης. Το receive window είναι δυναμικό, δηλαδή αλλάζει κατά τη διάρκεια της σύνδεσης. Ας μελετήσουμε το receive window στο αρχείο μεταφοράς. Υποθέτουμε ότι ο host A στέλνει ένα μεγάλο αρχείο στον host B μέσω μιας TCP σύνδεσης. Ο host B δημιουργεί-αναθέτει ένα receive buffer γι'αυτήν την σύνδεση. Το μέγεθος του συμβολίζεται με RcvBuffer. Κάθε τόσο η διαδικασία λειτουργίας στον host B διαβάσει από το buffer.

Ορίζουμε τις επόμενες μεταβλητές:

**LastByteRead** = ο αριθμός του τελευταίου byte στη ροή δεδομένων που διαβάστηκε από τον προσωρινό χώρο αποθήκευσης (buffer) από τη διαδικασία εφαρμογής του B.

**LastByteRcvd** = ο αριθμός του τελευταίου byte στη ροή δεδομένων ο οποίος έχει ληφθεί από το δίκτυο και έχει τοποθετηθεί στον προσωρινό αποθηκευτικό χώρο λήψης στο B.

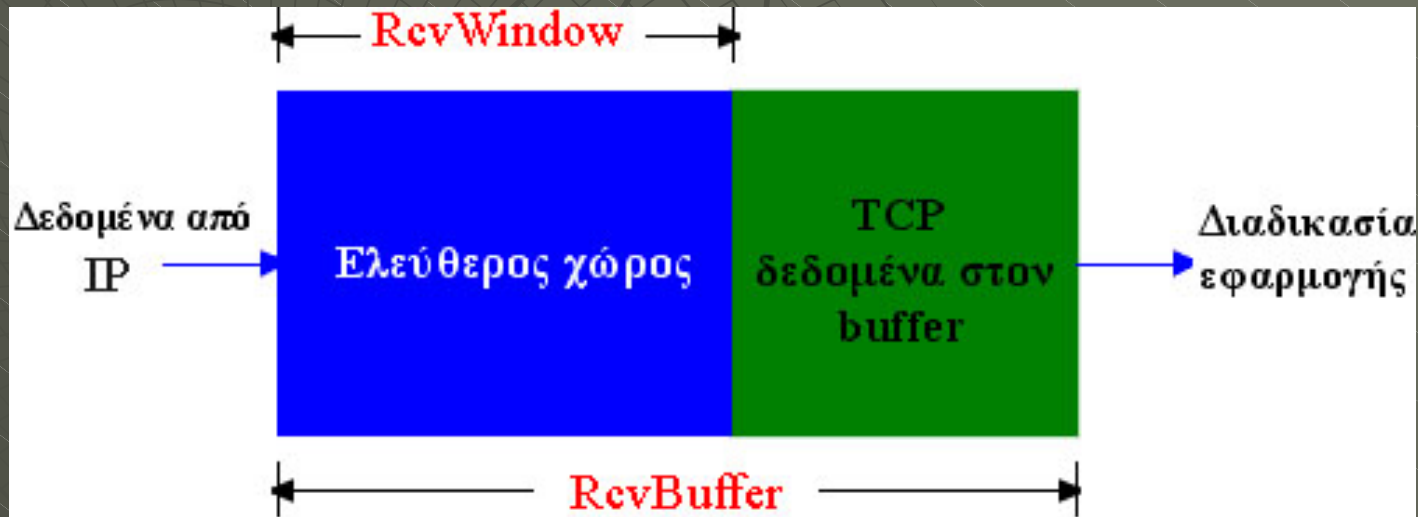
Επειδή δεν επιτρέπεται το TCP να υπερχειλίσει τον καθορισμένο προσωρινό χώρο αποθήκευσης πρέπει να ισχύει αυτό:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

Το παράθυρο λήψης που συμβολίζεται RcvWindow, τίθεται στο πλήθος του εφεδρικού δωματίου στον προσωρινό χώρο αποθήκευσης.

$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

Επειδή το εφεδρικό δωμάτιο αλλάζει με το χρόνο, το RcvWindow είναι δυναμικό. Η μεταβλητή RcvWindow παρουσιάζεται στην εικόνα 3.5-9.



