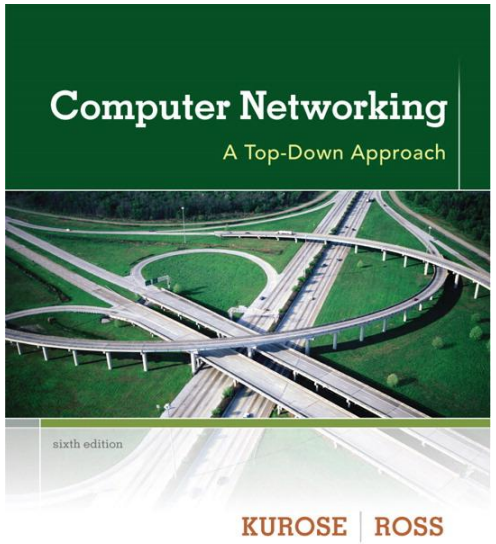
Κεφάλαιο 3
Επίπεδο Μεταφοράς



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Κεφάλαιο 3: Επίπεδο Μεταφοράς

Στόχοι:

- Κατανόηση των βασικών αρχών των υπηρεσιών του επιπέδου μεταφοράς. :
  - πολύπλεξη, αποπολύπλεξη
  - Αξιόπιστη μεταφορά δεδομένων
  - Έλεγχος ροής
  - Έλεγχος συμφόρησης
- Εκμάθηση σχετικά με τα πρωτόκολλα μεταφοράς του :
  - UDP: μη-συνδεσμοστραφής μεταφορά (connectionless transport)
  - TCP: συνδεσμοστραφής αξιόπιστη μεταφορά (connection-oriented reliable transport)
  - TCP έλεγχος συμφόρησης(congestion control)

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

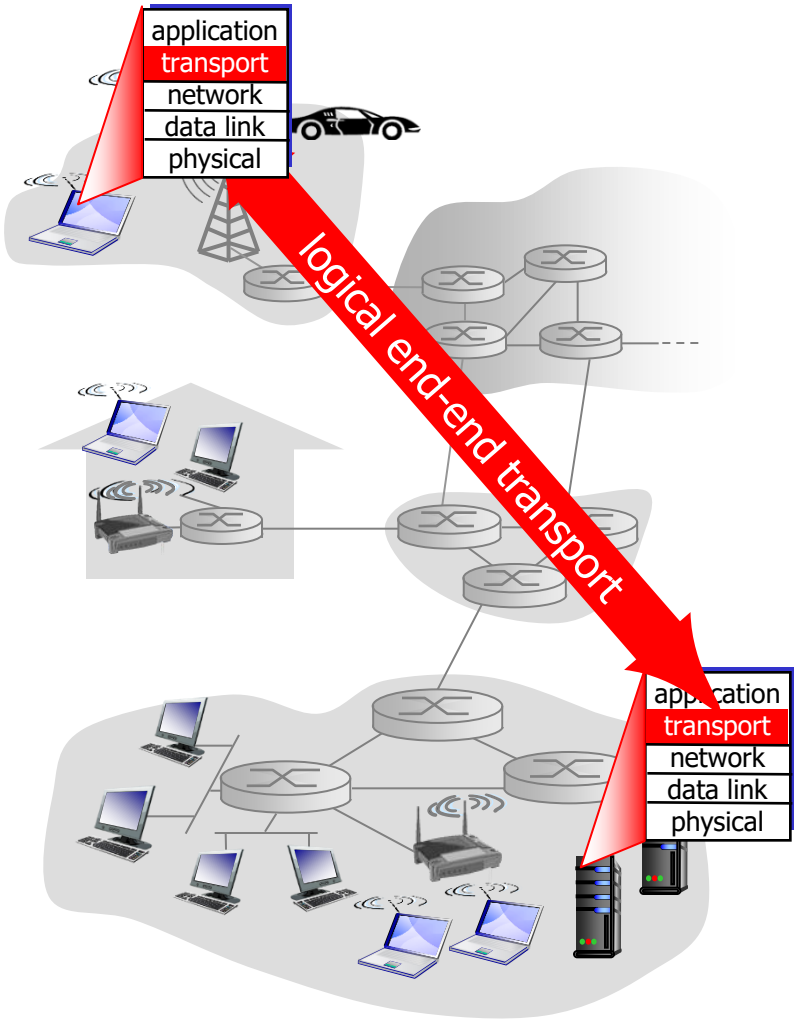
- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

Υπηρεσίες μεταφοράς και πρωτόκολλα

- Προσφέρουν *λογική επικοινωνία* μεταξύ διεργασιών που εκτελούνται σε διαφορετικούς υπολογιστές
- Τα πρωτόκολλα μεταφοράς τρέχουν στα ακραία συστήματα (end systems)
  - Πλευρά αποστολής: τεμαχίζει τα μηνύματα σε *segments*, και τα προωθεί στο επίπεδο δικτύου
  - Πλευρά λήψης: επανασυναρμολογεί τα segments σε μηνύματα και τα περνά στο επίπεδο εφαρμογών
- Υπάρχουν πάνω από ένα πρωτόκολλα μεταφοράς.
  - Internet: TCP και UDP



Επίπεδο μεταφοράς – επίπεδο δικτύου

- *network layer:* λογική επικοινωνία μεταξύ υπολογιστών
- *transport layer:* λογική επικοινωνία μεταξύ διεργασιών
  - Βασίζεται και βελτιώνει τις υπηρεσίες του επιπέδου δικτύου.

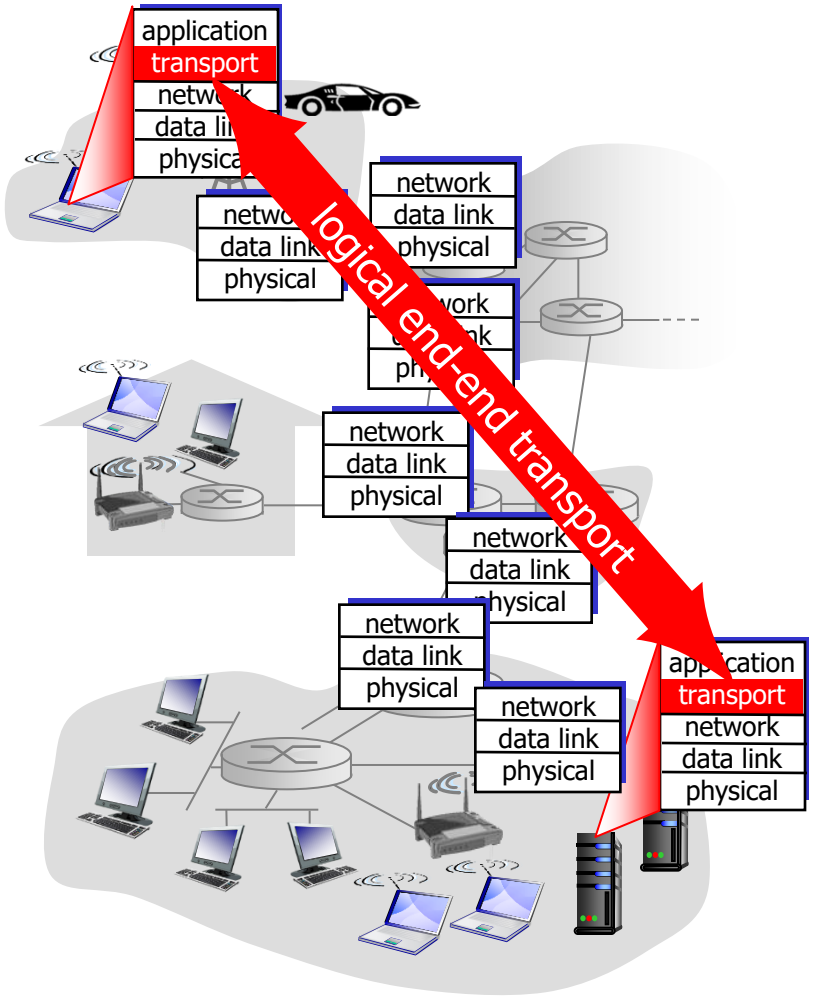
Αναλογία με σπίτι:

12 παιδιά στο σπίτι της Ann στέλνουν γράμματα σε 12 παιδιά στο σπίτι του Bill:

- hosts = σπίτια
- processes = παιδιά
- app messages = γράμματα στους φακέλους
- transport protocol = Ann και Bill που συλλέγουν και διανέμουν στα παιδιά μέσα στο σπίτι
- network-layer protocol = ταχυδρομική υπηρεσία.

Πρωτόκολλα επιπέδου μεταφοράς του Internet

- Αξιόπιστη, σε-σειρά παράδοση (TCP)
  - έλεγχος συμφόρησης (congestion control)
  - Έλεγχος ροής (flow control)
  - Δημιουργία σύνδεσης (connection setup)
- Μη αξιόπιστη, εκτός σειράς παράδοση (UDP)
  - Μια απλή επέκταση του IP
- Μη διαθέσιμες υπηρεσίες
  - Εγγυημένη καθυστέρηση
  - Εγγυημένο εύρος



Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

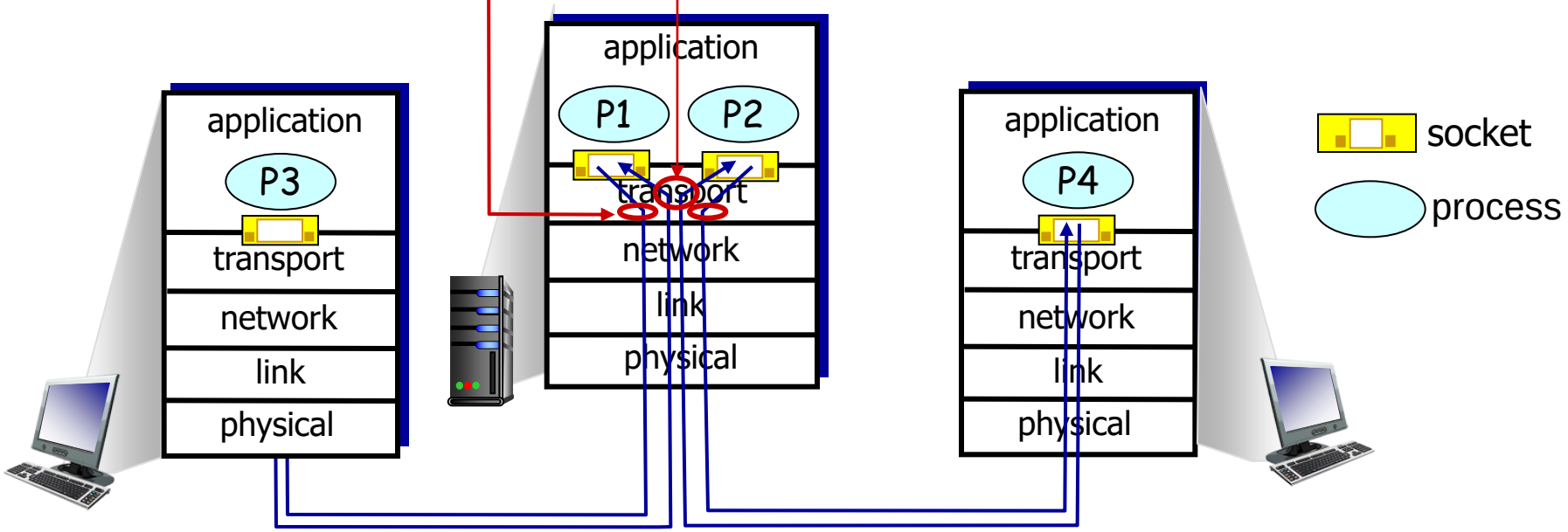
Πολύπλεξη/Αποπολύπλεξη

Πολύπλεξη στον αποστολέα:

Διαχείριση δεδομένων από πολλές sockets, προσθήκη κεφαλίδας μεταφοράς (χρησιμοποιείται για την αποπολύπλεξη)

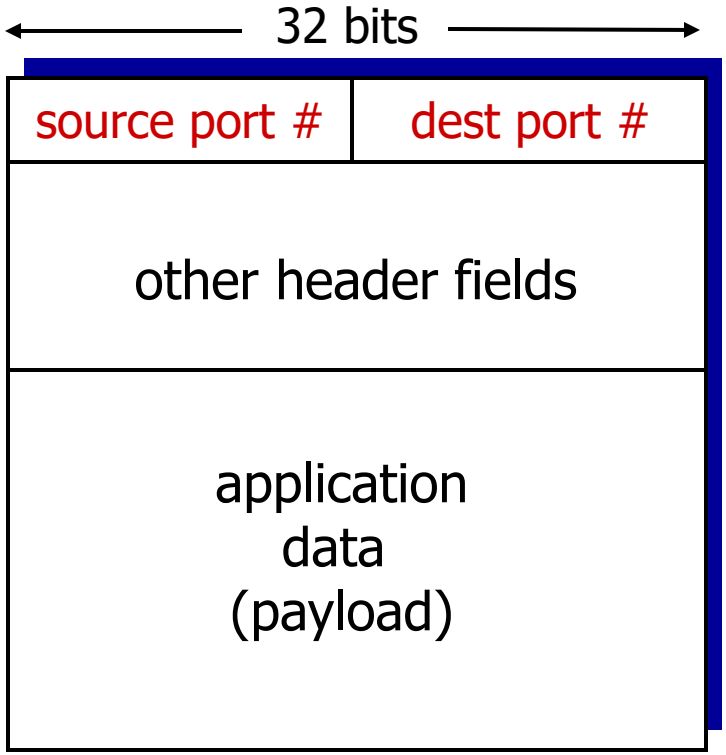
Αποπολύπλεξη στον παραλήπτη:

Χρήση της πληροφορίας της κεφαλίδας (header) για την παράδοση του segment στην σωστή socket.



Πως λειτουργεί η αποπολύπλεξη

- Ο host λαμβάνει IP datagrams (πακέτα δικτύου)
  - Κάθε datagram έχει την διεύθυνση IP του αποστολέα και την διεύθυνση IP του προορισμού.
  - κάθε datagram μεταφέρει ένα segment του επιπέδου μεταφοράς
  - Κάθε segment εχει τον αριθμό θύρας (port) του αποστολέα και του προορισμού.
- Ο host χρησιμοποιεί *IP addresses & port numbers* για να κατευθύνει το segment στην κατάλληλη socket



TCP/UDP segment format

Μη συνδεσμοστραφής αποπολύπλεξη (UDP)

- *υπενθύμιση:* η socket που δημιουργείται έχει τον τοπικό αριθμό θύρας (local port #):

```
DatagramSocket mySocket1
= new DatagramSocket(12534);
```

- *υπενθύμιση:* όταν δημιουργείται ένα datagram για να αποσταλεί από μια UDP socket, πρέπει να περιέχει
  - Διεύθυνση IP προορισμού
  - port # προορισμού

- Όταν ο host λαμβάνει ένα UDP segment:
  - Ελέγχει το port number προορισμού στο segment
  - Κατευθύνει το UDP segment στο socket με αυτό το port #.

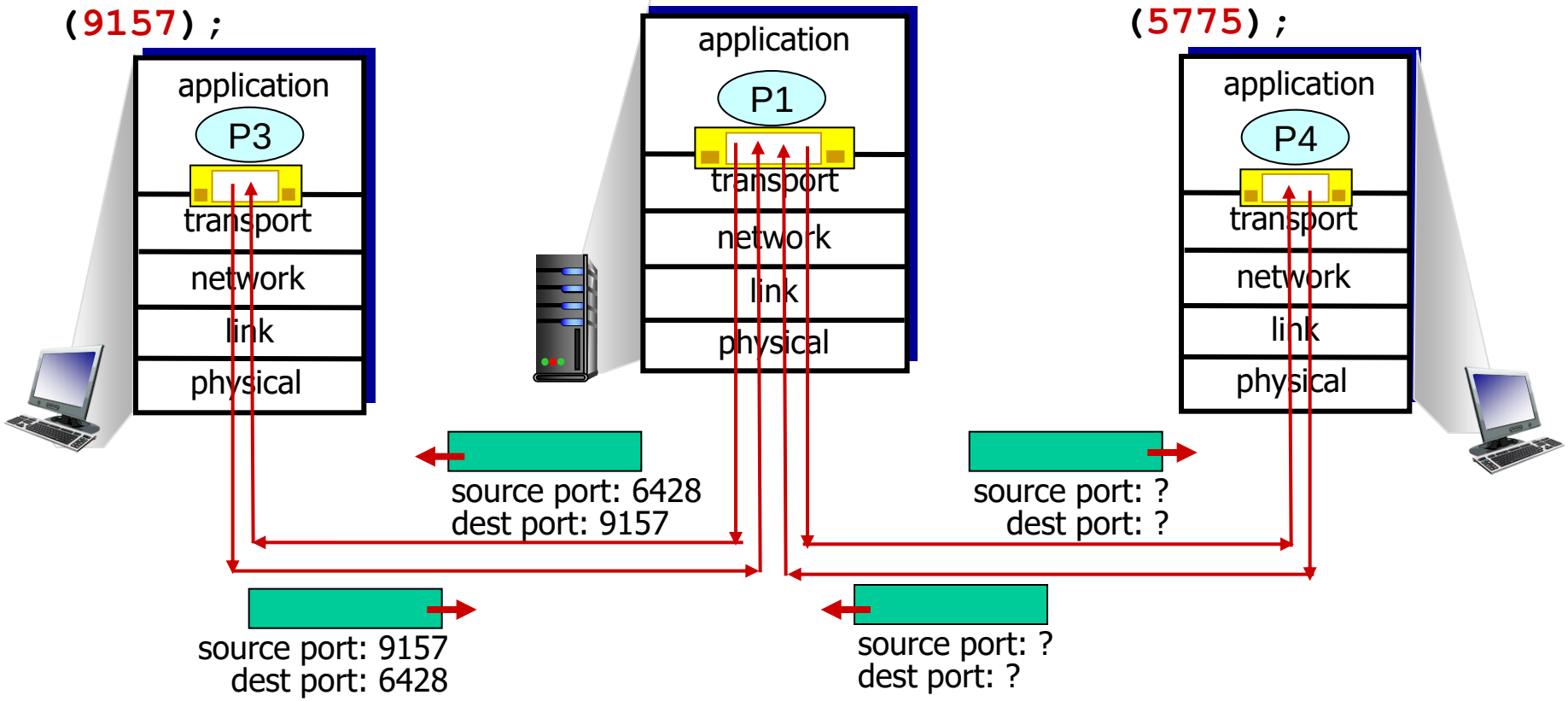
IP datagrams που έχουν το *ίδιο port # προορισμού,* αλλά διαφορετική διεύθυνση IP και/ή port number αποστολέα, θα κατευθυνθούν στην *ίδια socket* στον προορισμό.