Σχήμα 3.5-9 (Το παράθυρο και ο buffer λήψης)

Πώς μια σύνδεση χρησιμοποιεί τη μεταβλητή RcvWindow για να παρέχει την υπηρεσία ελέγχου ροής; Ο host B ενημερώνει το host A για το πόσος εφεδρικός χώρος υπάρχει στο buffer της σύνδεσης τοποθετώντας την τρέχουσα τιμή του RcvWindow στο πεδίο παραθύρου κάθε πακέτου που στέλνει στον A. Αρχικά ο host B ορίζει το  $RcvWindow = RcvBuffer$ . Σημειώνουμε ότι για να επιτευχθεί αυτό, ο host B πρέπει να παρακολουθεί αρκετές αλλά συγκεκριμένες μεταβλητές σύνδεσης. Ο host A με τη σειρά του παρακολουθεί δύο μεταβλητές, τη LastByteSent και τη LastByteAcked, των οποίων η σημασία είναι προφανής. Ας σημειώσουμε ότι η διαφορά μεταξύ των 2 μεταβλητών είναι το πλήθος των ανεπιβεβαίωτων δεδομένων που ο A έχει εισάγει στη σύνδεση. Διατηρώντας το πλήθος των ανεπιβεβαίωτων δεδομένων μικρότερο από την τιμή του RcvWindow, ο host A επιτυγχάνει να μην υπερχειλίσει ο buffer λήψης στον host B. Έτσι, ο host A σιγουρεύει ότι καθ'όλη τη διάρκεια σύνδεσης ισχύει:

$$LastByteSent - LastByteAcked \leq RcvWindow.$$

Υπάρχει ένα μικρό τεχνικό πρόβλημα μ'αυτή τη διαδικασία. Για να το δούμε, υποθέτουμε ότι ο buffer λήψης του host B "γεμίζει" έτσι ώστε το  $RcvWindow=0$  (εφόσον ενημερώνεται host A ότι  $RcvWindow=0$ ), υποθέτουμε επίσης ότι ο B δεν έχει τίποτα να στείλει στον A. Εφόσον η διαδικασία εφαρμογής στον host B αδειάζει τον buffer, το TCP δεν στέλνει νέα πακέτα με νέα  $RcvWindows$  στον host A. Το μόνο που θα κάνει το TCP είναι να στείλει ένα πακέτο στον host A αν έχει να στείλει δεδομένα ή μια επιβεβαίωση. Γι'αυτό το λόγο ο host A δεν ενημερώνεται ποτέ για το αν κάποιος χώρος ελευθερωθεί στο buffer λήψης του host B: ο host A μπλοκάρεται και δεν μπορεί να στείλει πλέον άλλα δεδομένα! Για να λυθεί αυτό το πρόβλημα, οι προδιαγραφές του TCP απαιτούν από τον host A να συνεχίσει να στέλνει πακέτα με δεδομένα ενός byte όταν το παράθυρο λήψης του B είναι 0. Αυτά τα πακέτα θα επιβεβαιωθούν από τον παραλήπτη. Τελικά ο buffer θα αρχίσει να αδειάζει και οι επιβεβαιώσεις θα περιέχουν μη μηδενικά  $RcvWindows$ .

Έχοντας περιγράψει την υπηρεσία ελέγχου ροής TCP, αναφέρουμε περιληπτικά εδώ ότι το UDP δεν υποστηρίζει έλεγχο ροής. Για να κατανοήσουμε το θέμα αυτό, θεωρούμε ότι στέλνουμε μια σειρά πακέτων του UDP από μια διαδικασία του host A σε μια διαδικασία του host B. Για μια χαρακτηριστική εφαρμογή UDP, η UDP θα επισυνάψει τα πακέτα (και για να είμαστε πιο ακριβείς, τα δεδομένα μέσα στα πακέτα) σε μία πεπερασμένου μεγέθους σειρά αναμονής που "προηγείται" της αντίστοιχης υποδοχής (για παράδειγμα η πόρτα για τη διαδικασία). Η διαδικασία διαβάζει ένα ολόκληρο πακέτο υποδοχής κάθε φορά από τη σειρά αναμονής. Εάν η διαδικασία δε διαβάσει αρκετά γρήγορα τα πακέτα από τη σειρά αναμονής, η σειρά αναμονής θα υπερχειλίσει (overflow) και κάποια πακέτα θα χαθούν (επειδή δεν υποστηρίζει έλεγχο ροής).

### 3.5.7. Round Trip Time και Timeout

Υπενθυμίζουμε ότι όταν ένας host στέλνει ένα πακέτο σε μια TCP σύνδεση, ξεκινάει μία **χρονομέτρηση** (timer). Αν η προθεσμία λήξει προτού ο host λάβει επιβεβαίωση για τα δεδομένα του πακέτου, ο host αναμεταδίδει το πακέτο. Το χρονικό διάστημα από τη στιγμή που ξεκινάει η χρονομέτρηση μέχρι τη στιγμή που λήγει ονομάζεται **timeout** του timer. Μία εύλογη ερώτηση είναι: πόσο μεγάλο μπορεί να είναι το timeout; Προφανώς, το timeout οφείλει να είναι μεγαλύτερο από το χρόνο roundtrip μιας σύνδεσης, δηλ. ο χρόνος από τη στιγμή που ένα πακέτο θα σταλεί έως ότου επιβεβαιωθεί, αλλιώς θα σταλούν περιττές αναμεταδόσεις. Αλλά το timeout δε μπορεί να είναι μεγαλύτερο από ένα χρόνο roundtrip· αλλιώς όταν ένα πακέτο χάνεται, το TCP δε θα αναμεταδώσει γρήγορα το πακέτο και γ'αυτό θα εμφανίσει σημαντικές καθυστερήσεις στη μεταφορά δεδομένων μέσα στη λειτουργία. Προτού συζητήσουμε το διάστημα του timeout λεπτομερέστερα ας ρίξουμε μια ματιά στο **RTT** (Roundtrip time). Τα παρακάτω βασίζονται στο TCP.

Το δείγμα RTT που συμβολίζεται ως **SampleRTT** για ένα πακέτο, είναι ο χρόνος από τη στιγμή που ένα πακέτο στέλνεται (περνάει στο IP) μέχρι την επιβεβαίωση της λήψης του πακέτου. Κάθε πακέτο που στέλνεται θα έχει το δικό του αντίστοιχο SampleRTT. Προφανώς οι τιμές του SampleRTT θα κυμαίνονται από πακέτο σε πακέτο λόγω της συμφόρησης στους routers και στο μεταβαλλόμενο φορτίο στα τελικά συστήματα. Λόγω αυτής της διακύμανσης κάθε δοθείσα τιμή SampleRTT μπορεί να μην ανταποκρίνεται στην πραγματικότητα. Για να υπολογίσουμε ένα τυπικό RTT είναι φυσικό να πάρουμε κάποιες μέσες SampleRTT τιμές. Το TCP διατηρεί μια μέση τιμή που ονομάζεται **EstimatedRTT** των SampleRTT τιμών. Έχοντας λάβει μια επιβεβαίωση και επιτυγχάνοντας ένα νέο SampleRTT το TCP ενημερώνει το EstimatedRTT σύμφωνα με τον παρακάτω τύπο:

$$\text{EstimatedRTT} = (1-x) \text{EstimatedRTT} + x\text{SampleRTT}$$

Ο παραπάνω τύπος είναι γραμμένος σε μια δηλωμένη γλώσσα προγραμματισμού - η καινούρια τιμή του EstimatedRTT είναι ένας «ζυγισμένος» συνδυασμός της προηγούμενης τιμής του EstimatedRTT και της καινούριας τιμής του SampleRTT. Στην περίπτωση που  $x=1$  ο παραπάνω τύπος γράφεται:

$$\text{EstimatedRTT} = .9\text{EstimatedRTT} + .1\text{SampleRTT}$$

Ας σημειωθεί ότι το EstimatedRTT είναι ένας «ζυγισμένος» μέσος όρος του SampleRTT. Όπως θα δούμε στην εργασία αυτός ο μέσος όρος ρίχνει περισσότερη βαρύτητα στα πρόσφατα δείγματα απ'ότι στα παλιά, αυτό είναι φυσικό εφόσον τα πιο πρόσφατα δείγματα αντανακλούν καλύτερα την παρούσα συμφόρηση στο δίκτυο. Μια τέτοια μέση τιμή καλείται exponential weighted moving average (EWMA). Η λέξη "Exponential" (εκθετικός) εμφανίζεται στην EWMA επειδή το «βάρος» ενός δοθέντος SampleRTT μειώνεται εκθετικά καθώς η ενημέρωση προχωράει

## *Ορίζοντας το timeout*

Το timeout πρέπει να οριστεί έτσι ώστε το timer να λήγει γρήγορα (για παράδειγμα πριν από την καθυστερημένη άφιξη ενός ACK του πακέτου) σε σπάνιες περιπτώσεις. Γι'αυτό είναι φυσικό να ορίσουμε το timeout ίσο με το EstimatedRTT συν κάποιο περιθώριο. Αυτό το περιθώριο πρέπει να είναι μεγάλο όταν υπάρχει μεγάλη διακύμανση στις τιμές του SampleRTT· θα πρέπει να είναι μικρό όταν υπάρχει μικρή διακύμανση. Το TCP ακολουθεί τον παρακάτω τύπο:

$$\text{Timeout} = \text{EstimatedRTT} + 4 * \text{Deviation}$$

όπου η **απόκλιση** (deviation) είναι ένας υπολογισμός στο πόσο το SampleRTT συνήθως αποκλίνει από το EstimatedRTT.