Connectionless demux: παράδειγμα

```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

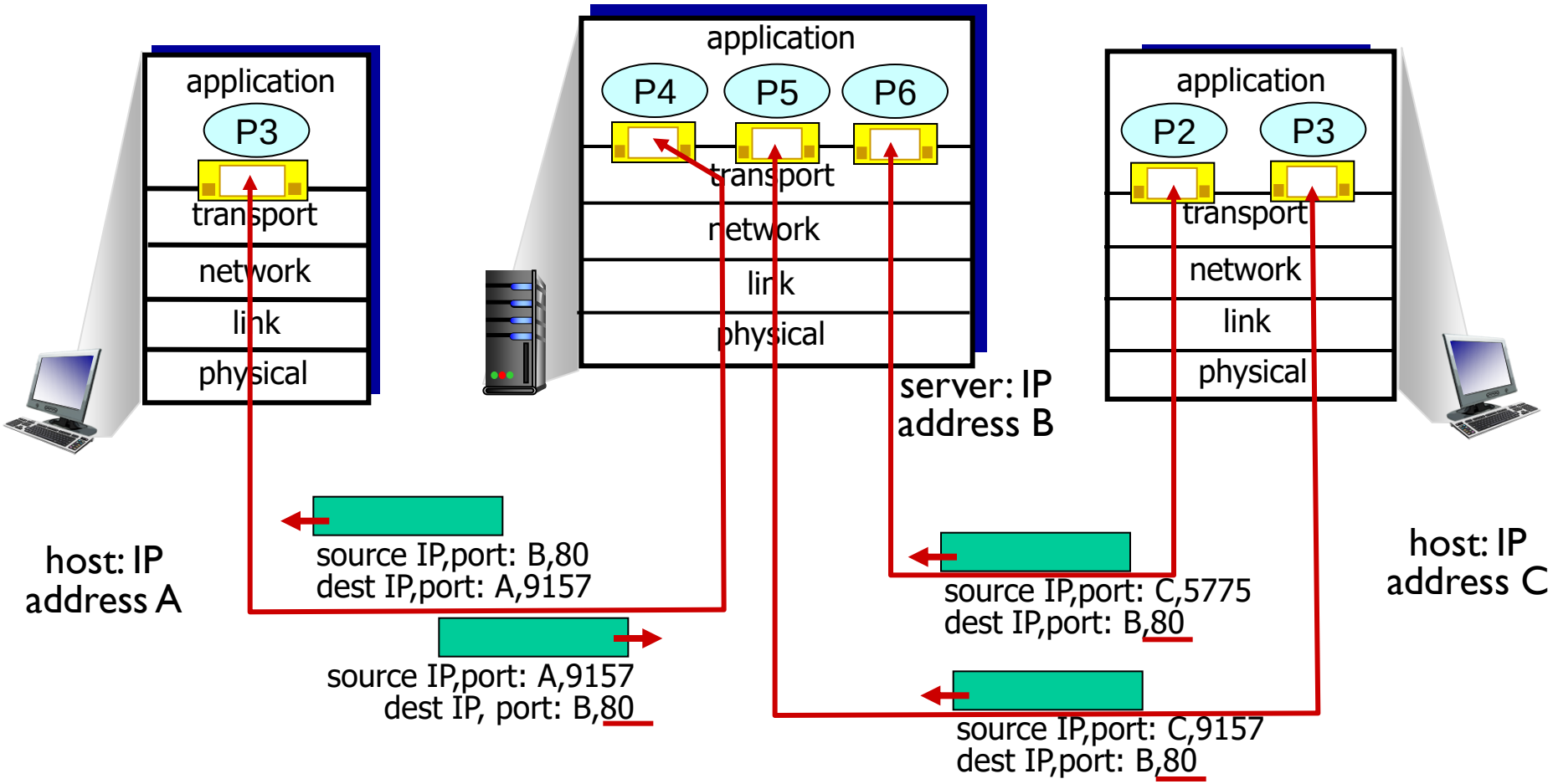
```
DatagramSocket
 mySocket1 = new
 DatagramSocket
 (5775);
```



Συνδεσμοστραφής αποπολύπλεξη (TCP)

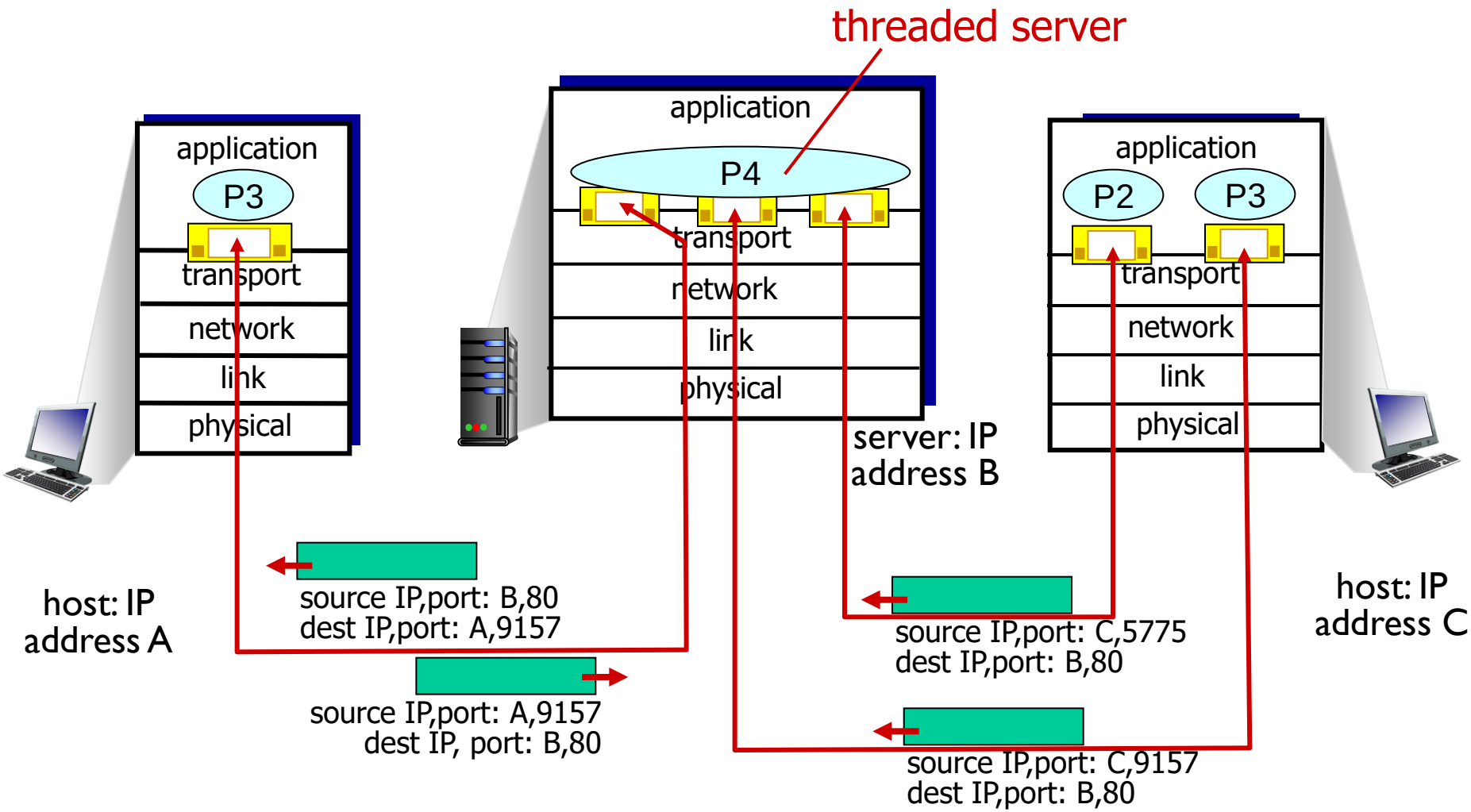
- Μια TCP socket προσδιορίζεται με 4 στοιχεία:
  - Διεύθυνση IP αποστολέα
  - Αριθμό port αποστολέα
  - Διεύθυνση IP προορισμού
  - Αριθμό port προορισμού
- demux: Ο παραλήπτης χρησιμοποιεί και τα τέσσερα στοιχεία για να κατευθύνει το segment στην κατάλληλη socket
- Ένας server host μπορεί να υποστηρίξει πολλές ταυτόχρονες TCP sockets:
  - Κάθε socket προσδιορίζεται με την δική τετράδα
- Οι web servers έχουν διαφορετικές sockets για κάθε client που συνδέεται
  - Σε μη-παραμένον HTTP έχουν διαφορετική socket για κάθε αίτημα (request).

Connection-oriented demux: παράδειγμα



Τρία segments, όλα προορίζονται για την διεύθυνση IP : B, και port προορισμού: 80. Κατευθύνονται σε *διαφορετικές* sockets

Connection-oriented demux: παράδειγμα



Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

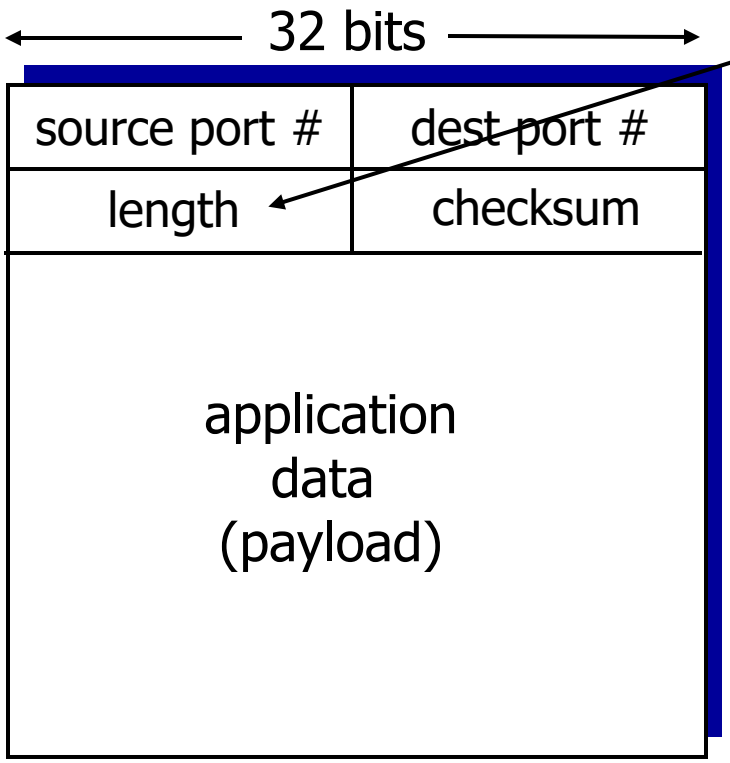
3.7 TCP έλεγχος συμφόρησης

UDP: User Datagram Protocol [RFC 768]

- Πολύ απλοϊκή και βασική υπηρεσία μεταφοράς. Τα UDP segments ενδέχεται:
  - Να χαθούν
  - Να παραδοθούν εκτός σειράς στην εφαρμογή
- *Μη-συνδεσμοστραφής (connectionless):*
  - Δεν υπάρχει ενεργή επικοινωνία (handshaking) μεταξύ UDP αποστολέα και παραλήπτη.
  - κάθε UDP segment αντιμετωπίζεται ανεξάρτητα από τα υπόλοιπα.

- Χρήσεις του UDP :
  - Εφαρμογές streaming multimedia (ανοχή σε απώλειες, ευαισθητες σε ρυθμό μετάδοσης)
  - DNS
  - SNMP
- Αξιόπιστη μεταφορά μέσω UDP:
  - Προσθήκη αξιοπιστίας στο επίπεδο εφαρμογών
  - Διάθρωση σφαλμάτων εξειδικευμένη για κάθε εφαρμογή!

UDP: κεφαλίδα ενός segment



Μορφή ενός UDP segment

Μήκος σε bytes του segment συμπεριλαμβανομένης της κεφαλίδας.

Γιατί υπάρχει το UDP?

- Δεν απαιτείται σύνδεση που μπορεί να προκαλεί καθυστέρηση.
- Είναι απλό.
- Μικρό μέγεθος κεφαλίδας
- Δεν υπάρχει έλεγχος συμφόρησης: Το UDP μπορεί να στέλνει δεδομένα όσο γρήγορα επιθυμούμε.

UDP checksum (Έλεγχος ακεραιότητας)

Στόχος: Ανίχνευση "λαθών" (π.χ., ανεστραμμένα bits) στο αποστελλόμενο segment

αποστολέας:

- Θεωρεί τα περιεχόμενα του segment, μαζί με τα πεδία της κεφαλίδας, ως σειρά 16-bit ακεραίων.
- checksum: πρόσθεση (one' s complement sum) των περιεχομένων του segment
- Ο αποστολέας τοποθετεί την τιμή του αθροίσματος (checksum value) στο αντίστοιχο πεδίο του segment.

παραλήπτης:

- Υπολογίζει το άθροισμα του λαμβανόμενου segment
- Συγκρίνει αν το υπολογισμένο checksum είναι ίσο με το checksum στο πεδίο του segment :
  - ΟΧΙ – ανιχνεύτηκε λάθος
  - ΝΑΙ – Δεν ανιχνεύτηκε λάθος.

    Μπορεί να υπάρχουν όμως λάθη…

Internet checksum: παράδειγμα

παράδειγμα: πρόσθεση δύο 16-bit ακεραίων

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Κρατούμενο	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Σημείωση: κατά την πρόσθεση αριθμών, ένα κρατούμενο από το πιο σημαντικό bit πρέπει να προστεθεί στο αποτέλεσμα

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

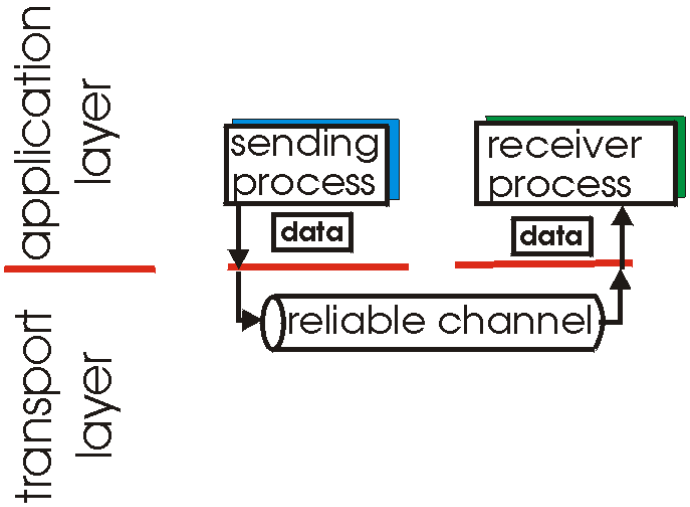
- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

Αρχές αξιόπιστης μεταφοράς δεδομένων (reliable data transfer- rdt)

- Είναι σημαντικό για τα επίπεδα εφαρμογών, μεταφοράς και δικτύου.
  - top-10 λίστα σημαντικών θεμάτων δικτύωσης

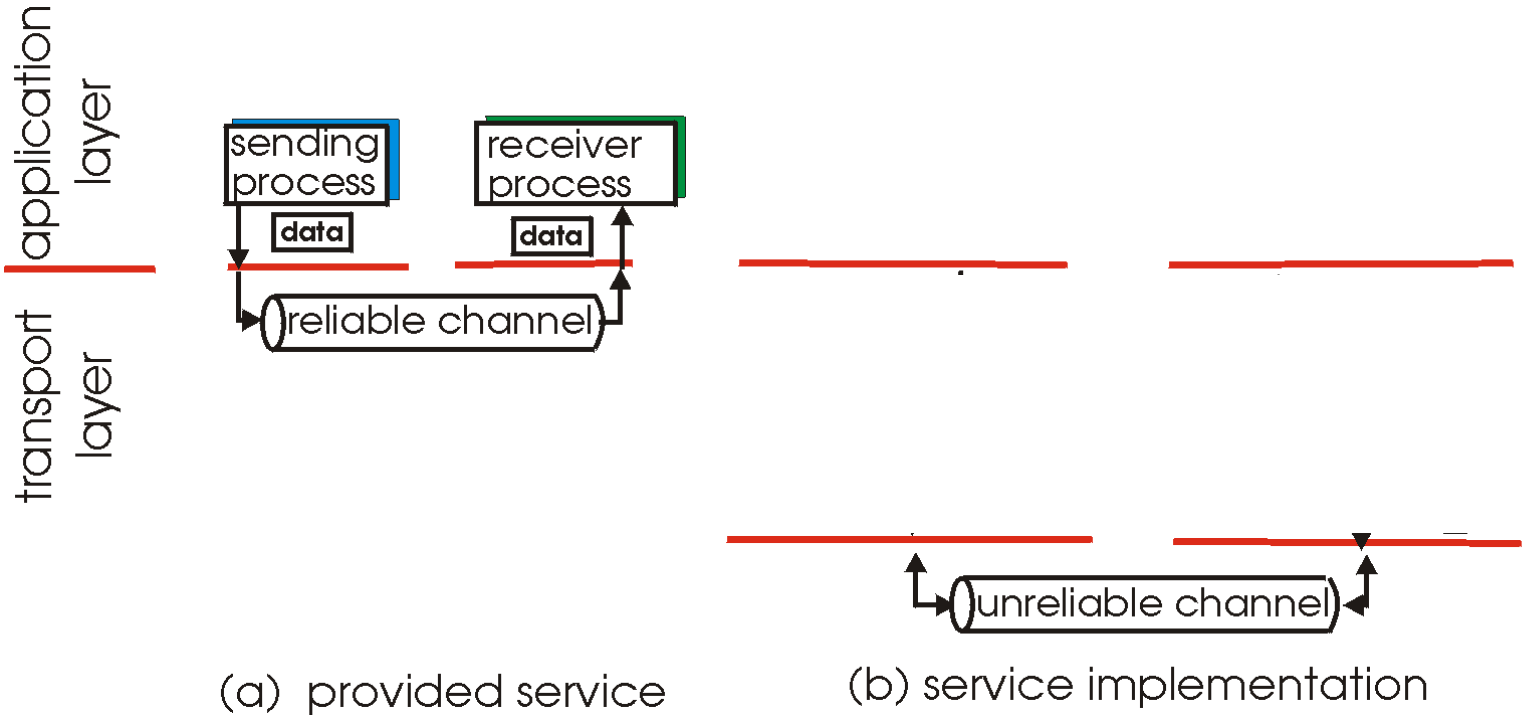


(a) provided service

- Τα χαρακτηριστικά του μη αξιόπιστου καναλιού καθορίζουν την πολυπλοκότητα του πρωτοκόλλου (rdt)

Αρχές αξιόπιστης μεταφοράς δεδομένων (reliable data transfer- rdt)

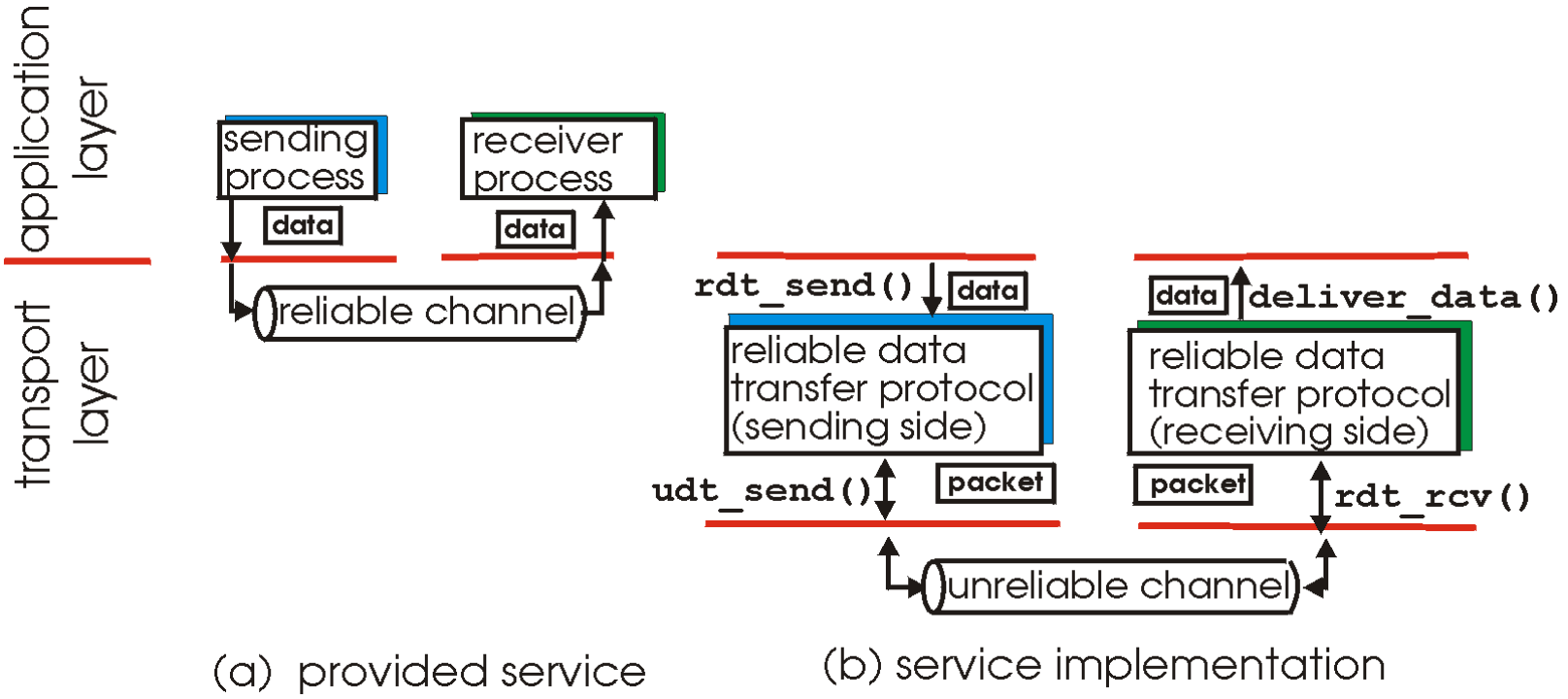
- Είναι σημαντικό για τα επίπεδα εφαρμογών, μεταφοράς και δικτύου.
  - top-10 λίστα σημαντικών θεμάτων δικτύωσης



- Τα χαρακτηριστικά του μη αξιόπιστου καναλιού καθορίζουν την πολυπλοκότητα του πρωτοκόλλου (rdt)

Αρχές αξιόπιστης μεταφοράς δεδομένων (reliable data transfer- rdt)

- Είναι σημαντικό για τα επίπεδα εφαρμογών, μεταφοράς και δικτύου.