$$\text{Deviation} = (1-x) \text{Deviation} + x / \text{SampleRTT} - \text{EstimatedRTT}.$$

Σημειώνουμε ότι deviation είναι μία EWMA του πόσο ένα SampleRTT αποκλίνει από ένα EstimatedRTT. Αν οι τιμές του SampleRTT έχουν μικρή διακύμανση, τότε η απόκλιση είναι μικρή και το timeout είναι ελάχιστα μεγαλύτερο από το EstimatedRTT· από την άλλη, αν υπάρχει μεγάλη διακύμανση, η απόκλιση θα είναι μεγάλη και το timeout θα είναι πολύ μεγαλύτερο από το EstimatedRTT.

### 3.5.8. Διαχείριση Σύνδεσης TCP (TCP connection management)


Σ' αυτή την υποενότητα θα ρίξουμε μια καλύτερη ματιά στο πώς επιτυγχάνεται μια σύνδεση TCP και πώς καταρρέει. Παρόλο που αυτό το συγκεκριμένο θέμα μπορεί να μην φαίνεται και τόσο σημαντικό, η επίτευξη της TCP σύνδεσης μπορεί να μεγαλώσει σημαντικά τις αναμενόμενες καθυστερήσεις (για παράδειγμα όταν σερφάρουμε στο Web). Ας κοιτάξουμε τώρα πώς επιτυγχάνεται μια σύνδεση TCP. Υποθέτουμε ότι η διαδικασία που τρέχει στον έναν host επιθυμεί να ξεκινήσει μια σύνδεση με μια άλλη διαδικασία σε ένα άλλο host . Ο host ο οποίος ξεκινάει την διαδικασία καλείται client host ενώ ο άλλος host καλείται server host. Η διαδικασία εφαρμογής του πελάτη πρώτα ενημερώνει τον TCP client ότι επιθυμεί να πραγματοποιήσει μια σύνδεση σε μια εφαρμογή στον server. Αυτή η εντολή δίνεται από έναν client στην Java με τον παρακάτω τρόπο:

```
Socket clientSocket = newSocket("hostname", "portnumber");
```

Τότε το TCP στον client προβαίνει στην πραγματοποίηση μιας σύνδεσης TCP με το TCP του server με τον ακόλουθο τρόπο:

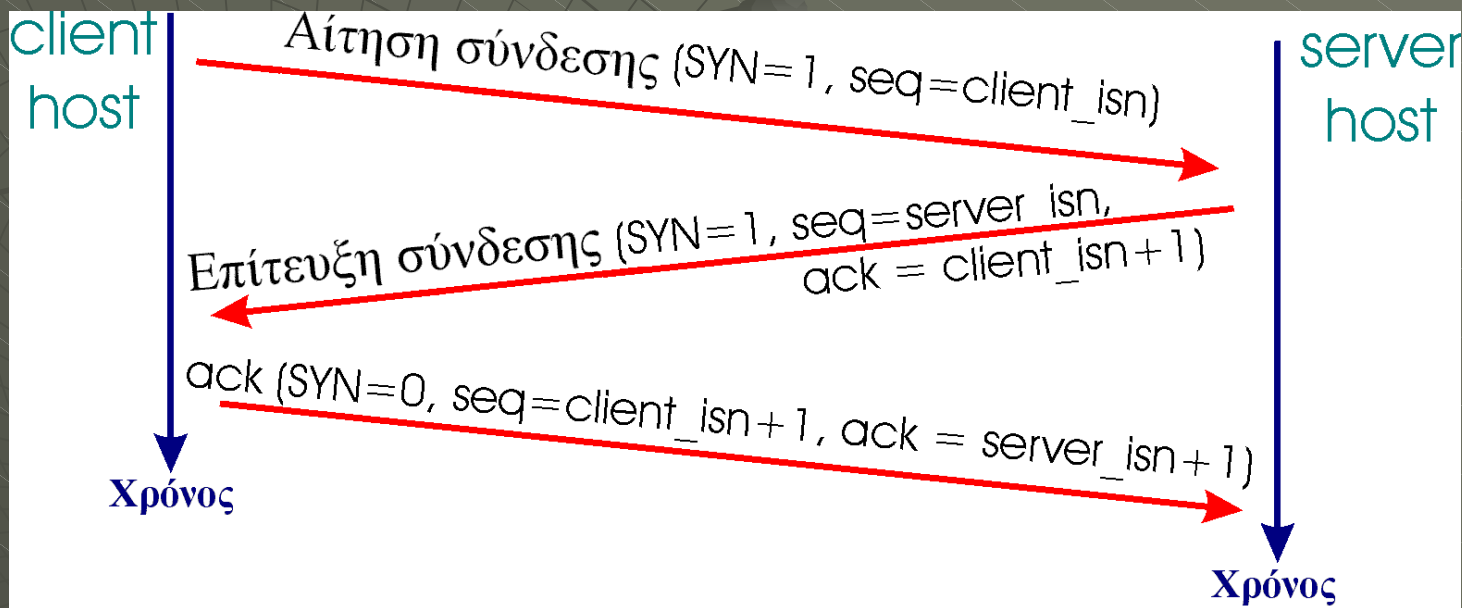
- Βήμα 1ο:** Το TCP του client πρώτα στέλνει ένα ξεχωριστό πακέτο TCP στον server του TCP. Αυτό το ξεχωριστό πακέτο δεν περιέχει καθόλου δεδομένα επιπέδου εφαρμογής. Όμως έχει τα bits σημαίας (flag bits) στην επικεφαλίδα του πακέτου γνωστά και ως SYN bit που είναι ορισμένα στην τιμή 1. Γι' αυτό το λόγο αυτό το ξεχωριστό πακέτο αναφέρεται ως SYN Segment. Επιπρόσθετα ο client επιλέγει έναν αρχικό αριθμό ακολουθίας (client\_isn) και τοποθετεί αυτόν τον αριθμό στο πεδίο αριθμού ακολουθίας του αρχικού TCP SYN πακέτου. Αυτό το πακέτο περιλαμβάνεται μέσα σε ένα IP datagram και στέλνεται στο internet.