  - top-10 λίστα σημαντικών θεμάτων δικτύωσης

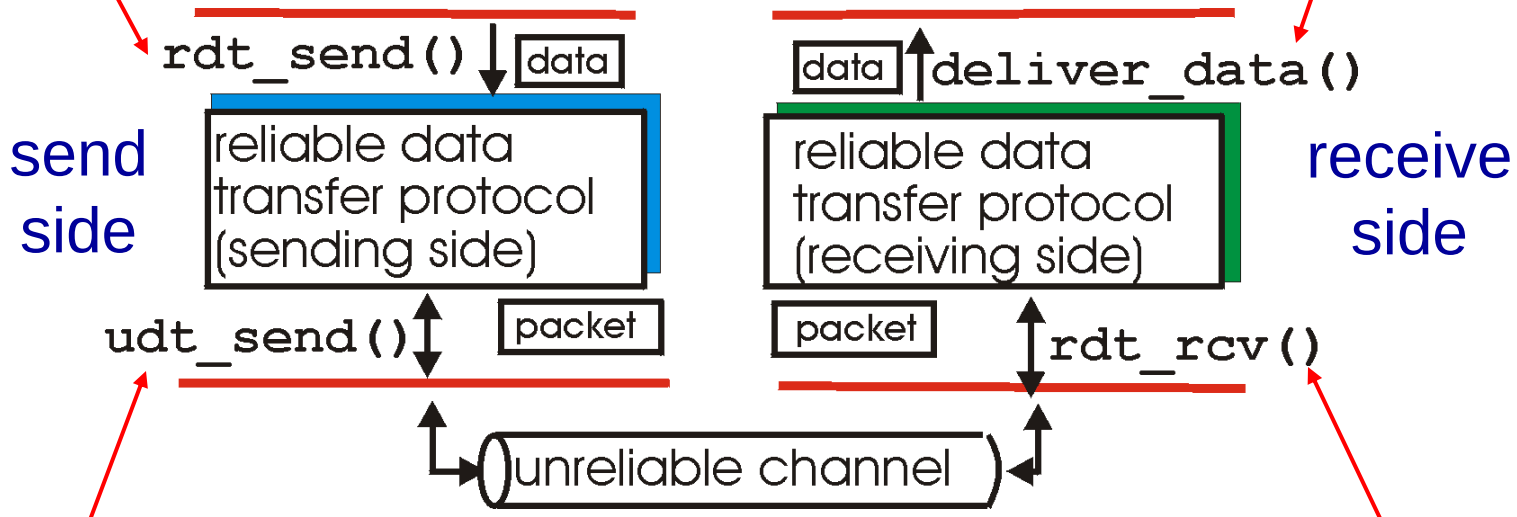


- Τα χαρακτηριστικά του μη αξιόπιστου καναλιού καθορίζουν την πολυπλοκότητα του πρωτοκόλλου (rdt)

Αξιόπιστη μεταφορά δεδομένων (rdt)

rdt_send() : καλείται από την εφαρμογή και της δίνονται τα δεδομένα για παράδοση στο ανώτερο επίπεδο του παραλήπτη

deliver_data() : καλείται από την rdt για να παραδώσει τα δεδομένα στο ανώτερο επίπεδο.



udt_send(): καλείται από την rdt, για να μεταφέρει το πακέτο στον παραλήπτη μέσω μη-αξιόπιστου διαύλου

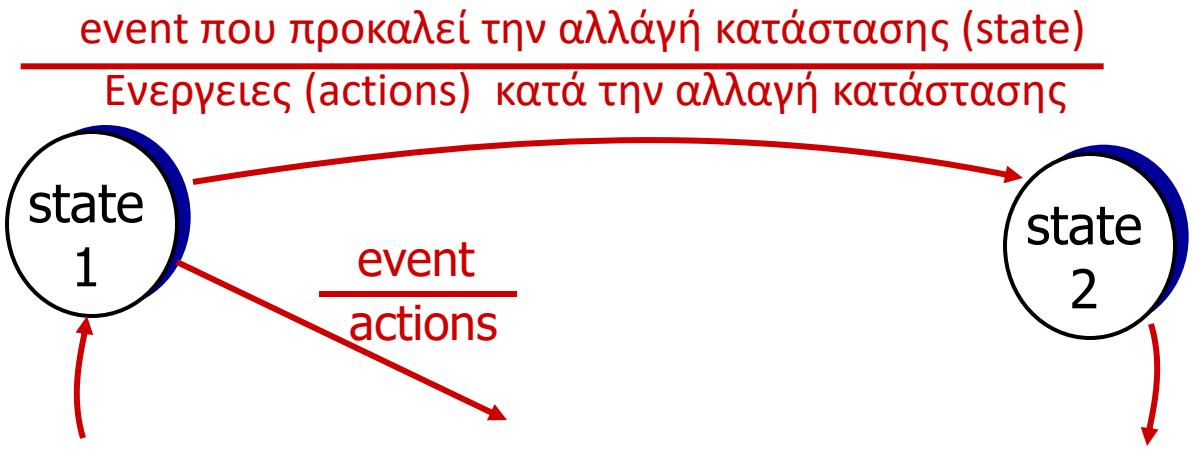
rdt_rcv() : καλείται όταν το πακέτο φτάσει στην πλευρά του παραλήπτη.

Αξιόπιστη μεταφορά δεδομένων (rdt)

Θα:

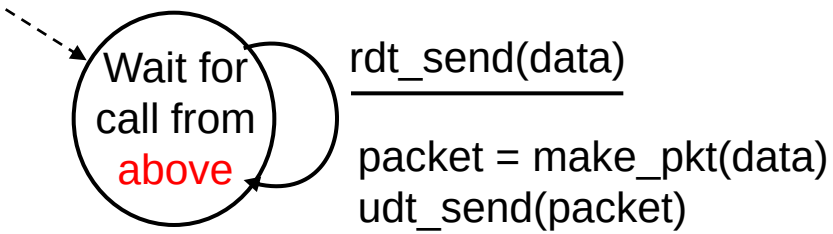
- Αναπτύξουμε σταδιακά ένα πρωτόκολλο αξιόπιστης μεταφοράς δεδομένων (rdt) στην πλευρά του αποστολέα και του παραλήπτη,
- Θα θεωρήσουμε μεταφορά δεδομένων σε μια κατεύθυνση
  - Αλλά πληροφορίες ελέγχου θα ρέουν και στις δύο κατευθύνσεις
- Θα χρησιμοποιήσουμε Αυτόματα Πεπερασμένων Καταστάσεων (Finite State Machines) για να περιγράψουμε τον sender και τον , receiver

state: η επόμενη κατάσταση καθορίζεται μοναδικά από το επόμενο γεγονός (event)

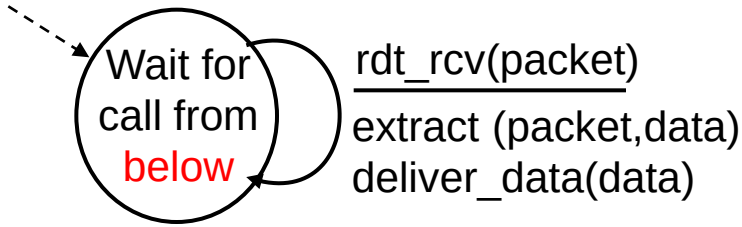


rdt1.0: reliable transfer over a reliable channel

- Ο δίαυλος μεταφοράς είναι απολύτως αξιόπιστος
  - Καθόλου λάθη σε bit
  - Καμία απώλεια πακέτων
- Χωριστά FSMs για αποστολέα και παραλήπτη:
  - Ο sender στέλνει δεδομένα στον υποκείμενο δίαυλο.
  - Ο receiver διαβάζει δεδομένα από υποκείμενο δίαυλο.



sender



receiver

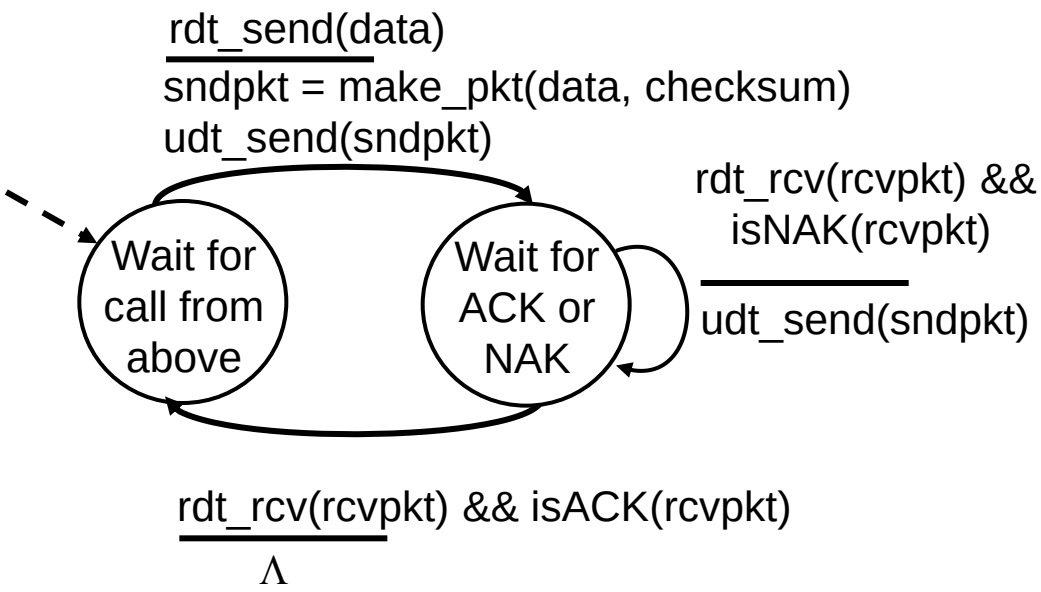
rdt2.0: Διαυλος με λάθη σε bit

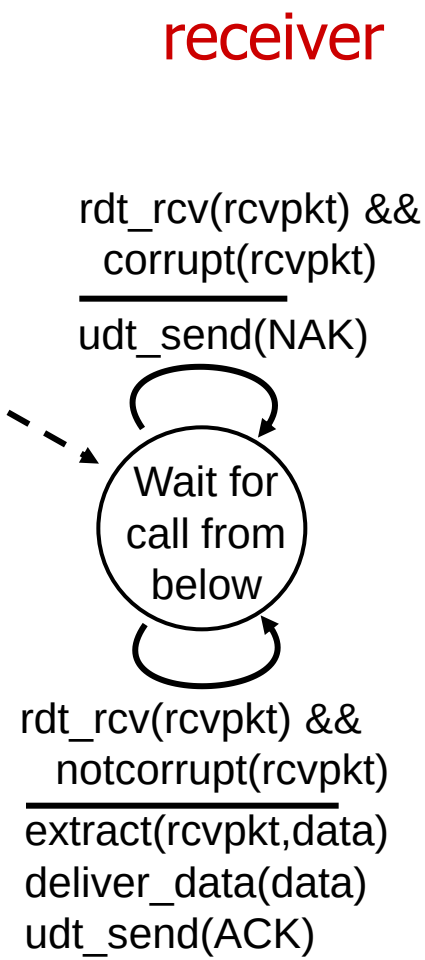
- Ο δίαυλος μπορεί να αλλάζει τις τιμές των bit σε ένα πακέτο
  - Χρήση του checksum για ανίχνευση λαθών σε bit.
- *Ερώτημα* : Πως διορθώνουμε τα λάθη;

rdt2.0: Διαυλος με λάθη σε bit

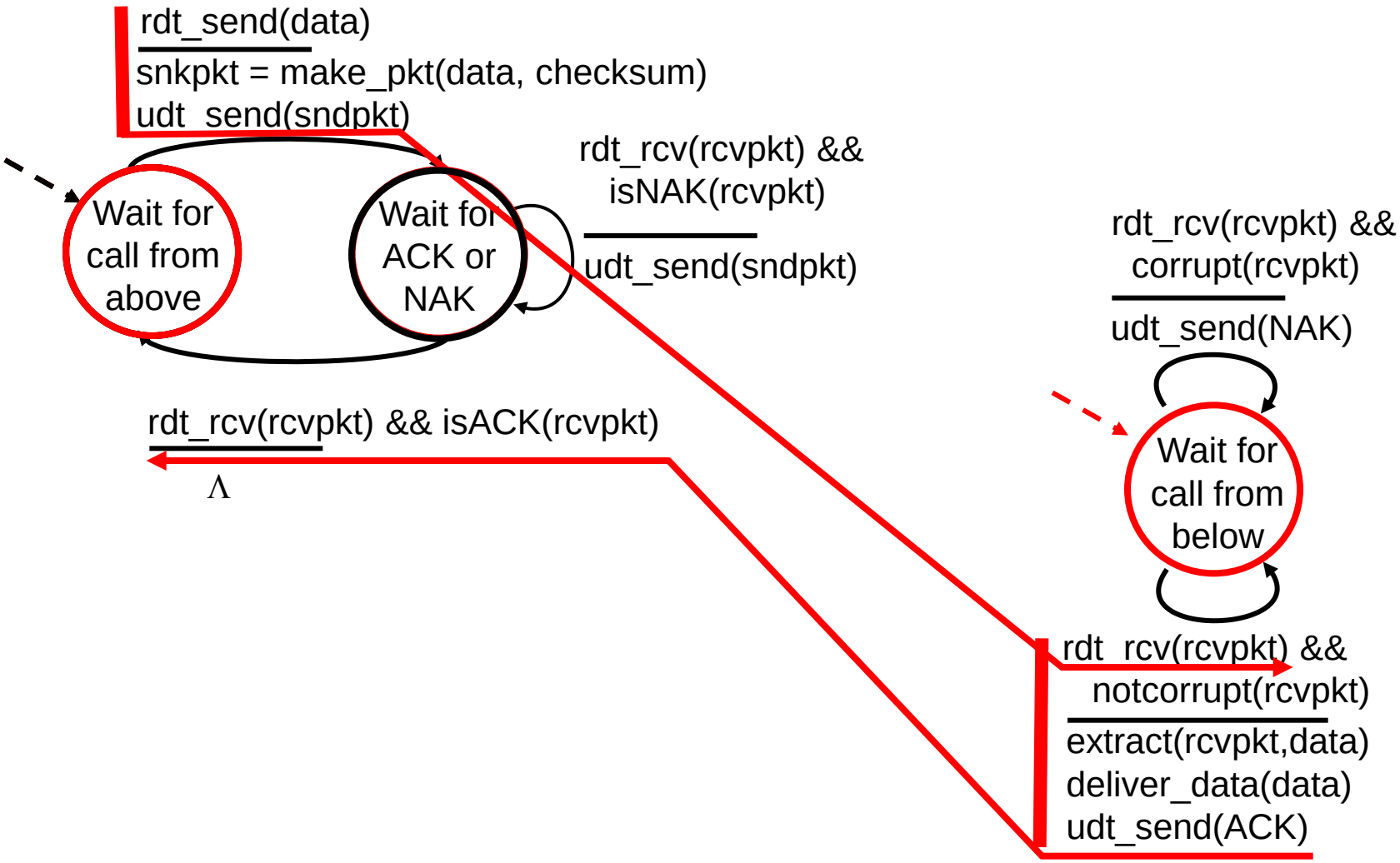
- Ο δίαυλος μπορεί να αλλάζει τις τιμές των bit σε ένα πακέτο
  - Χρήση του checksum για ανίχνευση λαθών σε bit.
- *Ερώτημα* : Πως διορθώνουμε τα λάθη;
  - *acknowledgements (ACKs):* ο παραλήπτης ενημερώνει τον αποστολέα ότι το πακέτο παραλήφθηκε σωστά.
  - *negative acknowledgements (NAKs):* ο παραλήπτης ενημερώνει τον αποστολέα ότι το πακέτο έχει λάθη.
  - Ο αποστολέας ξαναστέλνει το πακέτο αν πάρει ένα NAK
- Νέοι μηχανισμοί στο **rdt2.0** (περά από το **rdt1.0**):
  - Ανίχνευση Λαθών (error detection)
  - Ανατροφοδότηση (feedback): μηνύματα έλεγχου (ACK,NAK) από τον receiver στον sender

rdt2.0: Περιγραφή FSM

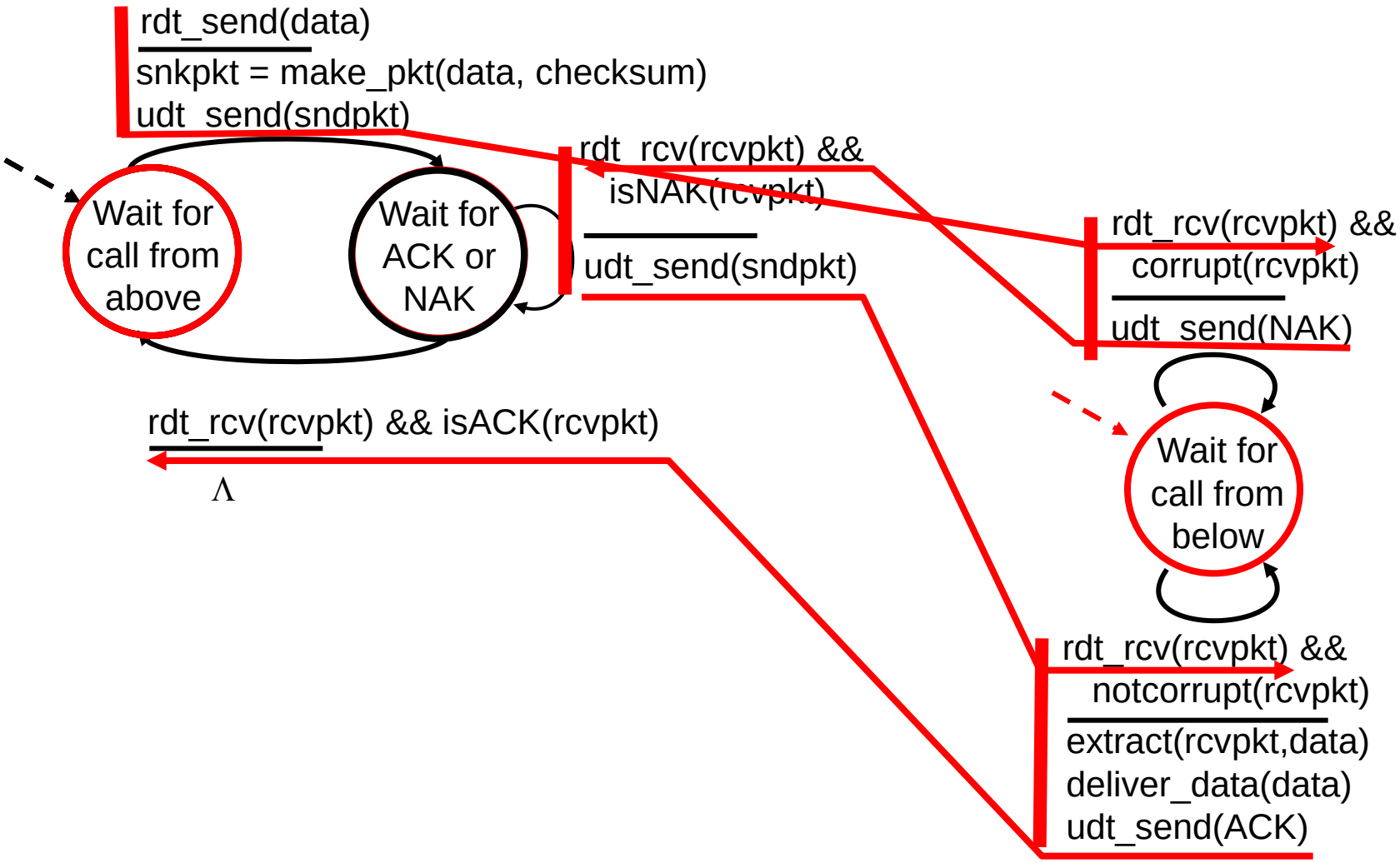




rdt2.0: Λειτουργία χωρίς λάθη



rdt2.0: Λειτουργία σε περίπτωση λάθους



rdt2.0 έχει ένα σημαντικό πρόβλημα.

τι θα συμβεί αν ACK / NAK καταστραφεί;

- Ο αποστολέας δεν ξέρει τι συνέβη στο δέκτη!
- δεν μπορεί απλά να ξαναστείλει: πιθανό διπλότυπο πακέτο.

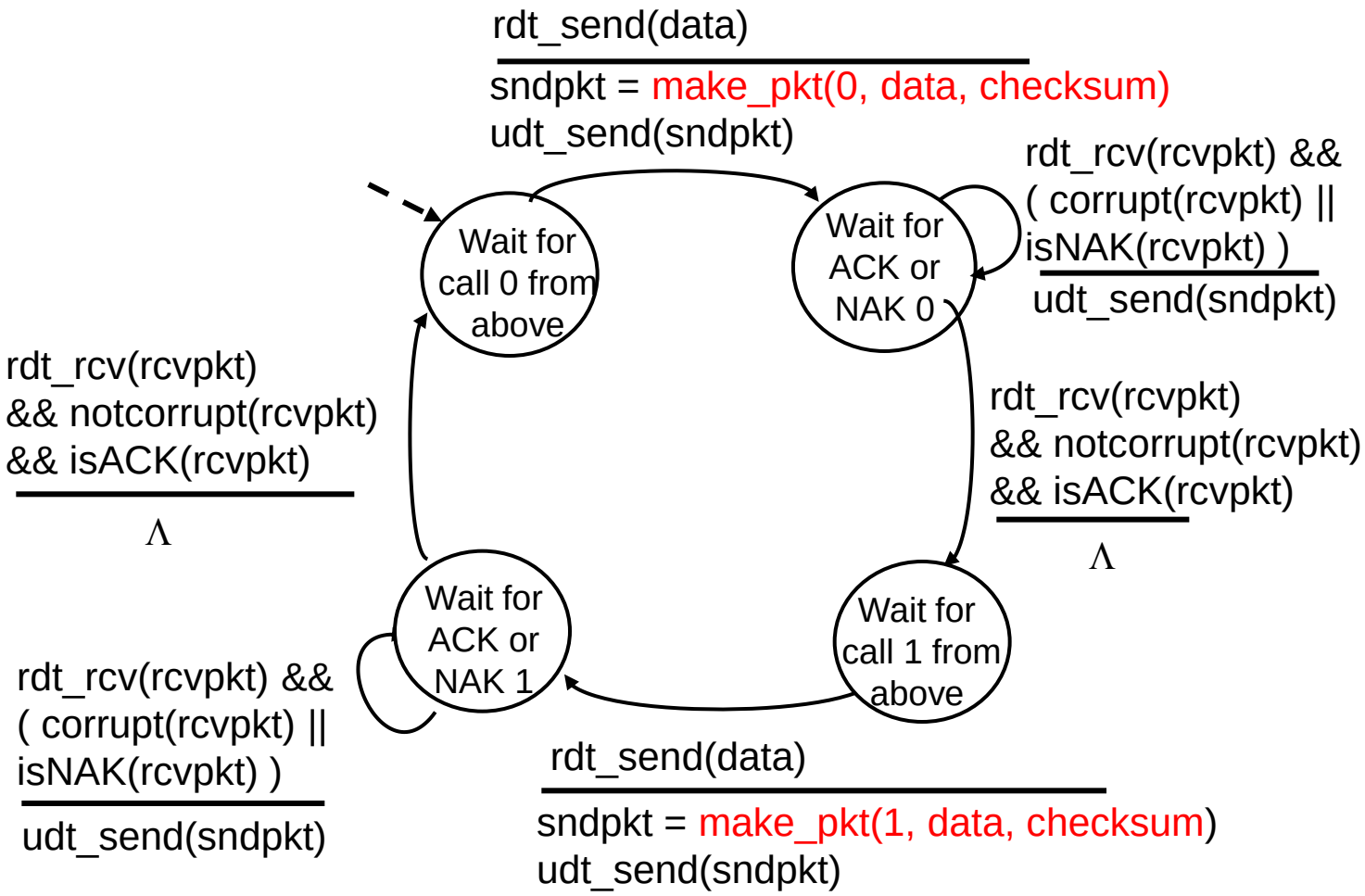
Διαχείριση διπλοτύπων:

- Ο sender ξαναστέλνει το τρέχον πακέτο αν ACK/NAK είναι κατεστραμμένο.
- Ο sender προσθέτει ένα αριθμό σειράς (*sequence number)* σ ε κάθε πακέτο.
- Ο receiver αγνοεί (δεν τα στέλνει στο ανώτερο επίπεδο) διπλά πακέτα.

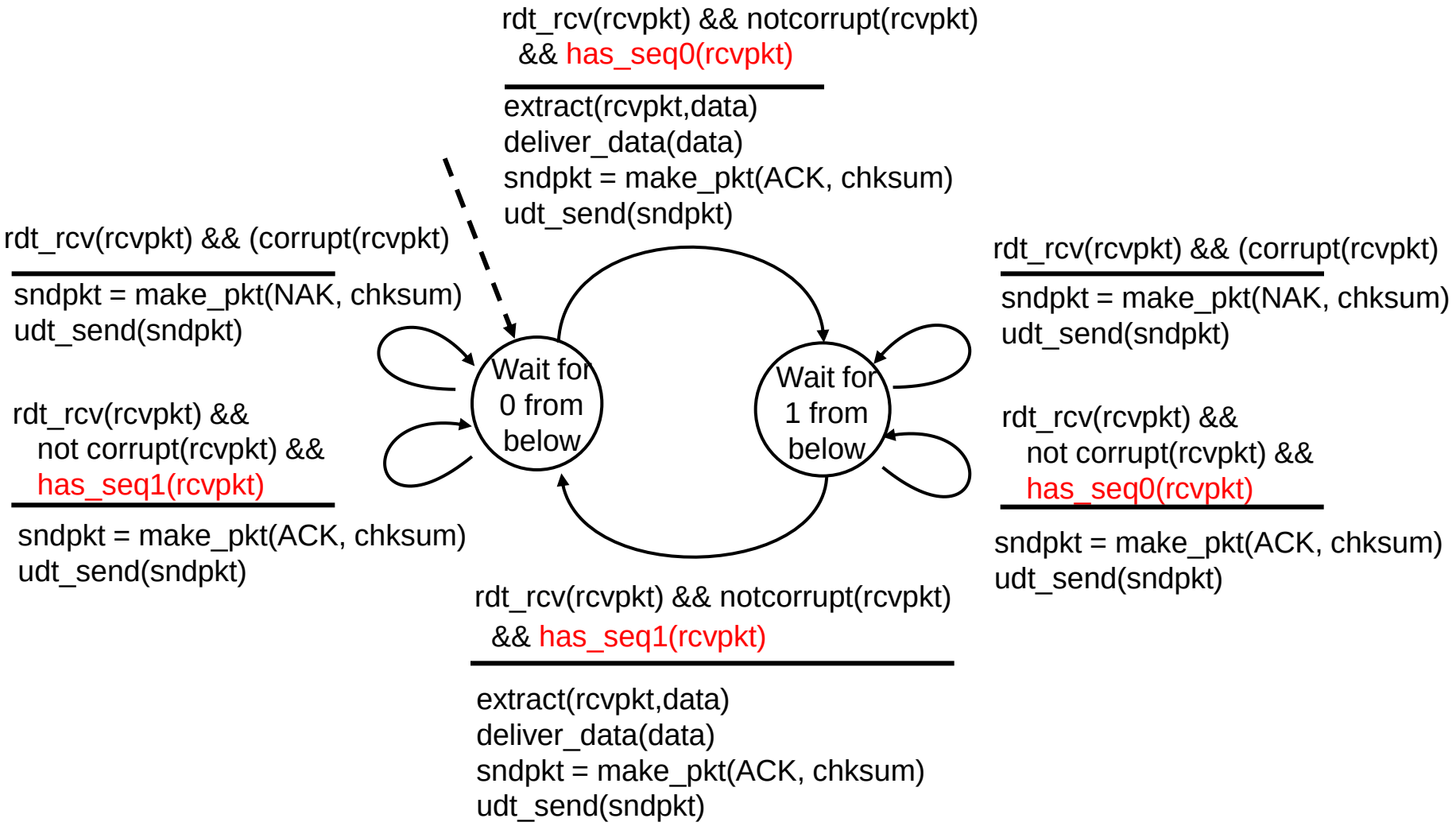
stop and wait

Ο αποστολέας στέλνει ένα πακέτο και περιμένει απάντηση από παραλήπτη.

rdt2.1: O sender, διαχειρίζεται κατεστραμμένα ACK/NAKs



rdt2.1: O receiver διαχειρίζεται κατεστραμμένα ACK/NAKs



rdt2.1: ανακεφαλαίωση

sender:

- Προσθήκη seq # στο πακέτο
- Δύο seq. #’ s (0,1) αρκούν. Γιατί?
- Πρέπει να εξετάζει αν τα ACK/NAK που λαμβάνει είναι κατεστραμμένα.
- Διπλάσιες καταστάσεις (states)
  - Η κατάσταση πρέπει να θυμάται αν το αναμενόμενο πακέτο πρέπει να έχει seq # 0 ή 1

receiver:

- Πρέπει να ελέγχει για διπλά πακέτα
  - Η κατάσταση (state) δείχνει αν αναμένεται πακέτο με seq # 0 ή 1.
- Σημείωση: Ο receiver δεν μπορεί να ξέρει αν το τελευταίο ACK/NAK έφτασε σωστά στον sender.

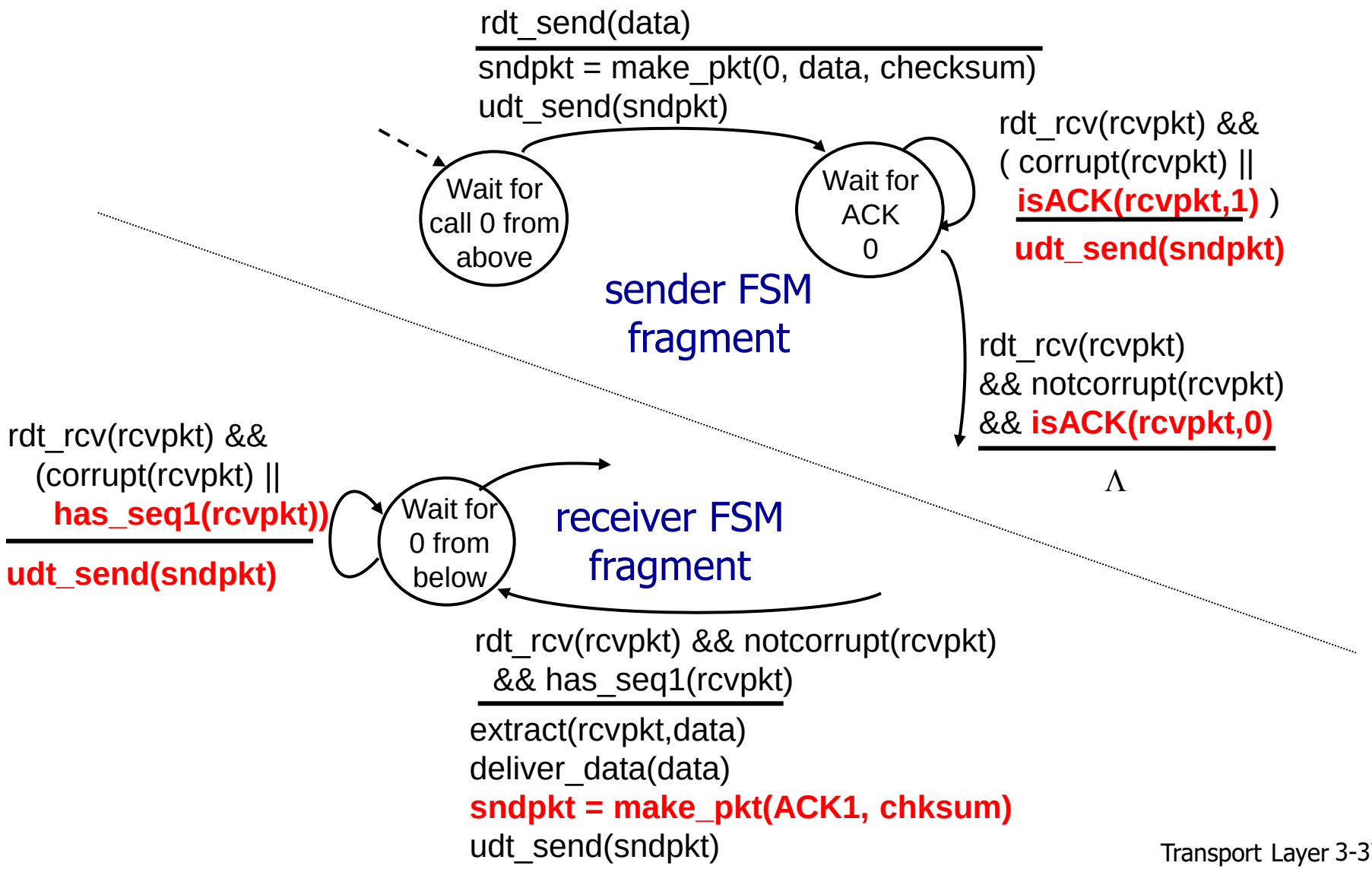
rdt2.2: Ένα πρωτόκολλο χωρίς NAK

- Ίδια λειτουργικότητα με rdt2.1, χρησιμοποιώντας μόνο ACKs