•**Βήμα 2<sup>ο</sup>**: Αφού το IP datagram που περιέχει το TCP SYN πακέτο φτάσει στον server host (υποθέτοντας ότι όντως φθάνει) ο server εξάγει το TCP SYN πακέτο από το datagram, δεσμεύει τους TCP buffers και τις μεταβλητές στην σύνδεση και στέλνει ένα πακέτο επιβεβαίωσης της σύνδεσης στο TCP του client. Αυτό το πακέτο επιβεβαίωσης της σύνδεσης δεν περιέχει καθόλου δεδομένα επιπέδου εφαρμογής. Περιέχει όμως τρία σημαντικά κομμάτια πληροφοριών στο τμήμα της επικεφαλίδας. Πρώτον το SYNbit λαμβάνει την τιμή 1. Δεύτερον, το πεδίο επιβεβαίωσης του τμήματος επικεφαλίδας του TCP ορίζεται στο  $isn + 1$ . Τελικά, ο server επιλέγει τον δικό του αρχικό αριθμό ακολουθίας ( $server\_isn$ ) και τοποθετεί αυτή την τιμή στο πεδίο αριθμού ακολουθίας της επικεφαλίδας του πακέτου TCP. Αυτό το πακέτο που επιβεβαιώνει την σύνδεση με λίγα λόγια λέει “έλαβα το πακέτο SYN για να ξεκινήσω μια σύνδεση με τον αρχικό σου αριθμό ακολουθίας, τον  $client\_isn$ . Δέχομαι να πραγματοποιήσουμε αυτή την σύνδεση. Ο δικός μου αρχικός αριθμός ακολουθίας είναι  $server\_isn$ ”. Το πακέτο επίτευξης της σύνδεσης αναφέρεται μερικές φορές ως πακέτο SYNACK.



•**Βήμα 3<sup>ο</sup>**: Έχοντας λάβει το πακέτο επίτευξης της σύνδεσης, ο client δεσμεύει buffers και μεταβλητές για τη σύνδεση. Τότε ο client host στέλνει ακόμα ένα πακέτο στον server. Αυτό το τελευταίο πακέτο επιβεβαιώνει το πακέτο επιβεβαίωσης της σύνδεσης του server (αυτό ο client το επιτυγχάνει τοποθετώντας την τιμή  $server-isn+1$  στο πεδίο επιβεβαίωσης της επικεφαλίδας του πακέτου TCP). Το SYN bit ορίζεται ως 0, εφόσον έχει επιτευχθεί η σύνδεση.