- Αντί για NAK, ο receiver στέλνει ACK για το τελευταίο πακέτο που παραλήφθηκε σωστά.
  - Ο receiver πρέπει να συμπεριλάβει και το seq # του πακέτου για το οποίο στέλνει ACK.

- Διπλό ACK στον sender έχει ως αποτέλεσμα την ίδια ενέργεια με το NAK: *αποστολή ξανά του τρέχοντος πακέτου.*

rdt2.2: τμήματα των sender, receiver



rdt3.0: Δίαυλοι με λάθη και απώλειες

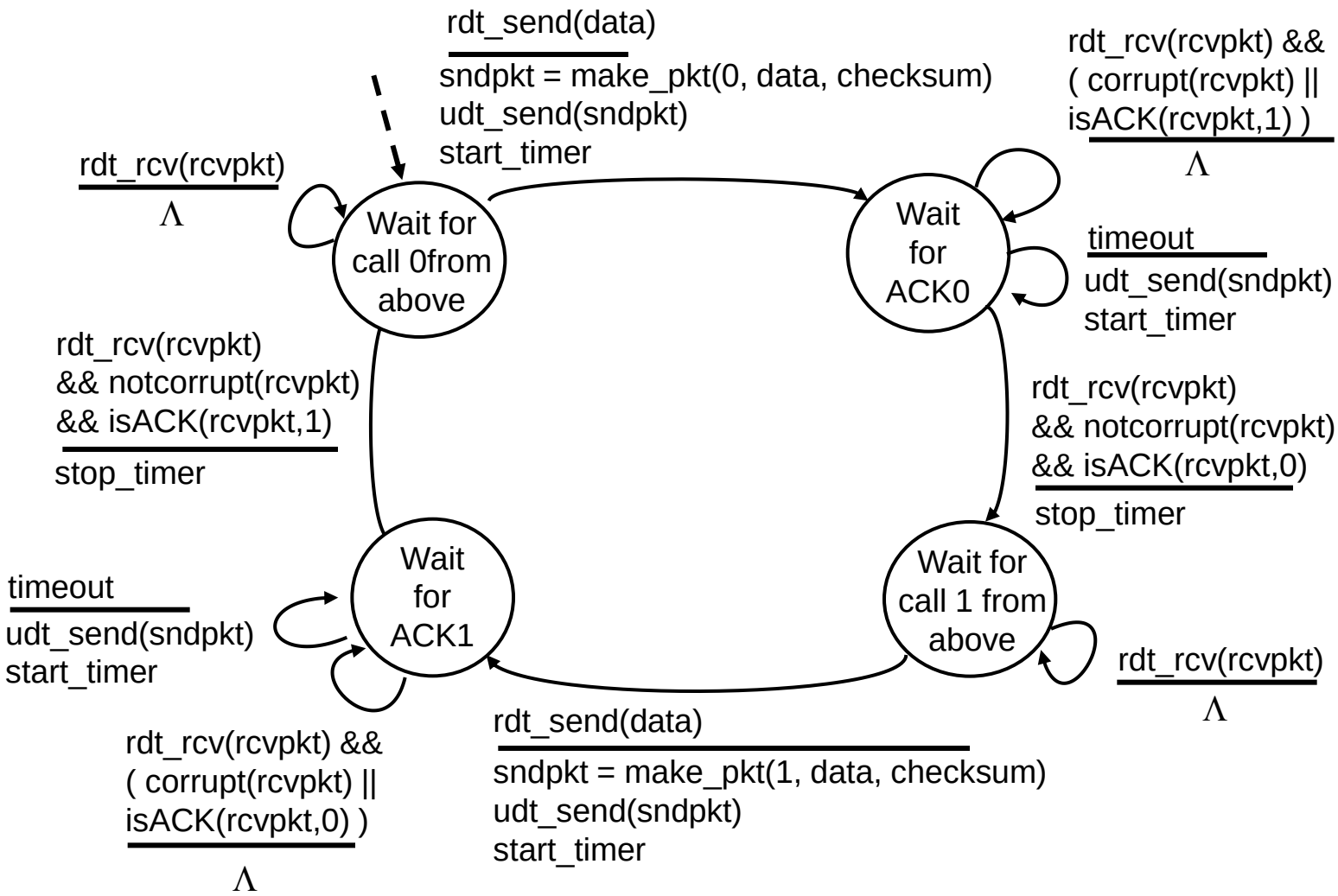
Νέα παραδοχή: ο υποκείμενος διαυλος μπορεί να χάσει πακέτα (data, ACKs)

- checksum, seq. #, ACKs, και επανάληψη μετάδοσης δεν είναι πλέον αρκετά.

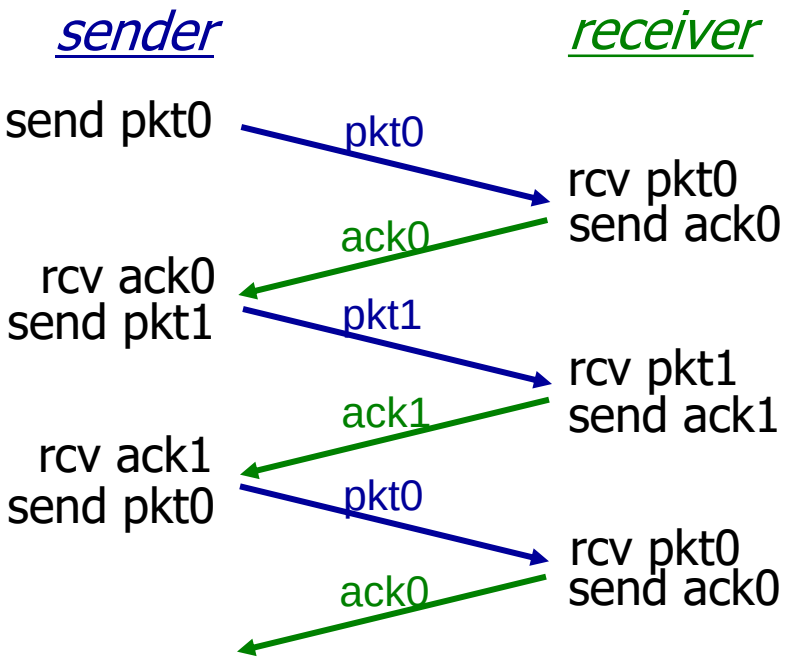
Προσέγγιση:

- Ο sender περιμένει ένα "λογικό" χρονικό διάστημα για ACK
- Ξαναστέλνει αν δεν πάρει ACK σε αυτό το χρονικό διάστημα.
- Αν το πακέτο (ή το ACK) απλά καθυστέρησαν (δεν χάθηκαν):
  - Η επανάληψη αποστολής θα στείλει διπλότυπο αλλά αυτό το επιλύει το seq. #'
  - Ο receiver πρέπει να προσδιορίσει το seq # του πακέτου στο ACK.
- Απαιτείται ένας μετρητής χρόνου.

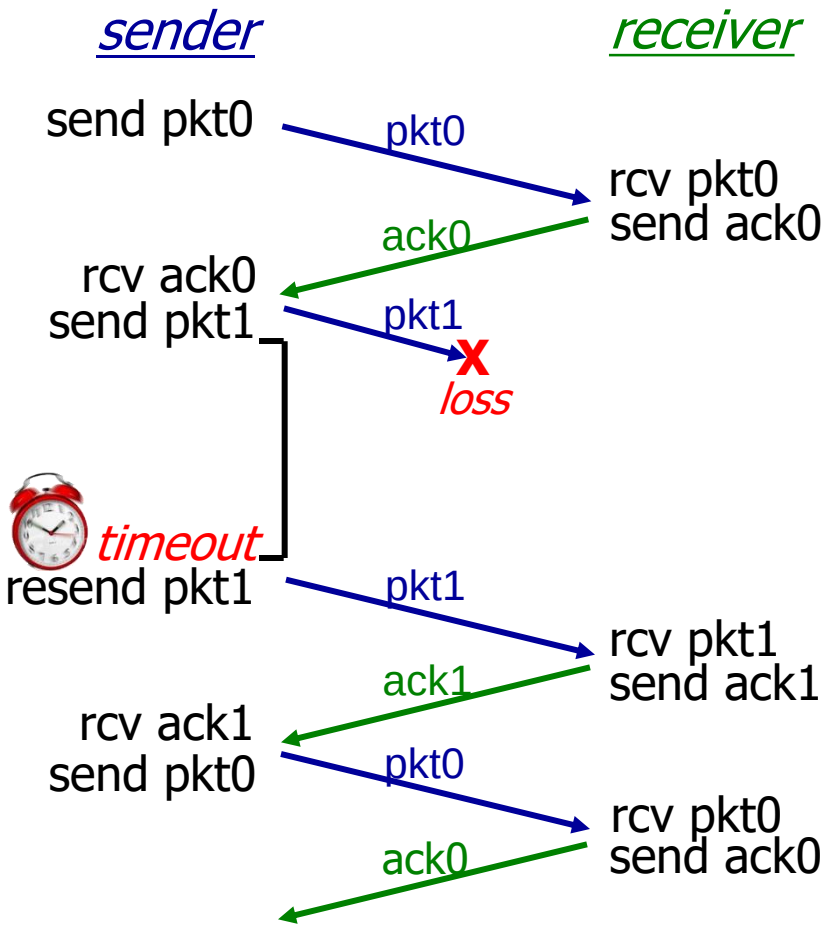
rdt3.0 sender



rdt3.0 σε λειτουργία

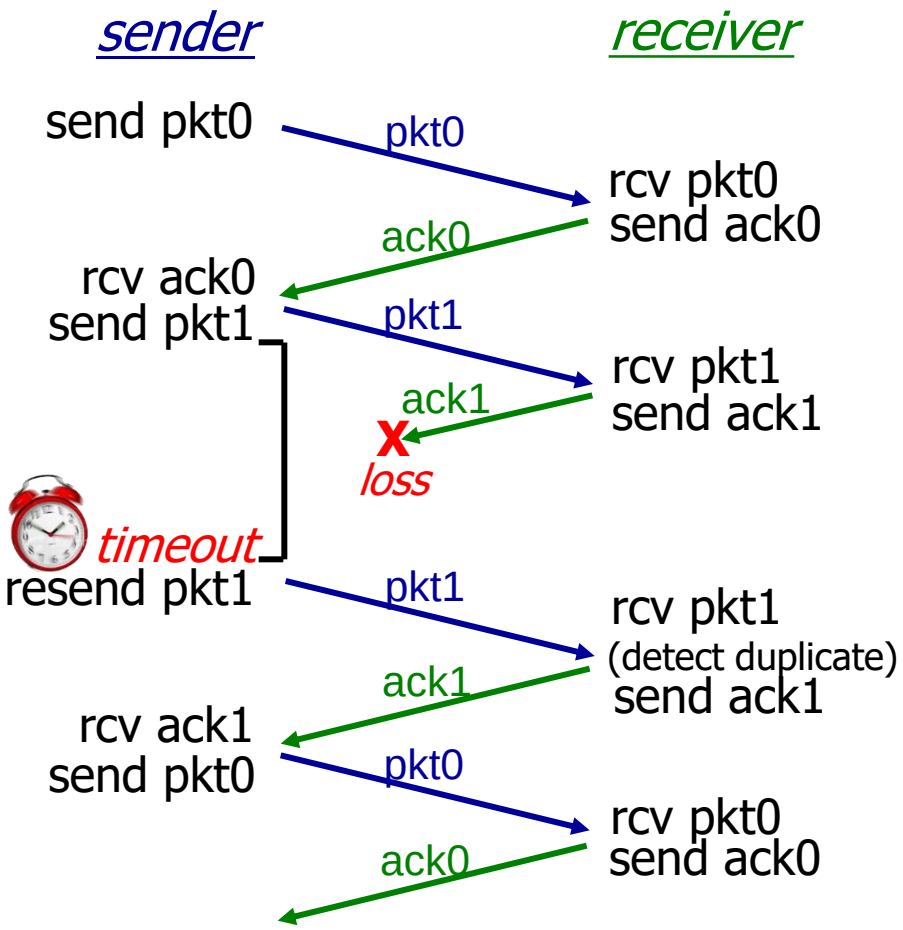


(a) no loss

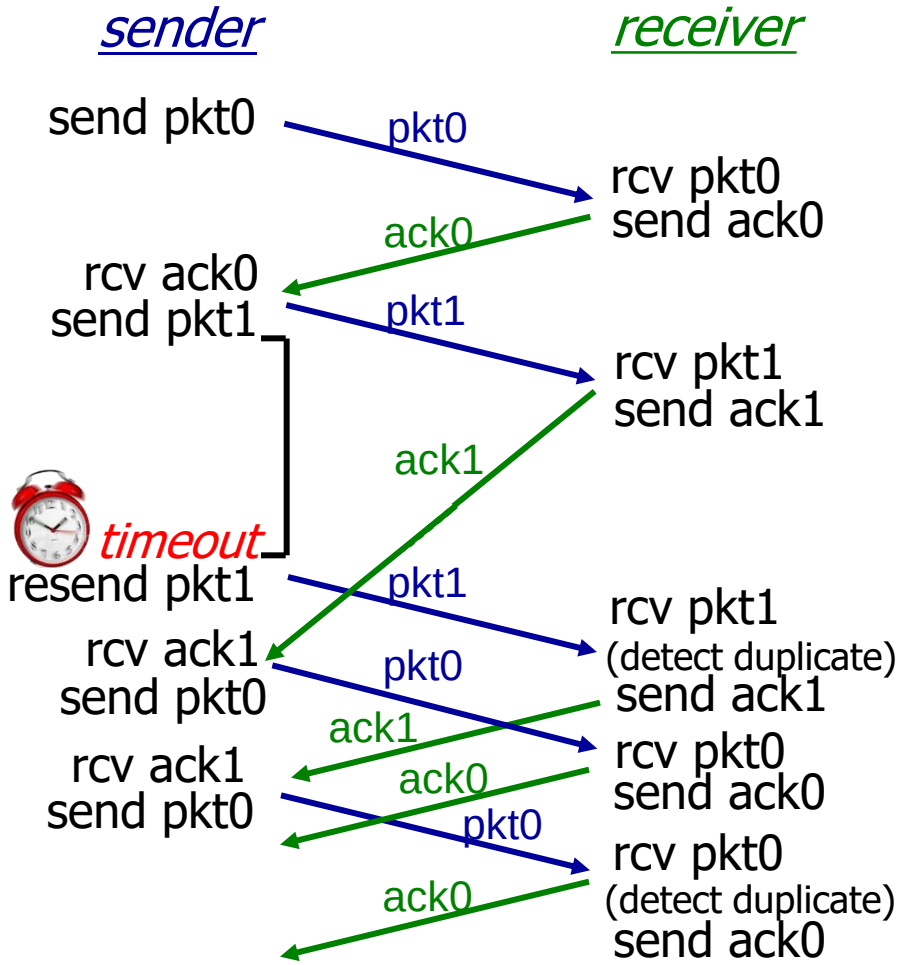


(b) packet loss

rdt3.0 σε λειτουργία

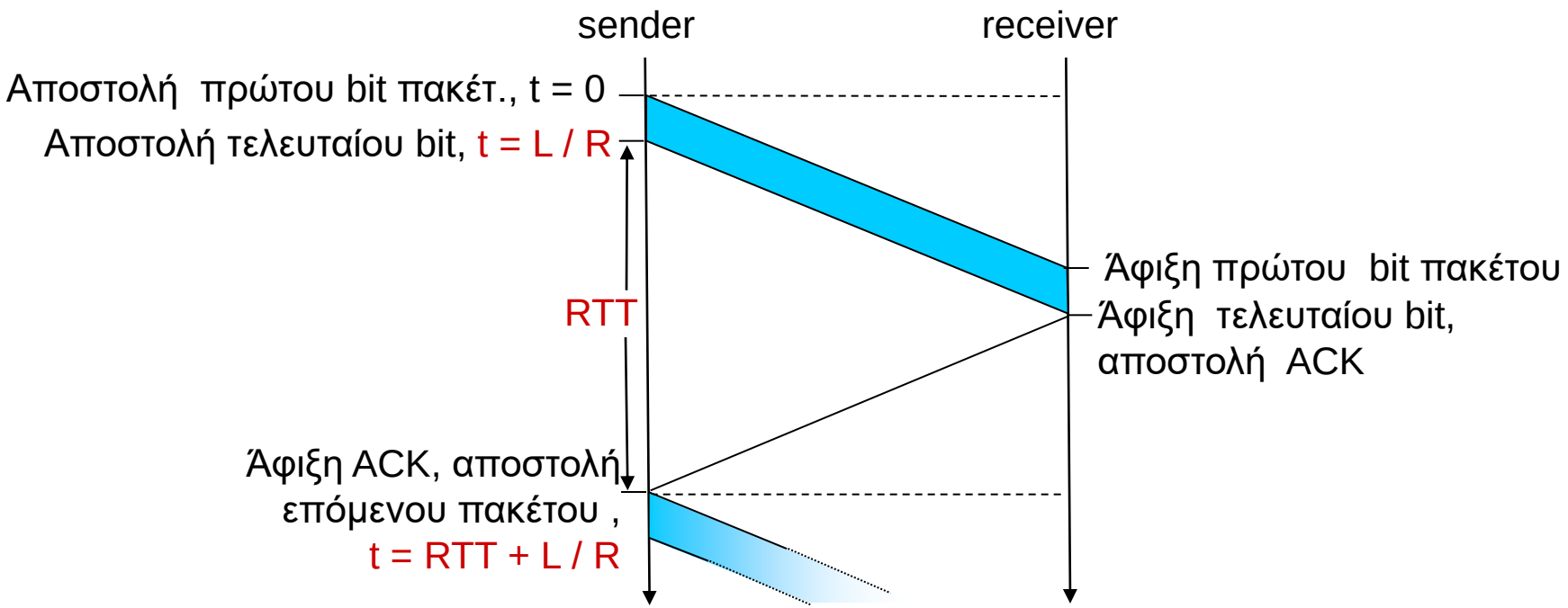


(c) ACK loss



(d) premature timeout/ delayed ACK

rdt3.0: λειτουργία stop-and-wait



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Επιδόσεις του rdt3.0

- Το rdt3.0 είναι σωστό, αλλά η απόδοση πολύ κακή.
- Π.χ..: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

  - U_{sender}: *utilization* – ποσοστό χρόνου που ο sender στέλνει δεδομένα

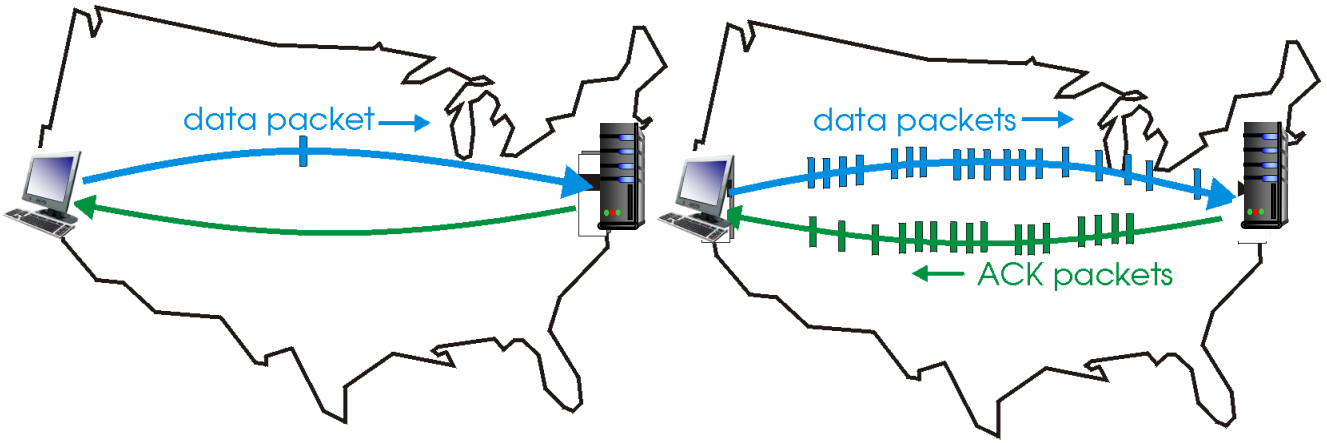
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

  - Αν RTT=30 msec, 1KB πακέτο κάθε 30 msec: 33kB/sec μετάδοση σε σύνδεση 1 Gbps.
  - Το πρωτόκολλο δικτύου περιορίζει την χρήση των φυσικών πόρων!

Πρωτόκολλα σωλήνωσης (pipelined)

pipelining: Ο αποστολέας επιτρέπει αποστολή πολλαπλών πακέτων σε εκκρεμότητα, τα οποία δεν έχουν ακόμη αναγνωριστεί από παραλήπτη.

- Το εύρος της αρίθμησης (sequence numbers) πρέπει να αυξηθεί.
- Απαιτείται buffer στον sender και/ή στον receiver

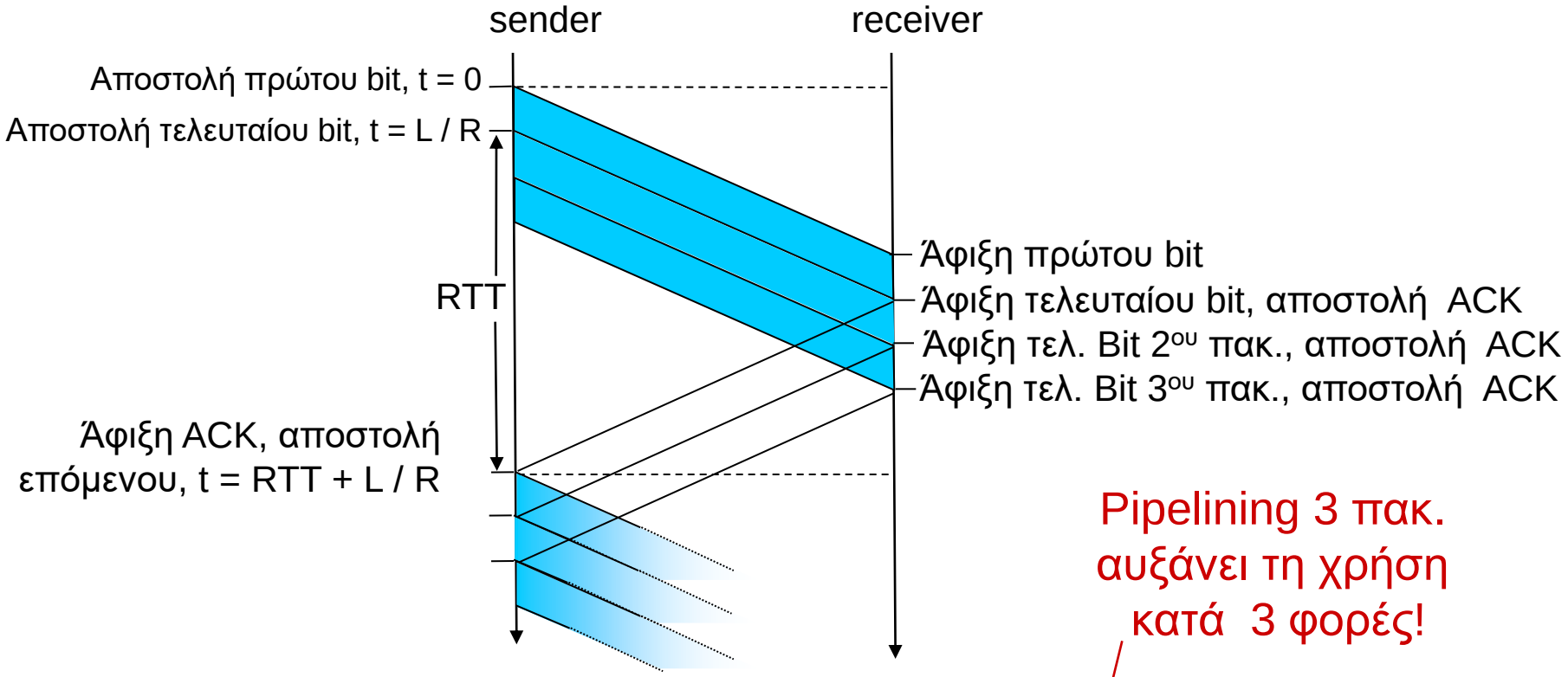


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Δύο γενικές μορφές pipelined πρωτοκόλλων:
- *go-Back-N (πηγαινε πίσω N)*
- *selective repeat (επιλεκτική επανάληψη)*

Pipelining: Αυξημένη χρησιμοποίηση



Pipelining 3 πακ. αυξάνει τη χρήση κατά 3 φορές!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Pipelined πρωτόκολλα: επισκόπηση

Go-back-N:

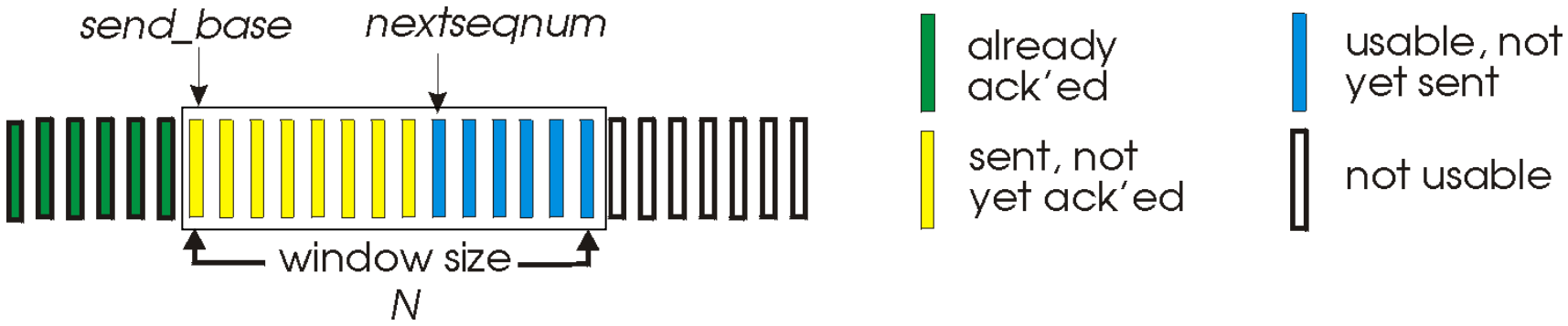
- Ο sender μπορεί να έχει μέχρι N μη αναγνωρισμένα πακέτα στην pipeline
- Ο receiver στέλνει μόνο αθροιστικά (*cumulative) ack*
  - Δεν στέλνει ack για ένα πακέτο αν υπάρχει κενό.
- Ο sender έχει timer για το αρχαιότερο πακέτο χωρίς ack
  - Όταν ο χρόνος εκπνεύσει, αποστέλλονται ξανά **όλα** τα μη αναγνωρισμένα πακέτα.

Selective Repeat:

- Ο sender μπορεί να έχει μέχρι N μη αναγνωρισμένα πακέτα στην pipeline
- Ο receiver στέλνει *ξεχωριστό ack* για κάθε πακέτο.
- Ο sender διατηρεί timer για κάθε μη αναγνωρισμένο πακέτο.
  - Όταν ο χρόνος εκπνεύσει, αποστέλλεται ξανά **μόνο** το πακέτο χωρίς ack.

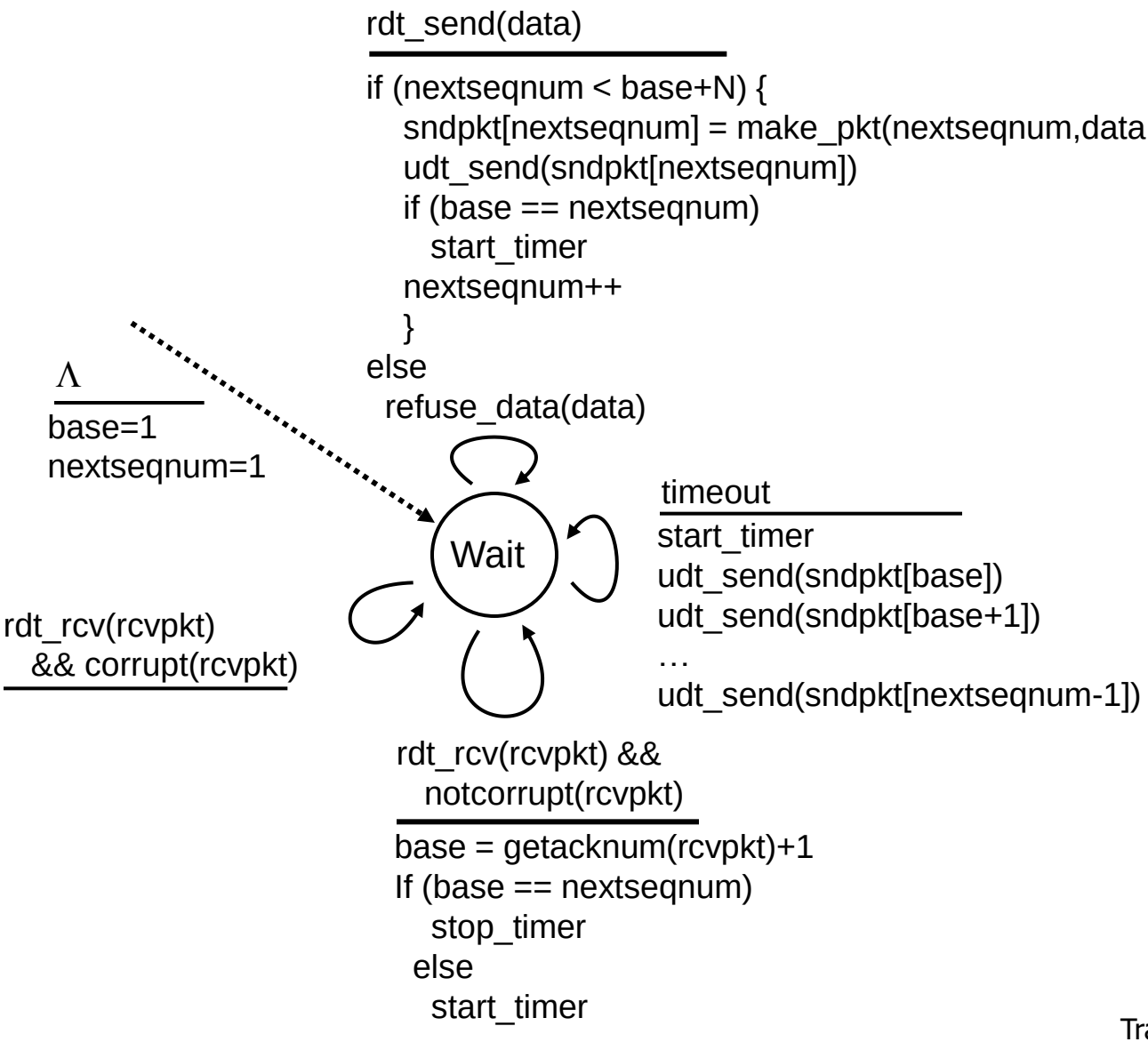
Go-Back-N: sender

- k-bit seq # στην κεφαλίδα του πακέτου
- Επιτρέπεται ένα «παράθυρο» μέχρι, συνεχόμενα πακέτα χωρίς ack.

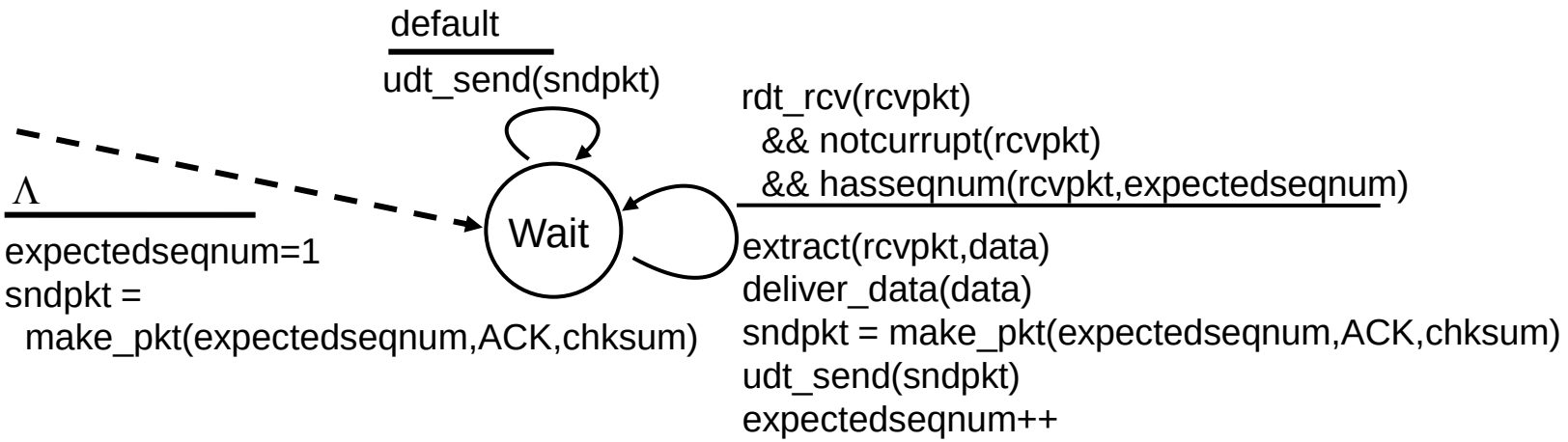


- ACK(n): αναγνώριση όλων των πακέτων μέχρι το seq # n - *"cumulative ACK" (αθροιστικό ACK)*
  - Μπορεί να λάβει διπλά ACKs
- timer για το αρχαιότερο πακέτο σε εκκρεμότητα
- *timeout(n):* αποστολή ξανά του πακέτου n και όλων με μεγαλύτερο αριθμό seq # στο παράθυρο.

GBN: Διάγραμμα λειτουργίας sender



GBN: Διάγραμμα λειτουργίας receiver

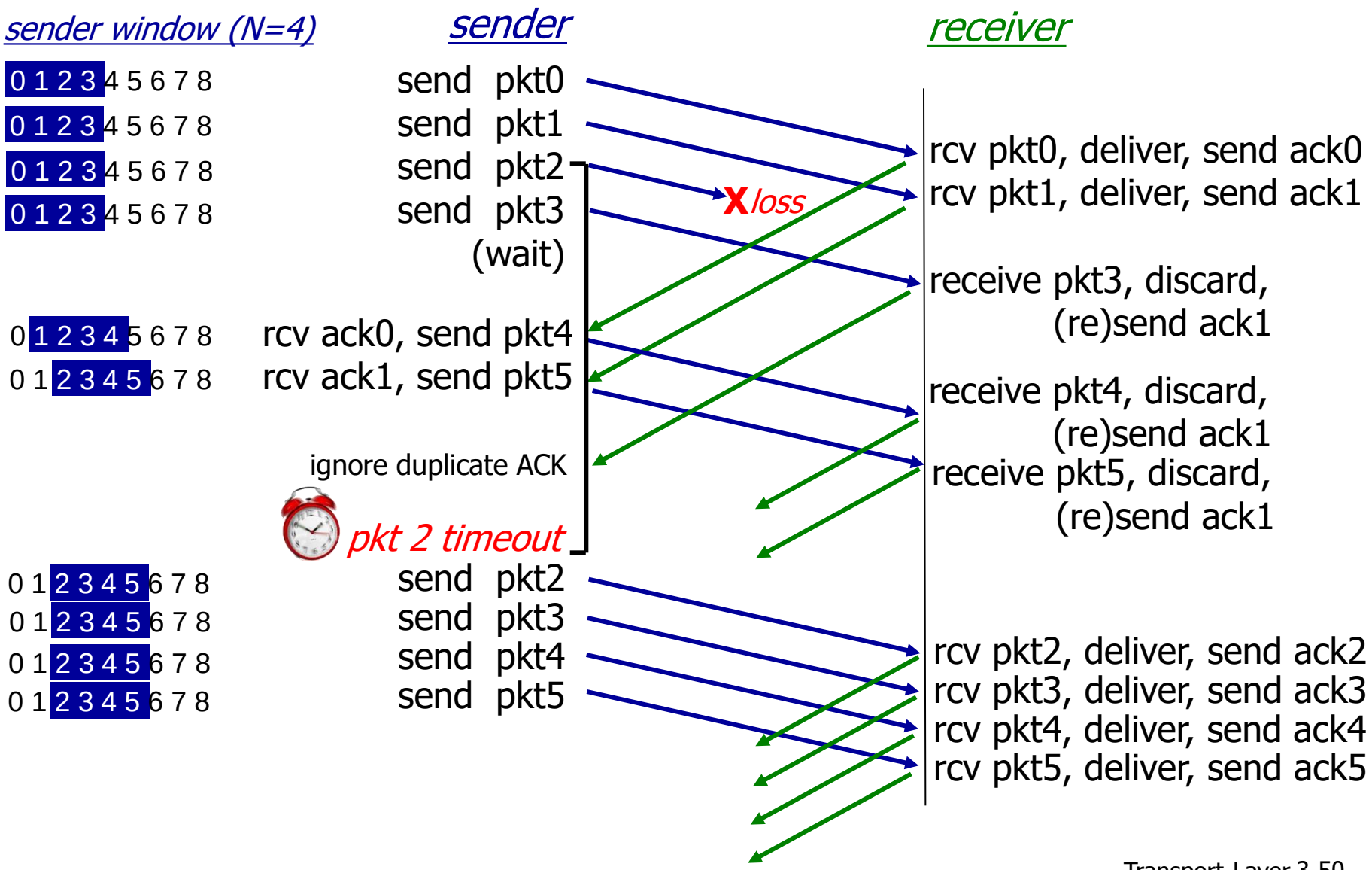


Μόνο ACK: στέλνει πάντα ACK για το πακετο με το μεγαλύτερο seq # που παραλήφθηκε σωστά και στην σειρά

- Μπορεί να δημιουργήσει διπλά ACKs
- Πρέπει να θυμάται μόνο το **expectedseqnum**

- Πακέτο εκτός σειράς:
  - Απόρριψη : *δεν υπάρχει προσωρινή αποθήκευση στο receiver!*
  - Επανάληψη ACK για πακέτο με τον μεγαλύτερο αριθμό σε σειρά.

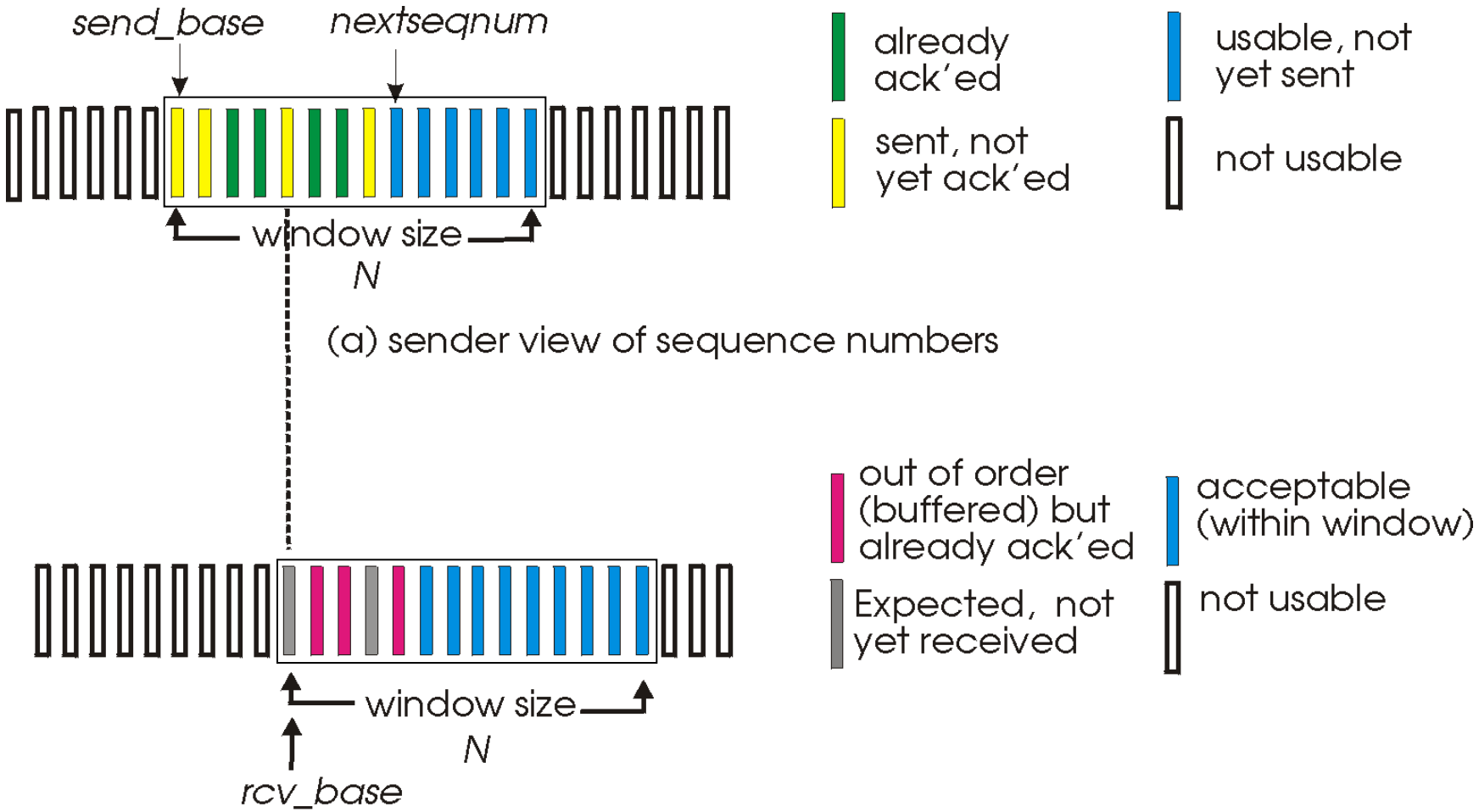
GBN σε λειτουργία



Selective repeat(επιλεκτική επανάληψη)

- Ο receiver αναγνωρίζει *ξεχωριστά* όλα τα πακέτα που παραλαμβάνει σωστά.
  - Αποθηκεύει πακέτα, όταν χρειάζεται, για να τα παραδώσει στην σειρά στο ανώτερο επίπεδο
- Ο sender στέλνει ξανά μόνο πακέτα για τα οποία δεν έχει πάρει ACK
  - Ο sender έχει timer για κάθε πακέτο
- Παράθυρο του sender
  - *N* μη συνεχόμενα seq #
  - Περιορίζει τους seq #s των απεσταλμένων, μη αναγνωρισμένων πακέτων.

Selective repeat: παράθυρα του sender και , receiver



Selective repeat

sender

data από ανώτερο επιπεδο:

- Αν το seq # είναι μεσα στα όρια του παράθυρου, αποστολή πακέτου

timeout(n):

- Επανάληψη αποστολής πακέτου n, νεα αρχή του timer

ACK(n) είναι μεταξύ [sendbase,sendbase+N]:

- Σημίωσε το πακέτο ως παραληφθεν
- Αν το n είναι το μικρότερο πακέτο χωρίς ACK, μετακίνησε τη βάση του παράθυρου στο επόμενο seq # χωρίς ACK

receiver

pkt n μεταξύ [rcvbase, rcvbase+N-1]

- Αποστολή ACK(n)
- Εκτός σειράς: αποθήκευση σε buffer
- Στη σειρά: παράδοση (επίσης παράδοση αποθηκευμένων σε σειρά πακέτων), μετακίνησε τη βάση του παράθυρου στο επόμενο μη παραληφθέν πακέτο

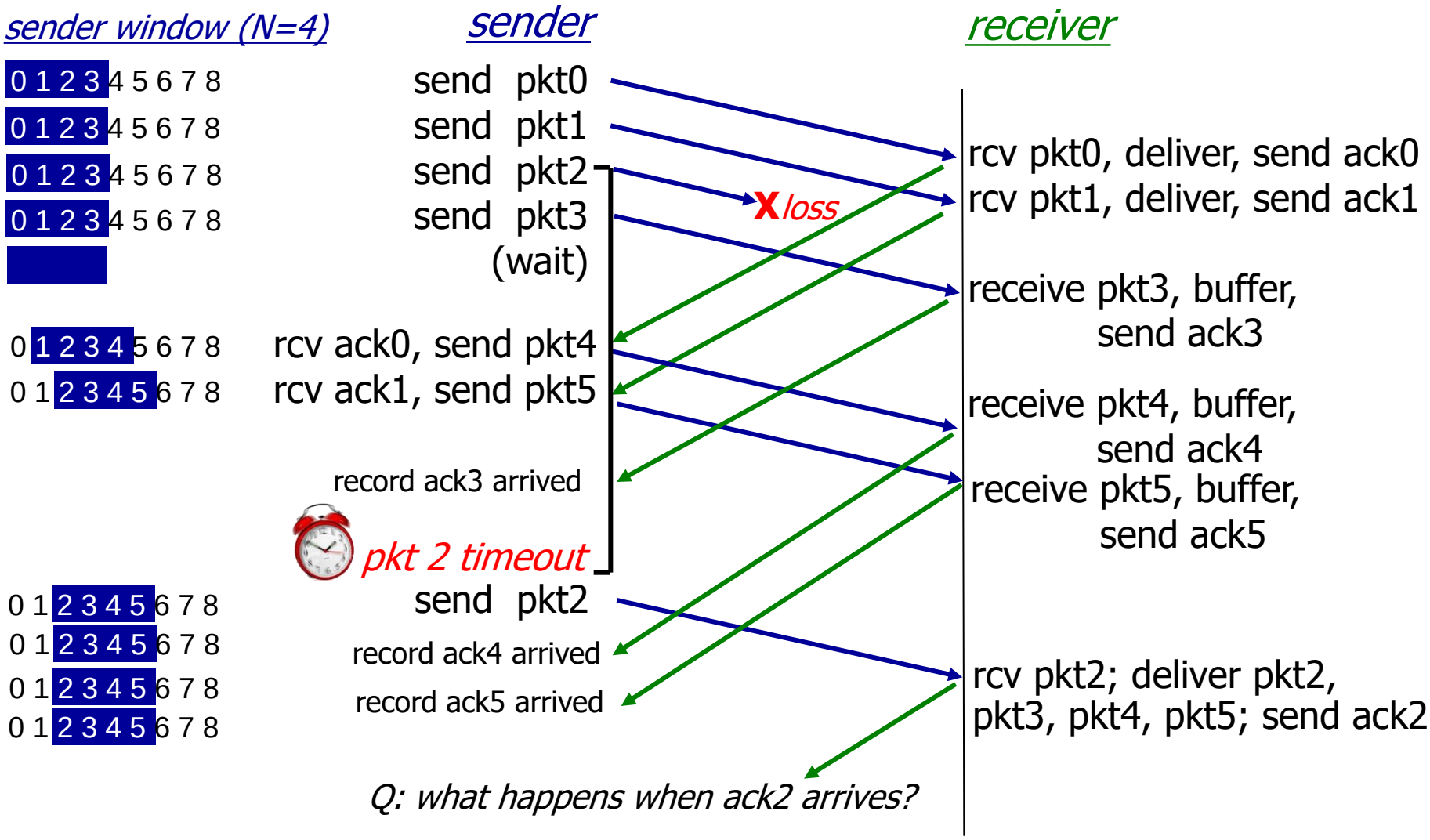
pkt n μεταξύ [rcvbase-N,rcvbase-1]

- Αποστολή ACK(n)

αλλιώς:

- Αγνόησε.

Selective repeat σε λειτουργία

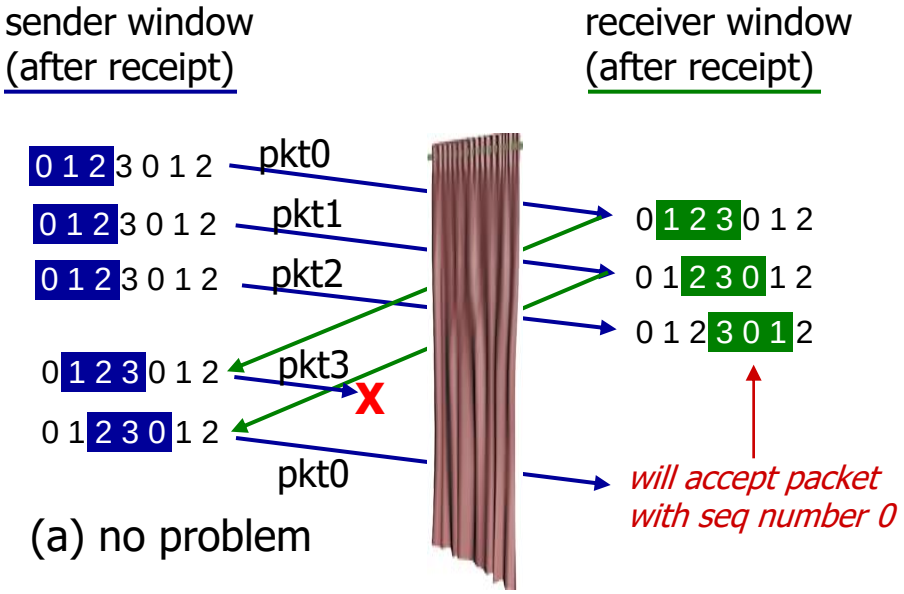


Selective repeat: dilemma

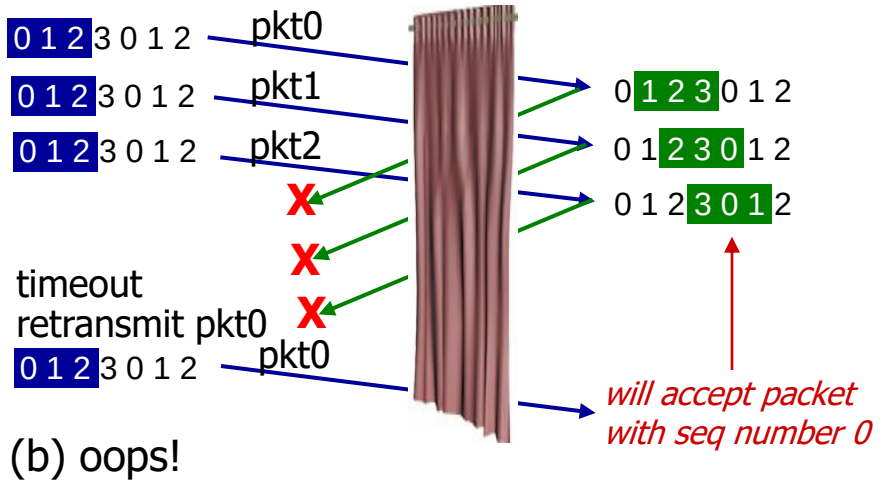
example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?



receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!



Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

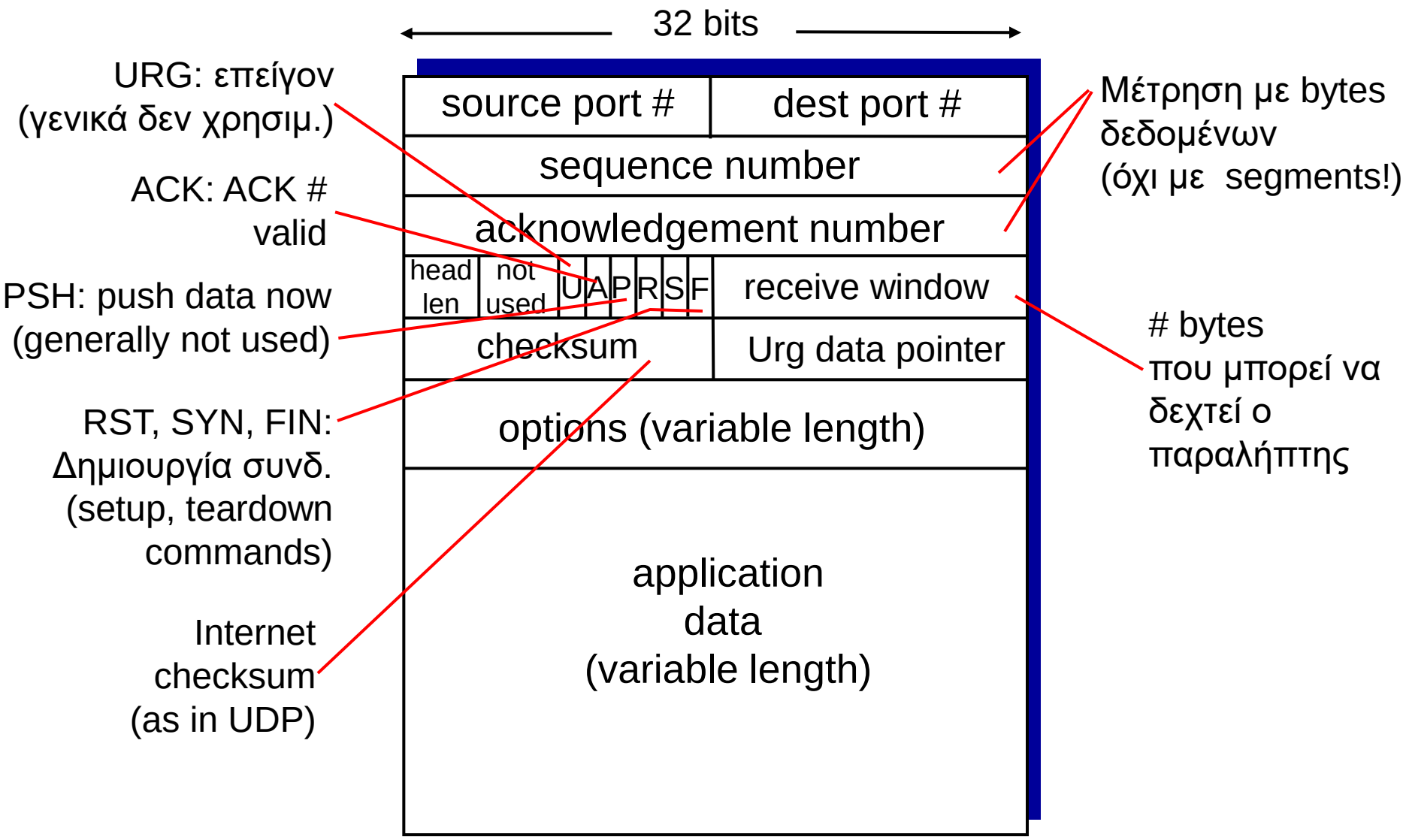
3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

TCP: Επισκόπηση RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - Ένας sender, ένας receiver
- αξιόπιστη, σε -σειρά *ροή δεδομένων:*
  - no "message boundaries"
- pipelined:
  - Ο έλεγχος συμφόρησης και ροής του TCP καθορίζουν το μέγεθος του παράθυρου
- full duplex data:
  - Αμφίδρομη ροή δεδομένων στην ίδια σύνδεση
  - MSS: Μέγιστο μέγεθος segment.
- συνδεσμοστραφές:
  - Χειραψία (handshaking – ανταλλαγή μηνυμάτων ελέγχου) αρχικοποιέι την κατάσταση του sender και receiver πριν την ανταλλαγή δεδομένων
- Ελεγχόμενη ροή:
  - Ο sender δεν θα υπερφορτώσει τον receiver

Δομή του TCP segment



TCP seq. numbers, ACKs

sequence numbers:

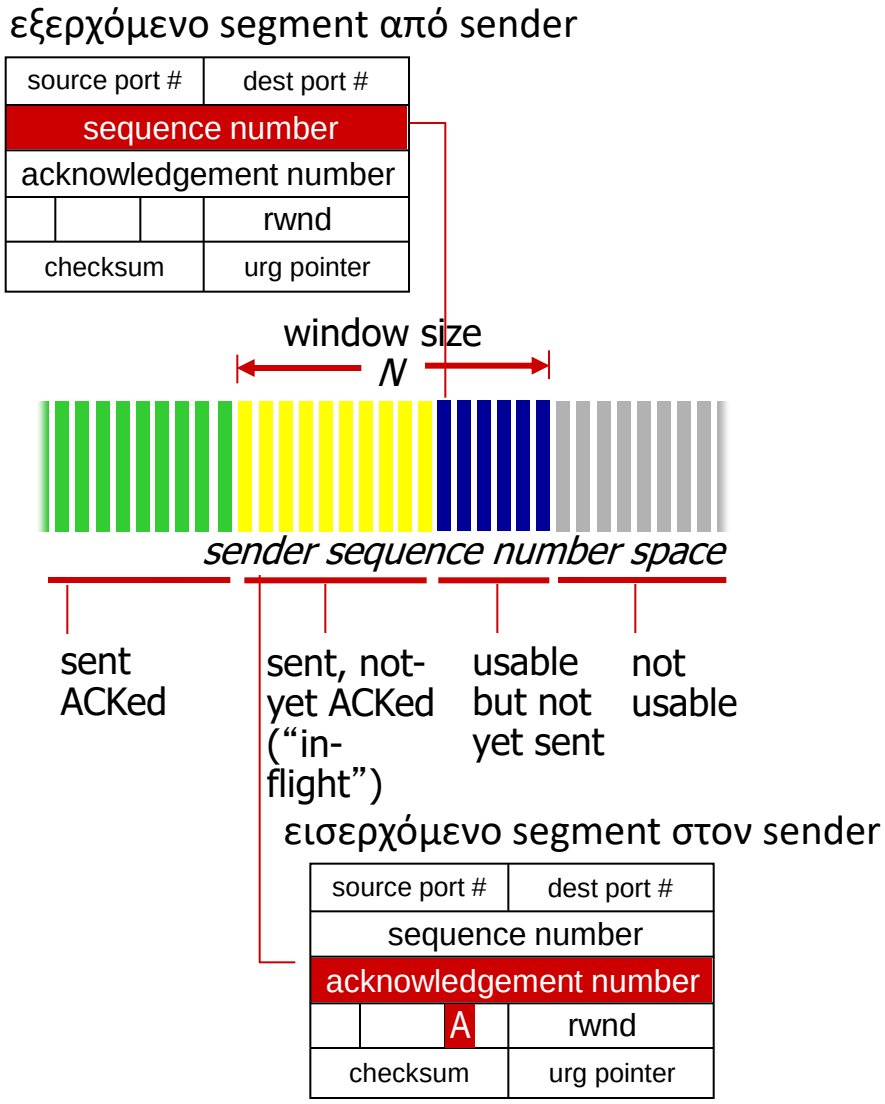
- Αριθμός του πρώτου byte στο segment

acknowledgements:

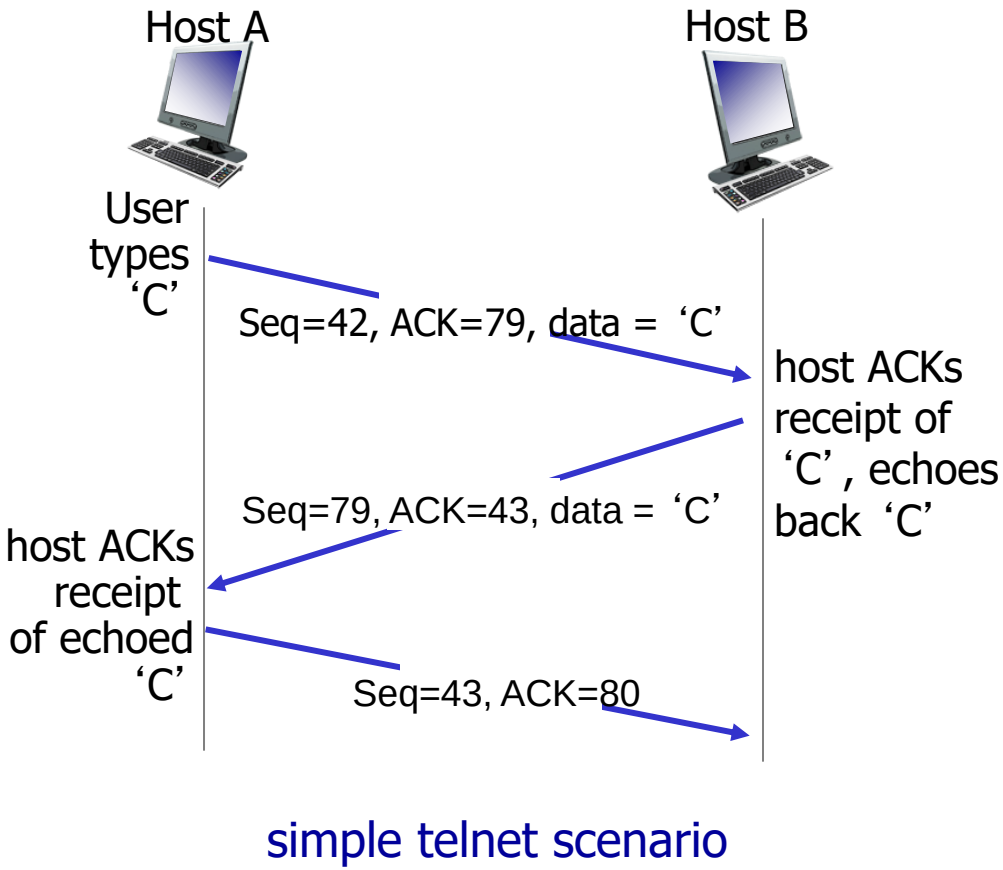
- seq # επόμενου byte που αναμένεται από το άλλο άκρο
- Αθροιστικό ACK

E: πως χειρίζεται ο receiver segments εκτός σειράς ;

- A: TCP δεν το προσδιορίζει. Επαφίεται στον προγραμματιστή.



TCP seq. numbers, ACKs



simple telnet scenario

TCP round trip time, timeout

E: πως ορίζεται η τιμή timeout στο TCP?

- Μεγαλύτερη από RTT
  - Αλλά το RTT μεταβάλλεται
- *Πολύ μικρή:* πρώιμο timeout, περιττές επαναμεταδόσεις
- *Πολύ μεγάλη:* αργή αντίδραση σε απώλειες πακέτων

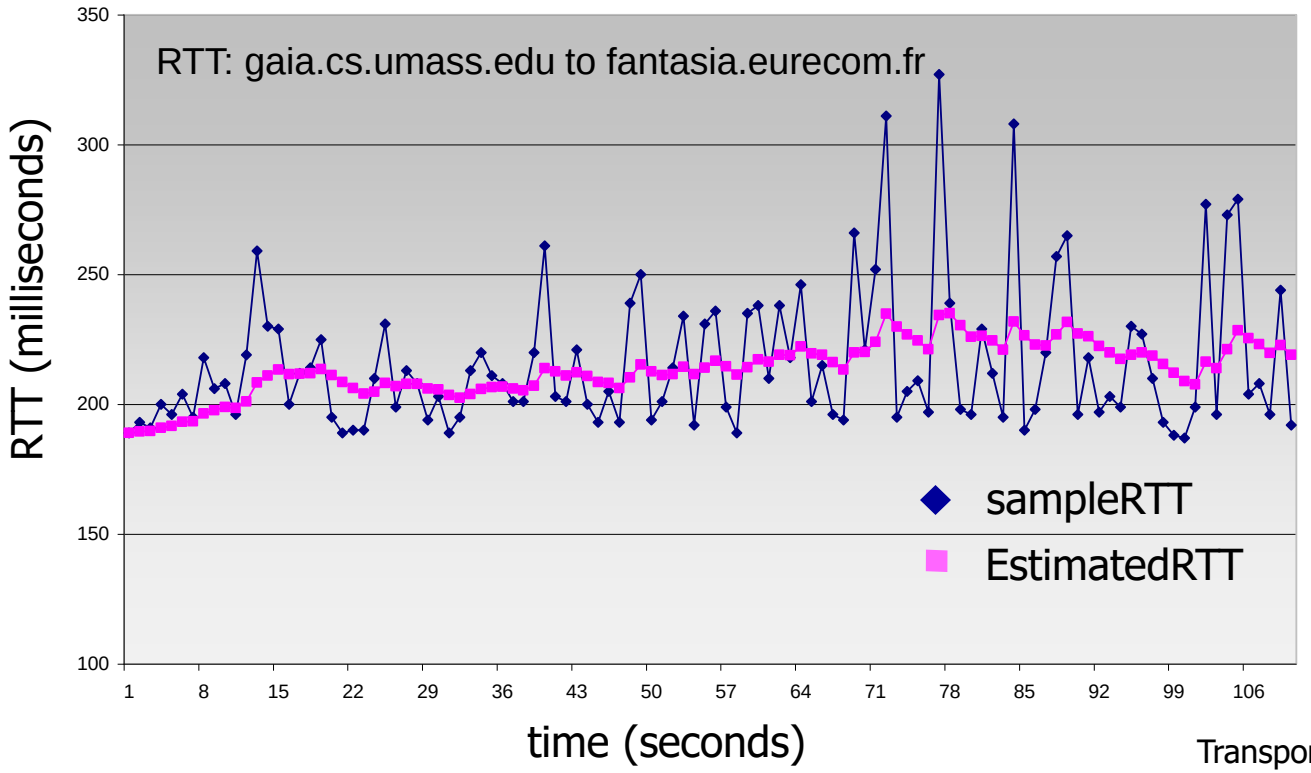
E: πως μπορεί να εκτιμηθεί το RTT?

- **SampleRTT**: μετρούμενος χρόνος από την αποστολή του μέχρι την λήψη του ACK
  - Αγνοούνται επαναλήψεις μεταδόσεων
- **SampleRTT** μεταβάλλεται, θέλουμε ο εκτιμώμενος χρόνος RTT να είναι "πιο ομαλός"
  - Μέσος όρος αρκετών πρόσφατων μετρήσεων όχι μόνο ο τρέχων **SampleRTT**

TCP round trip time, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- Εκθετικά σταθμισμένος κινητός μέσος όρος.
- Η επήρεια παλαιοτέρων μετρήσεων φθίνει εκθετικά
- Τυπική τιμή : α = 0.125



TCP round trip time, timeout

- Χρόνος timeout : **EstimatedRTT** συν "περιθώριο ασφαλείας"
  - Μεγάλες μεταβολές του**EstimatedRTT** -> μεγαλύτερο περιθώριο ασφαλείας.
- εκτίμηση της απόκλισης SampleRTT από τον EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
                β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)
```

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

Εκτιμώμενος RTT

"περιθώριο ασφαλείας"

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

TCP αξιόπιστη μεταφορά δεδομένων

- TCP δημιουργεί μια rdt υπηρεσία πάνω από την μη αξιόπιστη υπηρεσία του IP
  - pipelined segments
  - Αθροιστικά acks
  - Ένας μετρητής για επανάληψη μετάδοσης
- Επαναλήψεις ενεργοποιούνται από:
  - timeout
  - διπλά acks

Ας θεωρήσουμε αρχικά ένα απλοποιημένο TCP αποστολέα:

- Αγνοούμε διπλά acks
- Αγνοούμε έλεγχο ροής και έλεγχο συμφόρησης

Συμβάντα σε TCP sender:

Δεδομένα από την εφαρμογή:

- Δημιουργία segment με seq #
- seq # είναι ο αριθμός του πρώτου byte δεδομένων στο segment
- Αρχίζει ο timer αν δεν τρέχει ήδη
  - Θεωρήστε ότι ο timer μετρά από το τελευταίο segment χωρίς ACK
  - Χρονικό διάστημα : **TimeOutInterval**

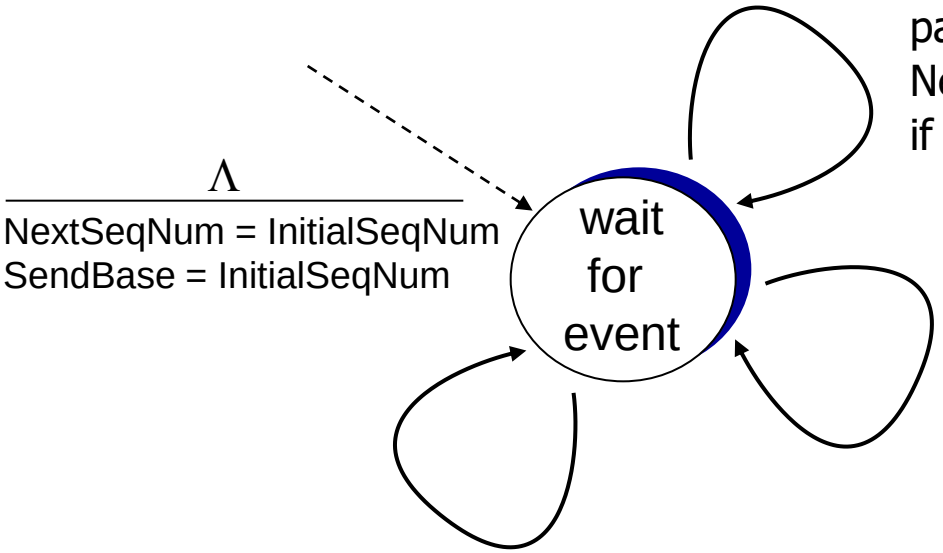
timeout:

- ξαναστέλνουμε το segment που προκάλεσε timeout
- Ξαναρχίζουμε τον timer

Λήψη ack:

- Αν το ack αναφέρεται σε segments που δεν είχε επιβεβαιωθεί προηγουμένως.
  - Ενημερώνουμε τι έχει επιβεβαιωθεί μέχρι τώρα
  - Αρχίζουμε τον timer αν υπάρχουν segments χωρίς ACK.