Με την επίτευξη των παραπάνω τριών βημάτων, οι hosts του client και του server μπορούν να ανταλλάξουν πακέτα δεδομένων μεταξύ τους. Σε κάθε ένα από αυτά τα μελλοντικά πακέτα το SYN bit θα έχει την τιμή 0. Σημειώνουμε ότι για να επιτευχθεί η σύνδεση, τρία πακέτα στέλνονται μεταξύ των δύο hosts, όπως αναπαραστήθηκε στο σχήμα 3.5-10. Γι'αυτό το λόγο, αυτή η διαδικασία επίτευξης συχνά αναφέρεται ως «τριπλή χειραψία» (three-way handshake).

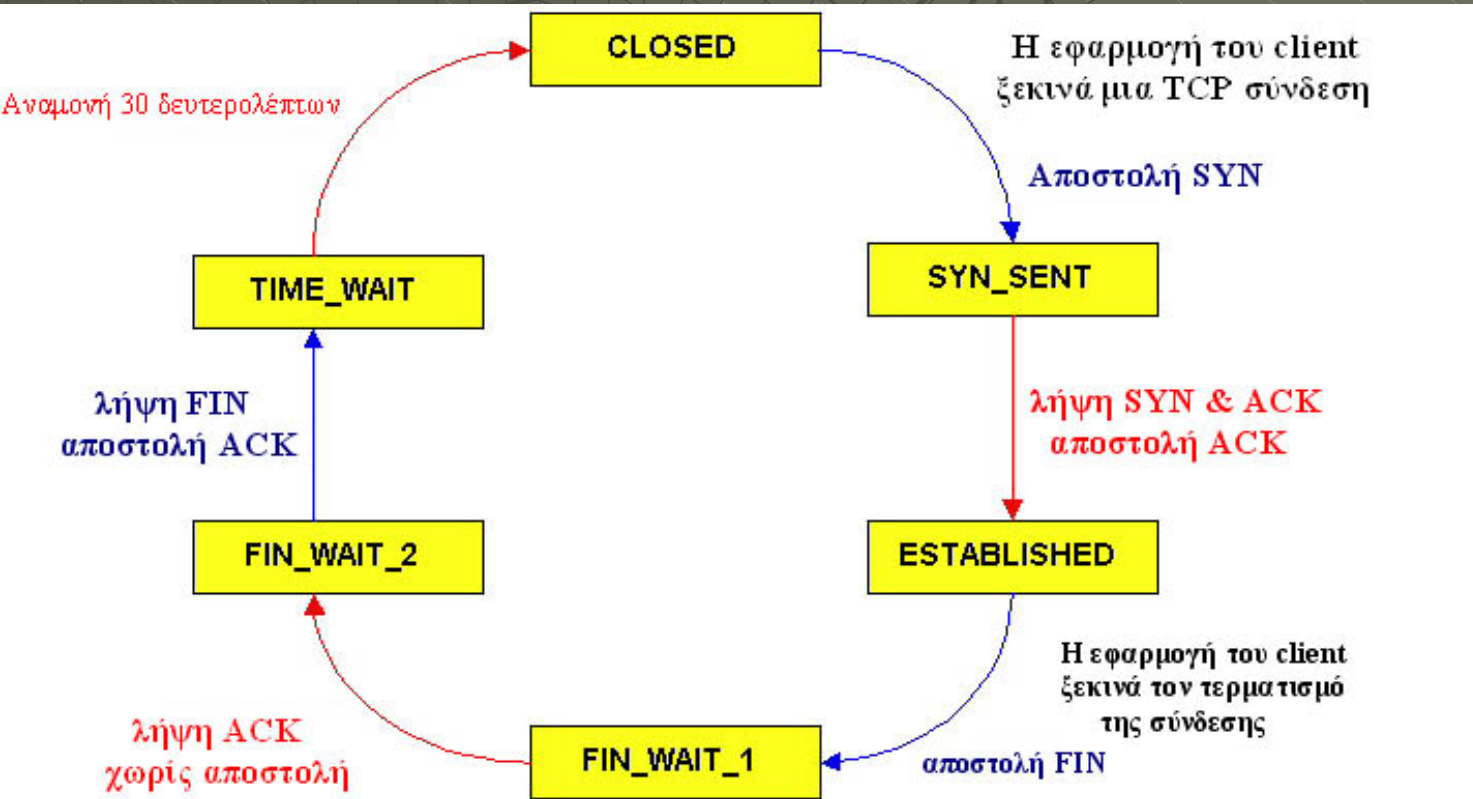


Σχήμα 3.5-10 (Τριπλή χειραψία TCP, ανταλλαγή πακέτων)

Όλα τα καλά πράγματα όμως κάποτε φθάνουν σε ένα τέλος. Και το ίδιο συμβαίνει στην TCP σύνδεση. Οποιαδήποτε από τις δύο διαδικασίες που συμμετέχουν σε μια TCP σύνδεση μπορεί να την τερματίσει. Όταν η σύνδεση τερματιστεί, οι πόροι (για παράδειγμα οι buffers και οι μεταβλητές) αποδεσμεύονται στους hosts. Για παράδειγμα, ας υποθέσουμε ότι ο client αποφασίζει να τερματίσει την σύνδεση. Η διαδικασία εφαρμογής του client εκδίδει μια εντολή τερματισμού. Αυτό υποχρεώνει τον client να στείλει ένα ξεχωριστό πακέτο TCP στην διαδικασία του server. Αυτό το ξεχωριστό πακέτο έχει ένα flag bit στην επικεφαλίδα του, το αποκαλούμενο **FINbit** που ορίζεται στο 1. Όταν ο server λάβει αυτό το πακέτο, στέλνει στον client ένα πακέτο επιβεβαίωσης ως απάντηση. Τότε ο server στέλνει το δικό του πακέτο «λήξης της σύνδεσης» (shutdown segment). Σ' αυτό το σημείο όλοι οι πόροι στους δύο hosts είναι πια αποδεσμευμένοι.

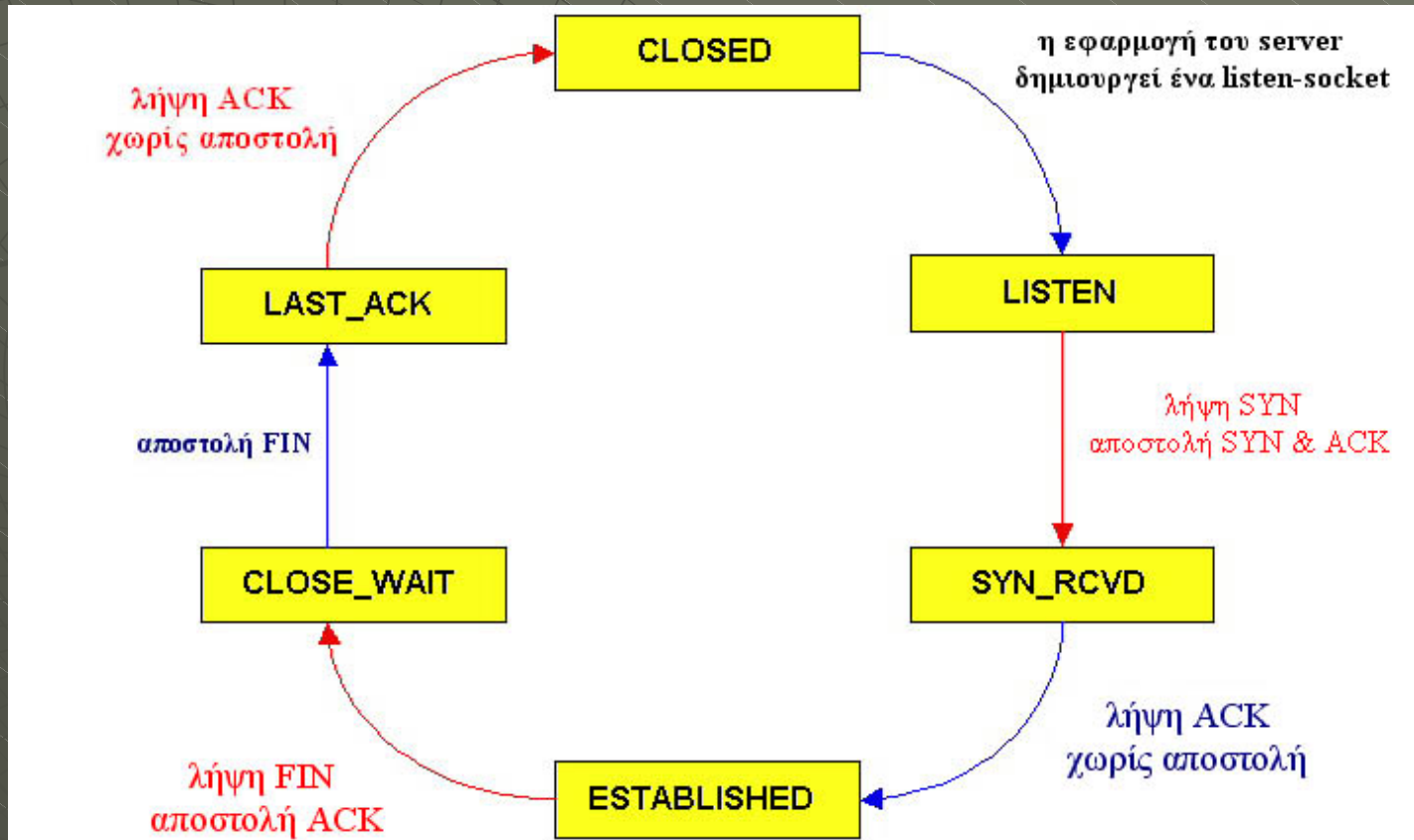
Κατά τη διάρκεια μιας TCP σύνδεσης, το πρωτόκολλο TCP που τρέχει σε κάθε host, αλλάζει καταστάσεις σε ποικίλες λειτουργίες του TCP (TCP states). Το σχήμα 3.5-11 απεικονίζει μια τυπική ακολουθία των **καταστάσεων του TCP** (TCP states) στις οποίες ο client μπορεί να εισέλθει. Το TCP στον client να στείλει ένα πακέτο SYN στο TCP του server. Αφού σταλεί το SYN πακέτο, ο client του TCP εισέρχεται στην κατάσταση **SYN\_SENT**. Ενώ βρίσκεται σε αυτή τη κατάσταση (**SYN\_STATE**), ο client TCP περιμένει ένα πακέτο από τον server TCP που περιέχει μια επιβεβαίωση για το προηγούμενο πακέτο του client, όπως και το SYN bit ορίζεται ως 1. Αφού ληφθεί ένα τέτοιο πακέτο, ο client TCP εισέρχεται στην κατάσταση **ESTABLISHED**. Όσο βρίσκεται σ' αυτήν την κατάσταση, ο TCP client μπορεί να στείλει και να λάβει πακέτα του TCP που περιέχουν payloads δεδομένων (δεδομένα που έχουν δημιουργηθεί από εφαρμογές).