TCP sender (απλοποιημένος)



data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

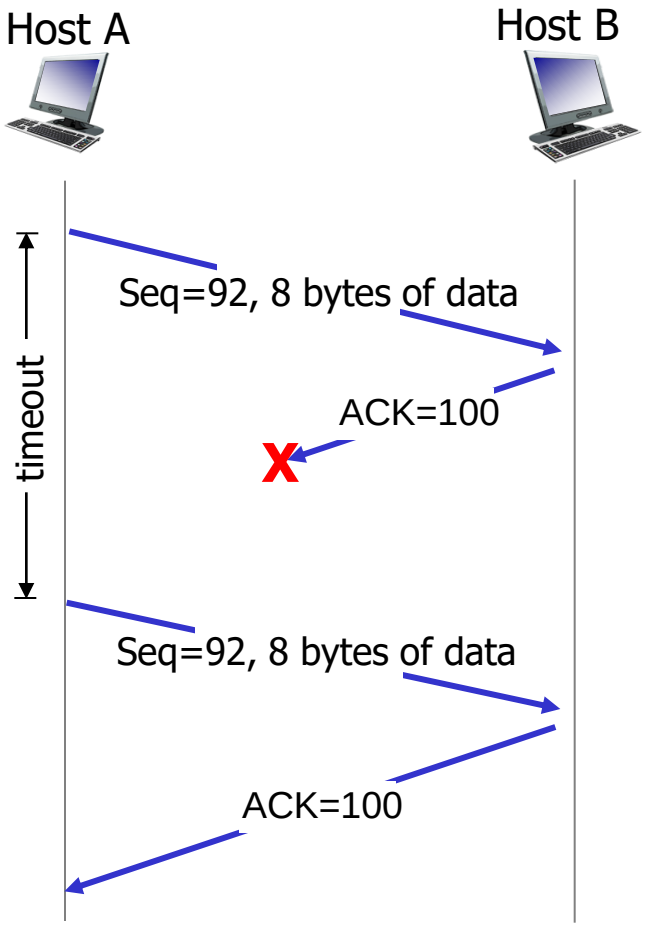
timeout

retransmit not-yet-acked segment
with smallest seq. #
start timer

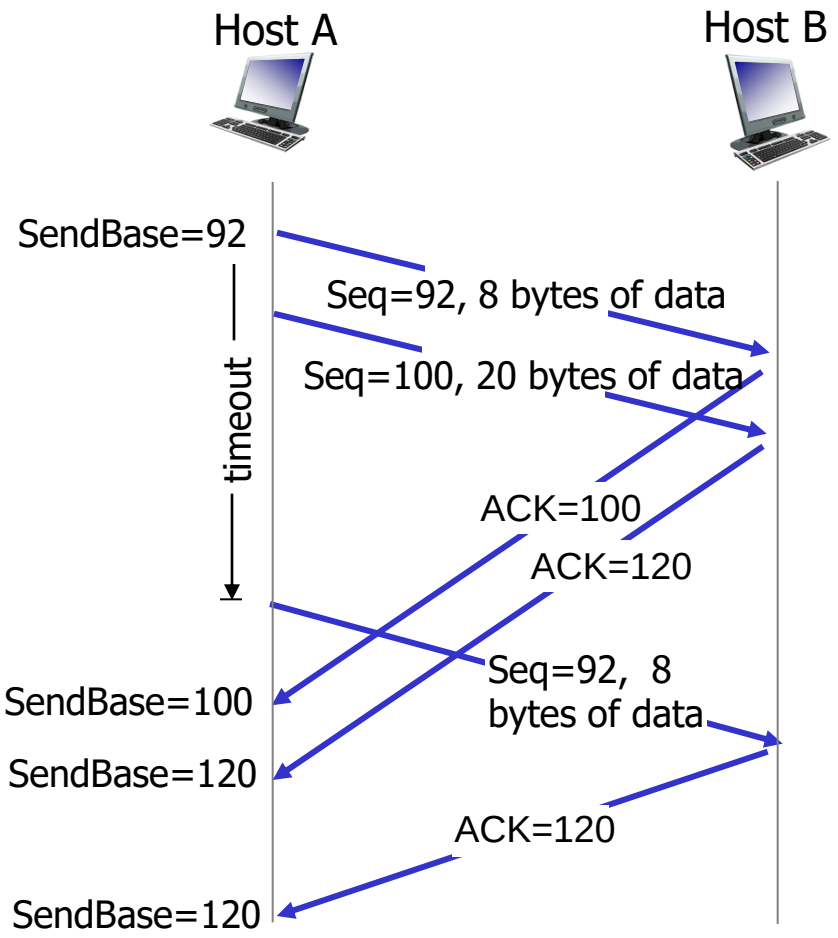
ACK received, with ACK field value y

```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: last cumulatively ACKed byte */
   if (there are currently not-yet-acked segments)
       start timer
     else stop timer
    }
```

TCP: περιπτώσεις επανάληψης μετάδοσης

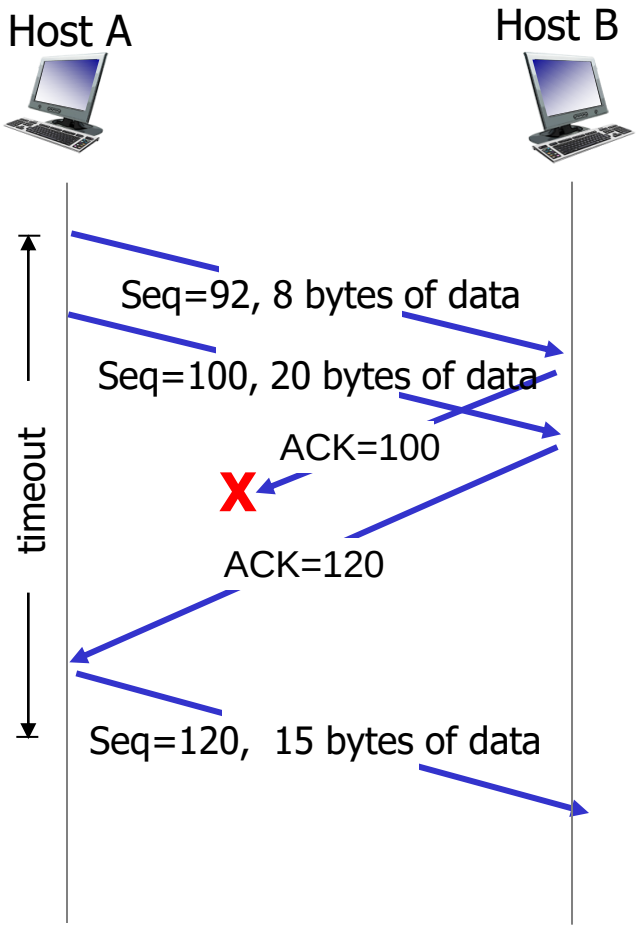


Περίπτωση απώλειας ACK



Πρώιμο timeout

TCP: περιπτώσεις επανάληψης μετάδοσης



αθροιστικό ACK

Δημιουργία TCP ACK [RFC 1122, RFC 2581]

Συμβάν στον receiver	*Ενέργεια του TCP receiver*
Άφιξη segment στη σειρά με αναμενόμενο seq #. Όλα τα Δεδομένα μέχρι seq # έχουν επιβεβαιωθεί.	Καθυστερημένο ACK. Περιμένει μέχρι 500ms για επόμενο segment. Αν δεν έλθει στέλνει ACK
Άφιξη segment στη σειρά με αναμενόμενο seq #.Για ένα άλλο segment εκκρεμεί ACK	Στέλνει αμέσως ένα αθροιστικό ACK, Επιβεβαιώνοντας και τα δύο segments σε σειρά
Άφιξη segment εκτός σειράς με seq #. μεγαλύτερο από Αναμενόμενο. Ανιχνεύτηκε κενό	Στέλνει αμέσως *duplicate ACK*, δηλώνοντας το seq. # του byte το οποίο αναμένεται
Άφιξη ενός segment που μερικά ή ολικά συμπληρώνει ένα κενό	Στέλνει αμέσως ACK, με την προϋπόθεση ότι το segment ξεκινά στο κάτω άκρο του κενού

TCP γρήγορη επανάληψη αποστολής(fast retransmit)

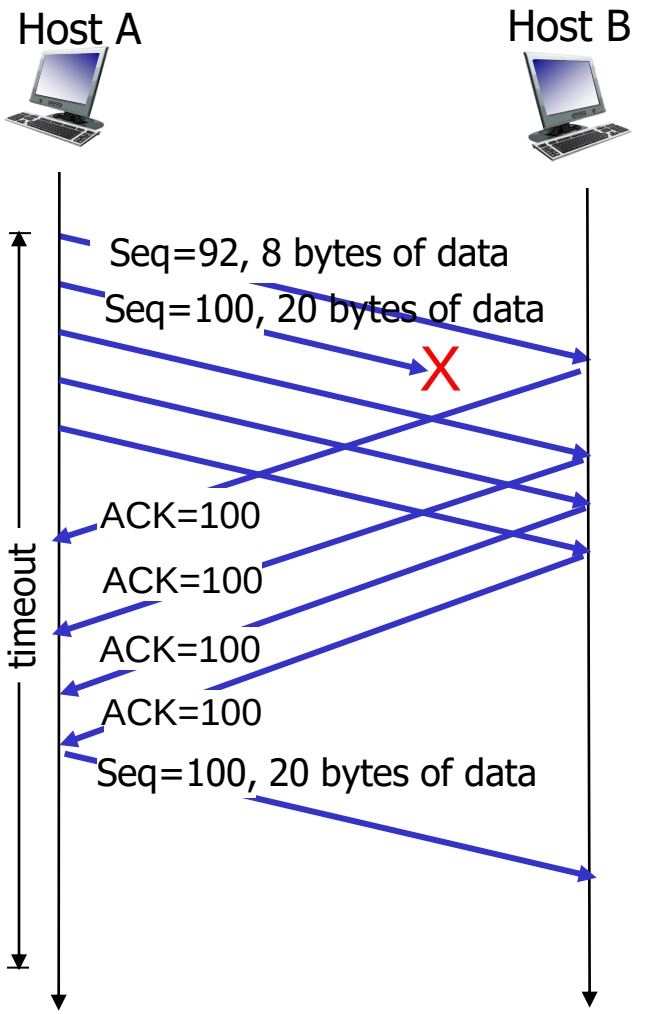
- Συχνά η περίοδος time-out είναι σχετικά μεγάλη:
  - Μεγάλη καθυστέρηση πριν ξαναστα λθεί χαμένο πακέτο
- Ανίχνευση χαμένων segments μέσω διπλών ACKs.
  - Ο sender συχνά στέλνει πολλά segments το ένα πίσω από το άλλο.
  - Αν κάποιο segment χαθεί, πιθανότατα θα υπάρξουν πολλά διπλά ACKs.

TCP fast retransmit

Αν ο sender λάβει 3 ACKs για τα ίδια δεδομένα (“triple duplicate ACKs”), τότε ξαναστέλνει το μη επιβεβαιωμένο segment με τον μικρότερο seq #

- Πιθανότατα αυτό το segment έχει χαθεί, οπότε δεν περιμένει για timeout

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη -συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

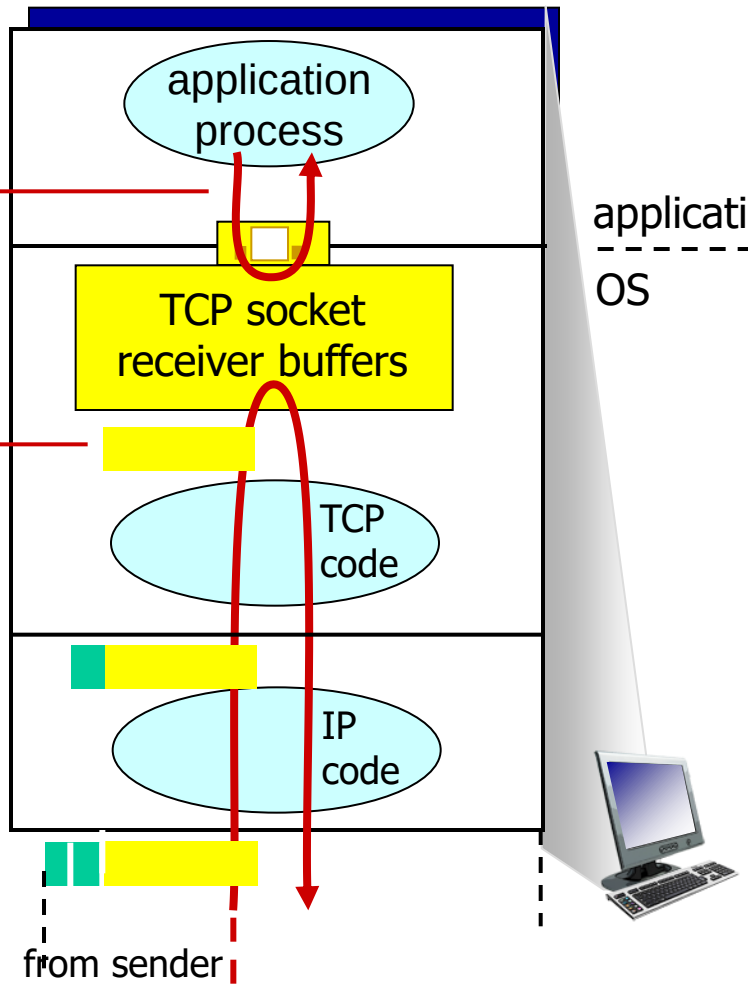
3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

TCP ελεγχος ροής

Η εφαρμογή μπορεί να αφαιρεί τα δεδομένα από τους buffers της TCP socket....

... πιο αργά από όσο τα μεταφέρει ο TCP receiver. (στέλνει ο sender)



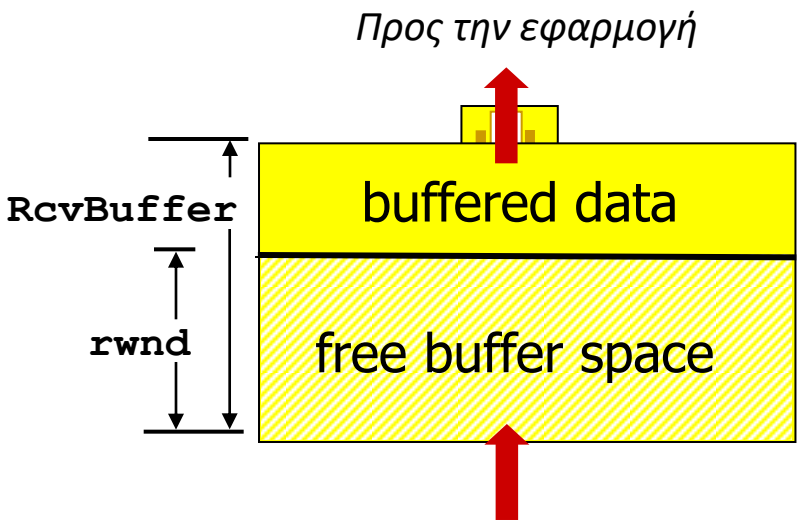
receiver protocol stack

Ελεγχος ροής

Ο receiver ρυθμίζει τον sender, να μη στέλνει πολλά δεδομένα και πολύ γρήγορα, ώστε να μη υπερχειλίζει ο Buffer του receiver

TCP έλεγχος ροής

- Ο receiver "ενημερώνει" για τον ελεύθερο χώρο στον buffer, με το πεδίο **rwnd** στην κεφαλίδα των segments που στέλνει στον αποστολέα.
  - Το μέγεθος του **RcvBuffer** ορίζεται με τις επιλογές του socket (τυπική προεπιλεγμένη τιμή είναι **4096** bytes)
  - Πολλά Λειτουργικά συστήματα προσαρμόζουν αυτόματα τον **RcvBuffer**
- Ο sender περιορίζει την ποσότητα των δεδομένων χωρίς επιβεβαίωση ("in-flight") στην τιμή του **rwnd** του receiver**.**
- Εγγυάται ότι ο Buffer του receiver δεν θα υπερχειλίσει.



Αποθήκευση στην πλευρά του receiver-

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

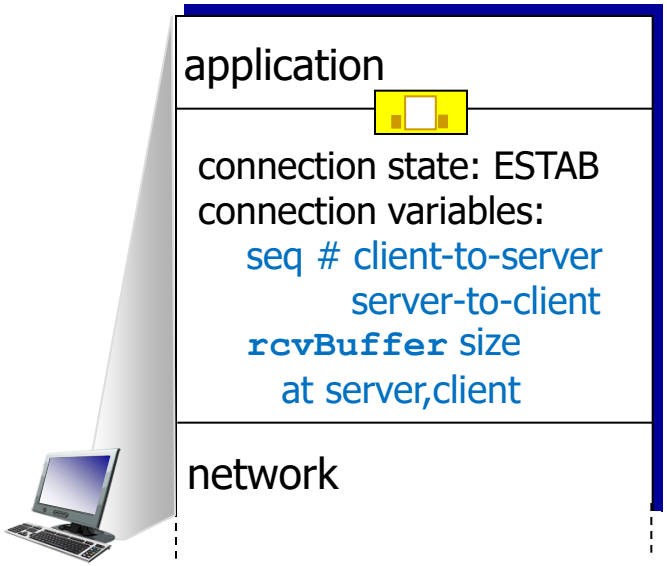
3.6 Αρχές έλεγχου συμφόρησης

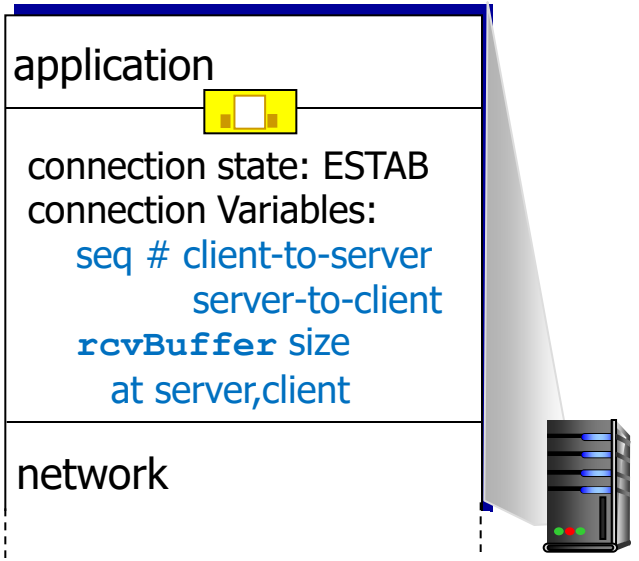
3.7 TCP έλεγχος συμφόρησης

Δημιουργία TCP Συνδέσμου

Πριν αρχίσει η ανταλλαγή δεδομένων, sender/receiver "συμφωνούν":

- Να δημιουργήσουν ένα σύνδεσμο (ο καθένας ξέρει ότι ο άλλος συμφωνεί στην δημιουργία του συνδέσμου)
- Στις παραμέτρους του συνδέσμου.