Υποθέτουμε ότι η εφαρμογή του client θέλει να τερματίσει την σύνδεση. Αυτό αναγκάζει το TCP client να στείλει ένα πακέτο TCP που να έχει το FIN bit ορισμένο στο 1 και να μπει στην **FIN\_WAIT\_1** κατάσταση. Όσο βρίσκεται σε αυτή τη κατάσταση, ο client TCP περιμένει ένα πακέτο επιβεβαίωσης TCP από τον server. Όταν λάβει αυτό το πακέτο, ο client TCP εισέρχεται στη κατάσταση **FIN\_WAIT\_2**. Όσο βρίσκεται στην κατάσταση **FIN\_WAIT\_2**, ο client περιμένει άλλο ένα πακέτο από το server το οποίο θα έχει το FIN bit στην τιμή 1· αφού λάβει αυτό το πακέτο, ο client TCP επιβεβαιώνει το πακέτο του server και μπαίνει στην κατάσταση **TIME\_WAIT**. Αυτή η κατάσταση επιτρέπει τον TCP client να ξαναστείλει την τελευταία επιβεβαίωση σε περίπτωση που η ACK χαθεί. Η διάρκεια της κατάστασης **TIME\_WAIT** εξαρτάται από την εφαρμογή. Κάποιες συνηθισμένες τιμές είναι: 30 δευτερόλεπτα, 1 λεπτό και 2 λεπτά. Μετά την αναμονή, η σύνδεση κλείνει επίσημα και όλοι οι πόροι από την πλευρά του client (συμπεριλαμβανομένων των port numbers) ελευθερώνονται.



Σχήμα 3.5-11 (Μια αναπαράσταση της εναλλαγής των τυπικών καταστάσεων του client)

Το σχήμα 3.5-12 απεικονίζει μια σειρά καταστάσεων στις οποίες μπορεί να εισέλθει η πλευρά server του TCP· οι εναλλαγές είναι ξεκάθαρες. Σε αυτά τα δύο διαγράμματα εναλλαγής καταστάσεων, έχουμε δείξει μόνο πώς μια TCP σύνδεση επιτυγχάνεται και τελειώνει σε κανονικές συνθήκες. Δε θα περιγράψουμε τι συμβαίνει σε συγκεκριμένες προβληματικές καταστάσεις, όπως για παράδειγμα όταν και οι δύο πλευρές της σύνδεσης θέλουν να την τερματίσουν ταυτόχρονα.



Σχήμα 3.5-12 (Μια αναπαράσταση της εναλλαγής των τυπικών καταστάσεων ενός server)