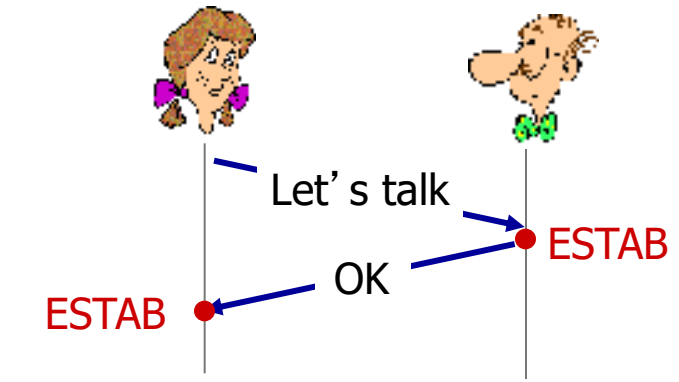


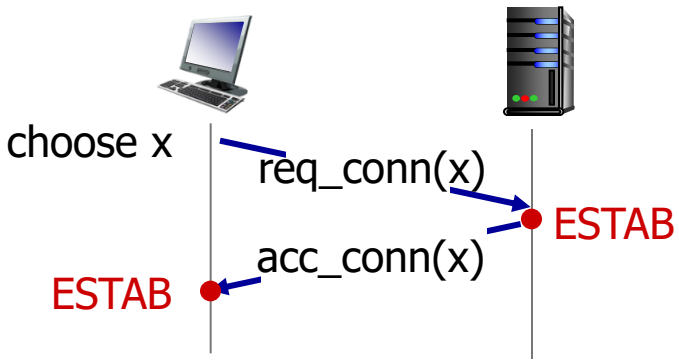

```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

```
Socket connectionSocket =
   welcomeSocket.accept();
```

Συμφωνία για δημιουργία συνδέσμου

2-way handshake:



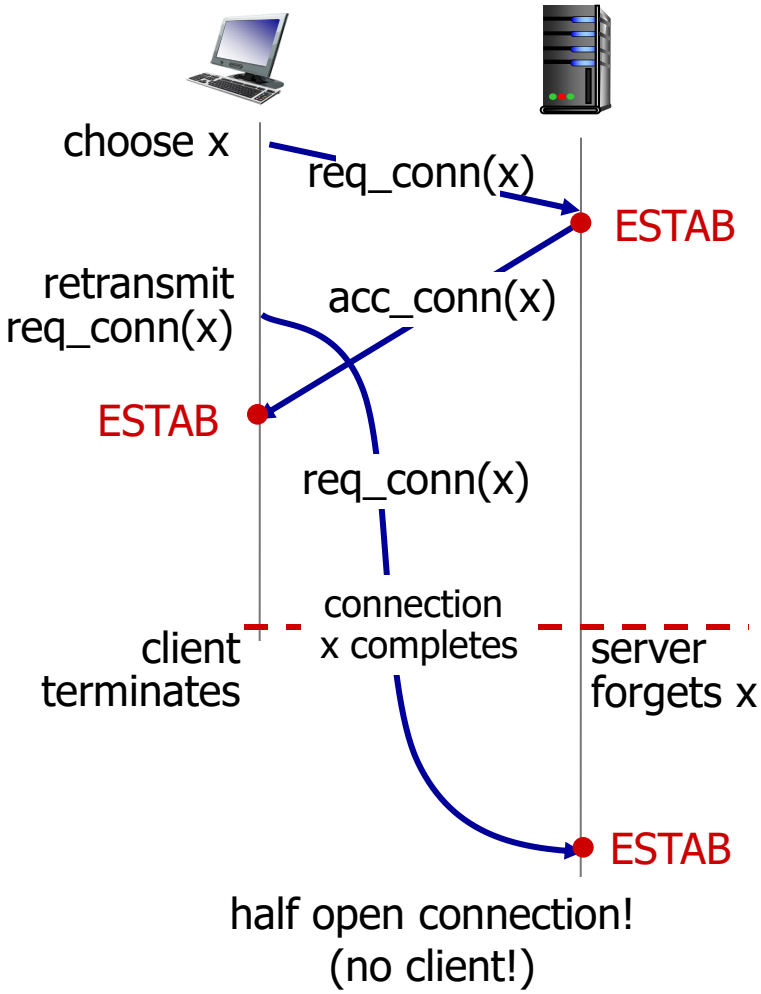


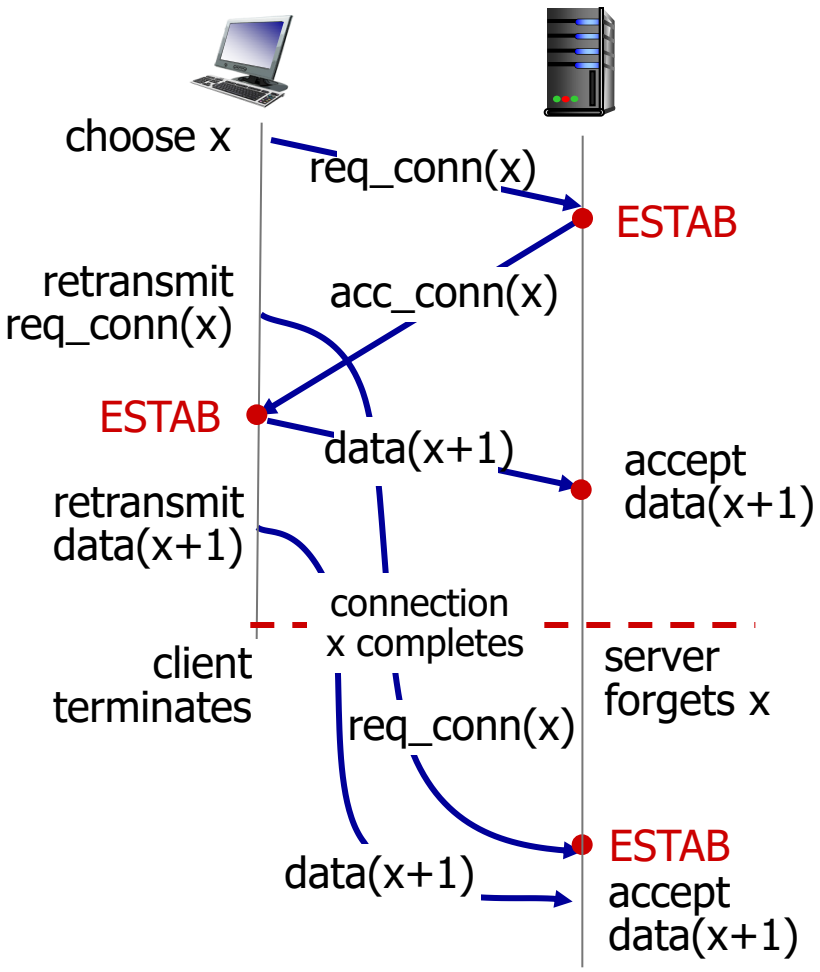
E: Λειτουργεί πάντα αυτό το μοντέλο στο δίκτυο;

- Μεταβλητές καθυστερήσεις
- Επανάληψη αιτήματος σύνδεσης λόγω απώλειας μηνύματος

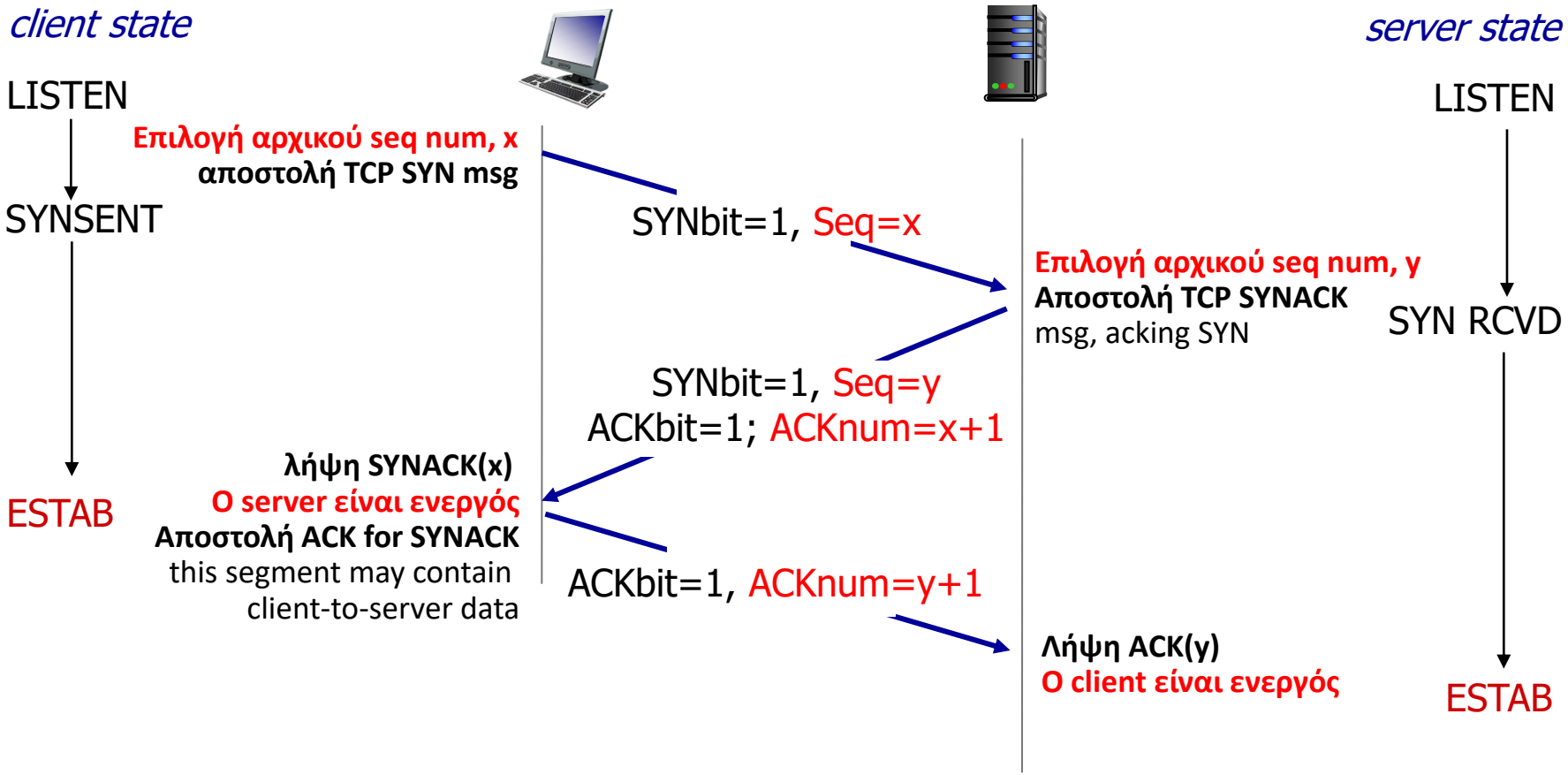
Συμφωνία για δημιουργία συνδέσμου

2-way handshake σενάρια αποτυχίας:

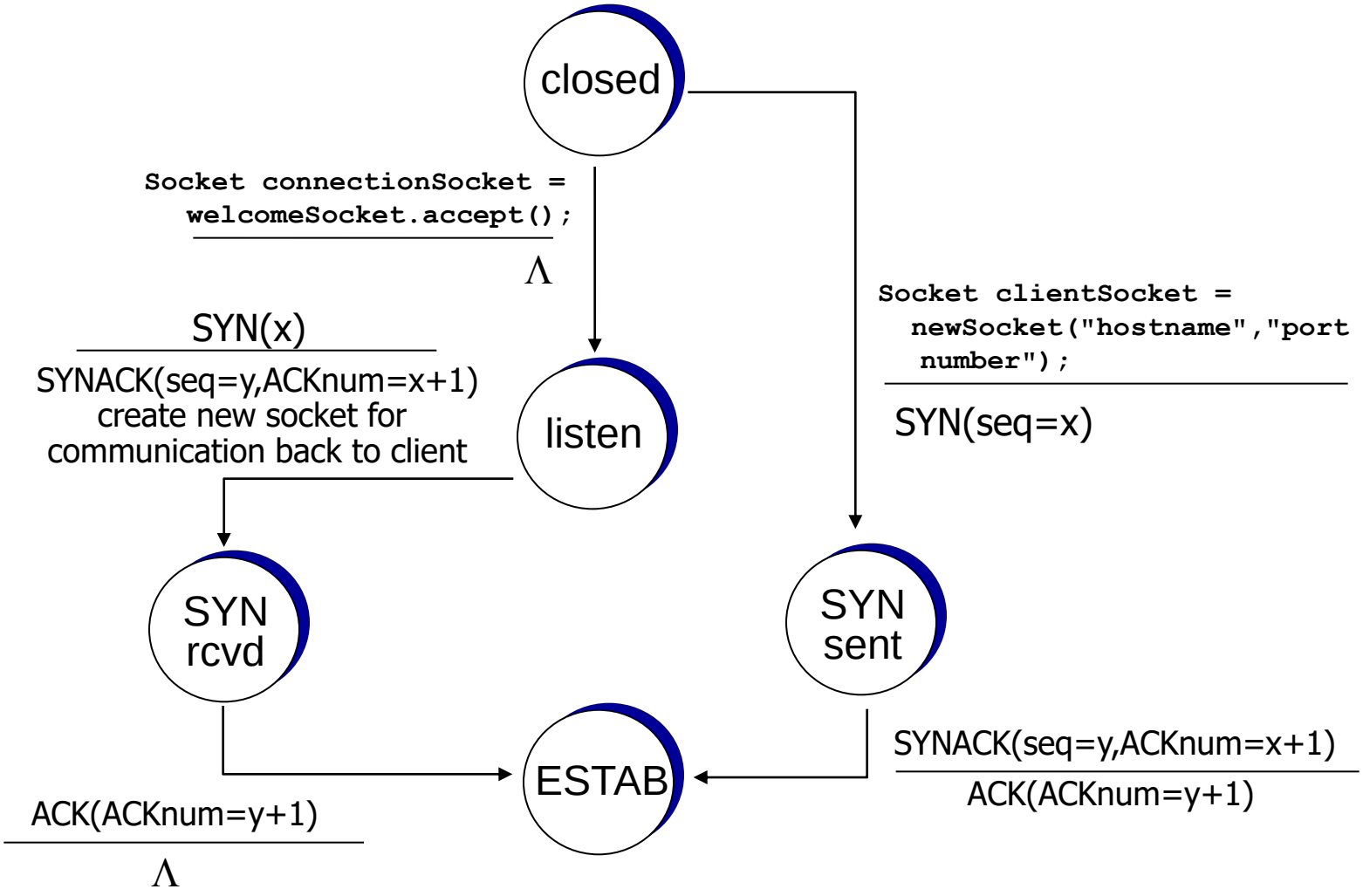




TCP τριπλή χειραψία (3-way handshake)



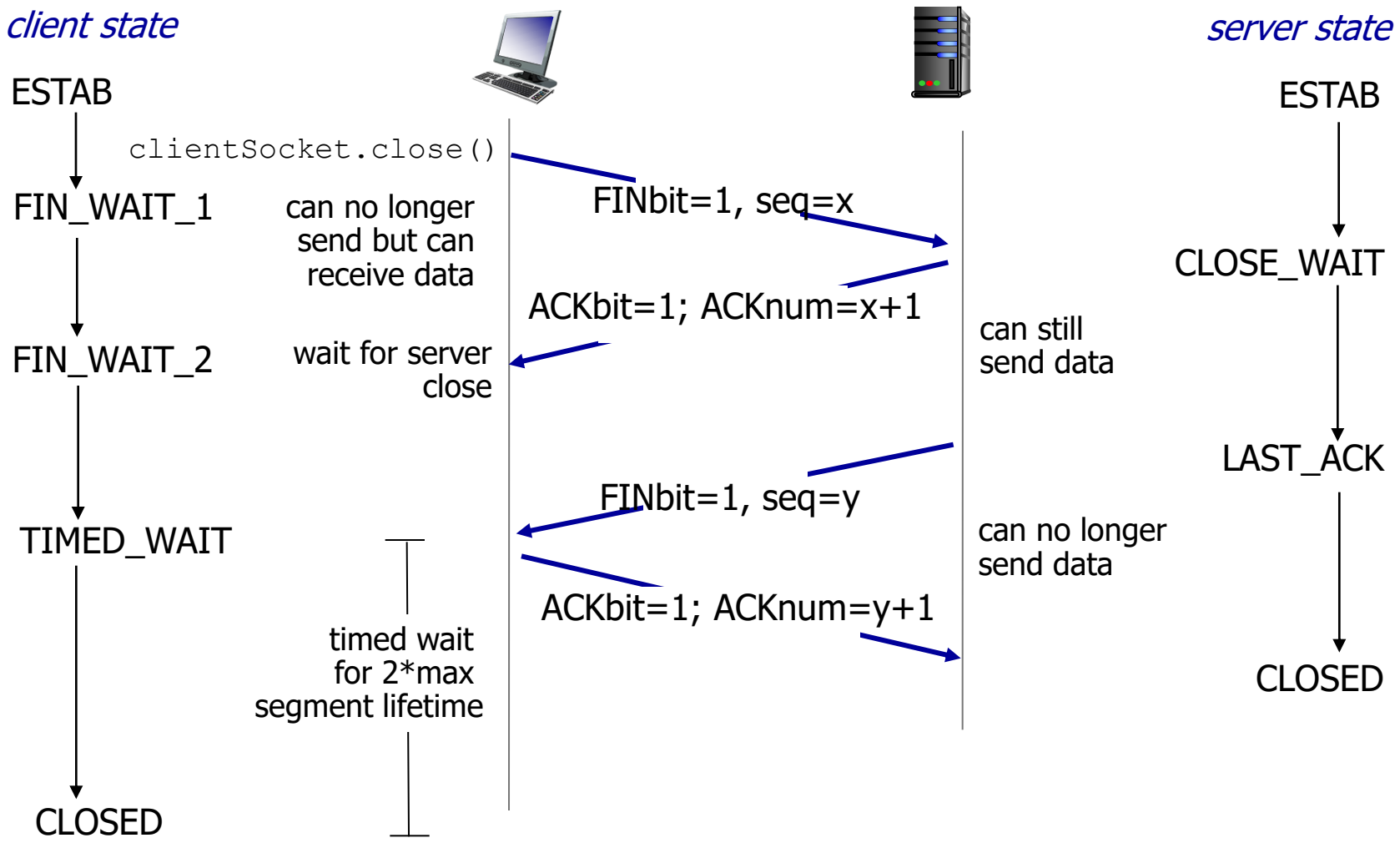
TCP 3-way handshake: FSM



TCP: Κλείσιμο ενός συνδέσμου

- Ο client και ο server κλείνουν ο καθένας την δική του πλευρά του συνδέσμου
  - στέλνουν TCP segment με το FIN bit = 1
- Απαντούν στο FIN με ένα ACK
  - Όταν πάρουν ένα FIN, το ACK μπορεί να συνδιαστεί με το δικό τους FIN
- Ταυτόχρονες ανταλλαγές FIN είναι αποδεκτές.

TCP: κλείσιμο ενός συνδέσμου



Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

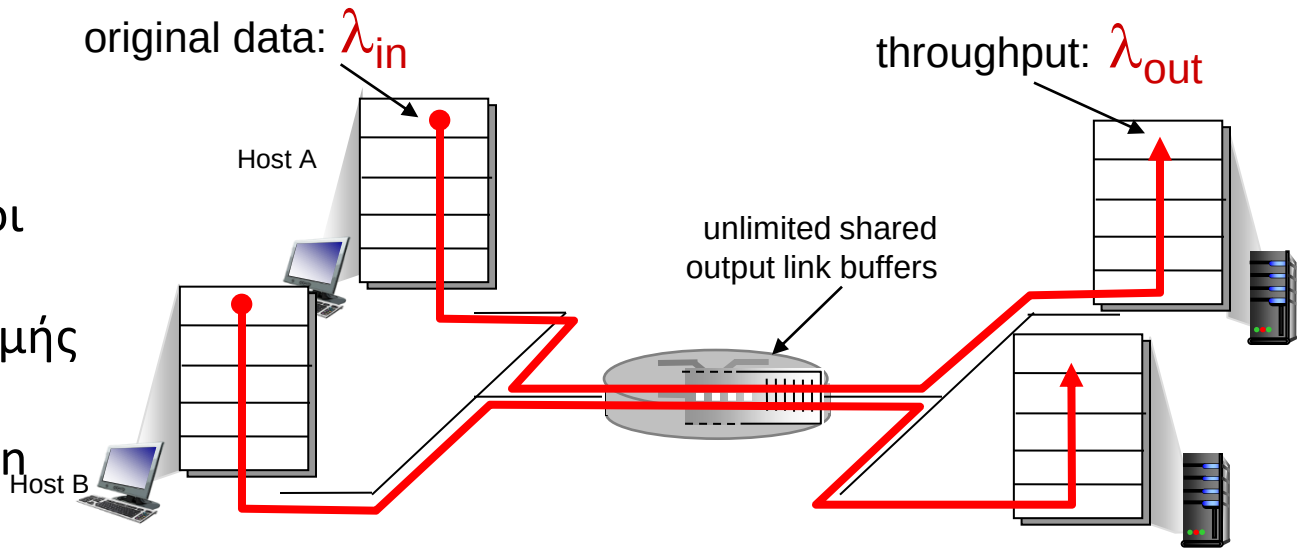
Αρχές Ελέγχου Συμφόρησης

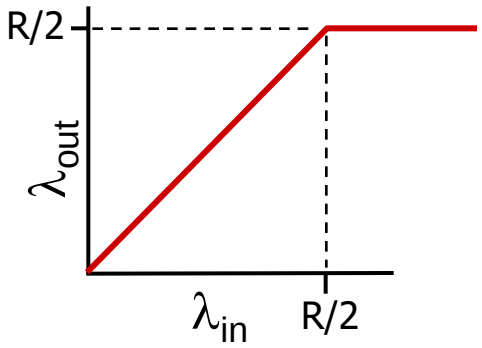
Συμφόρηση (congestion):

- "Πολλές πηγές στέλνουν πιο πολλά δεδομένα πιο γρήγορα από όσο μπορεί να διαχειριστεί το δίκτυο"
- Διαφορετική από τον έλεγχο ροής!
- Πως εκδηλώνεται :
  - Χαμένα πακέτα (υπερχείλιση buffer στους routers)
  - Μεγάλες καθυστερήσεις (ουρές στους buffers των router)
- Σημαντικό πρόβλημα στα δίκτυα!

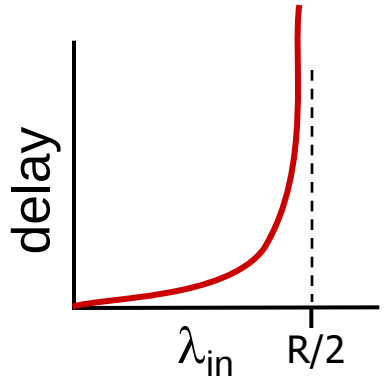
Αιτίες/κόστος Συμφόρησης: σενάριο 1

- Δυο senders, δυο receivers
- ένας router, άπειροι buffers
- Χωρητικότητα γραμμής (link capacity): R
- Χωρίς retransmission





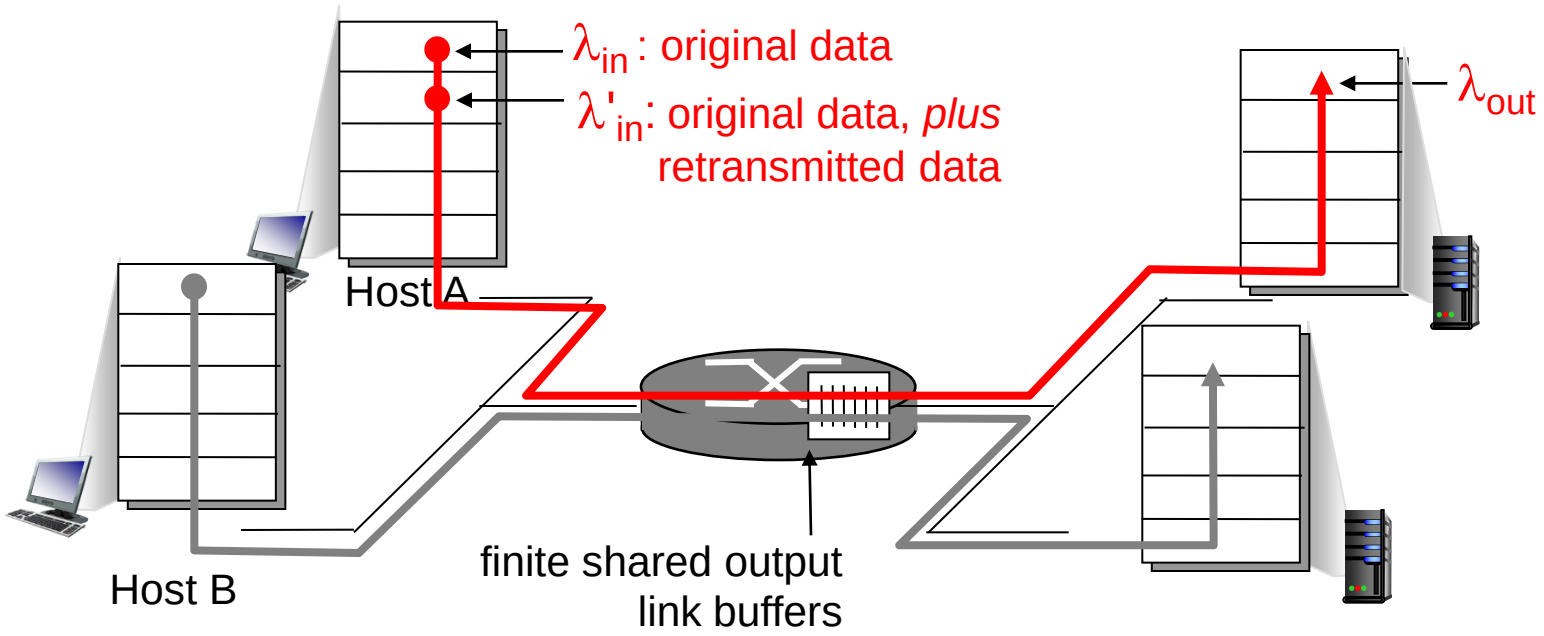
- Μέγιστος ρυθμός μετάδοσης ανά sender : R/2



- Μεγάλες καθυστερήσεις καθώς η ρυθμός άφιξης, λ_{in}, πλησιάζει την χωρητικότητα

Αιτίες/κόστος Συμφόρησης: σενάριο 2

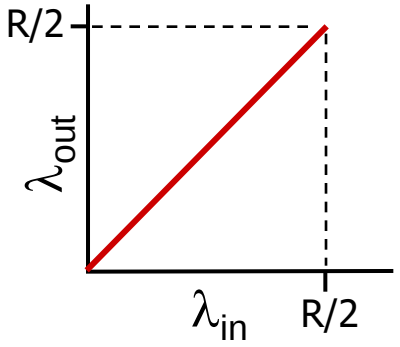
- Ένας router, *περιορισμένοι* buffers
- Ο sender ξαναστέλνει πακέτα αν καθυστερήσουν (time-out)
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input περιλαμβάνει και *retransmissions* : $\lambda'_{in} >= \lambda_{in}$

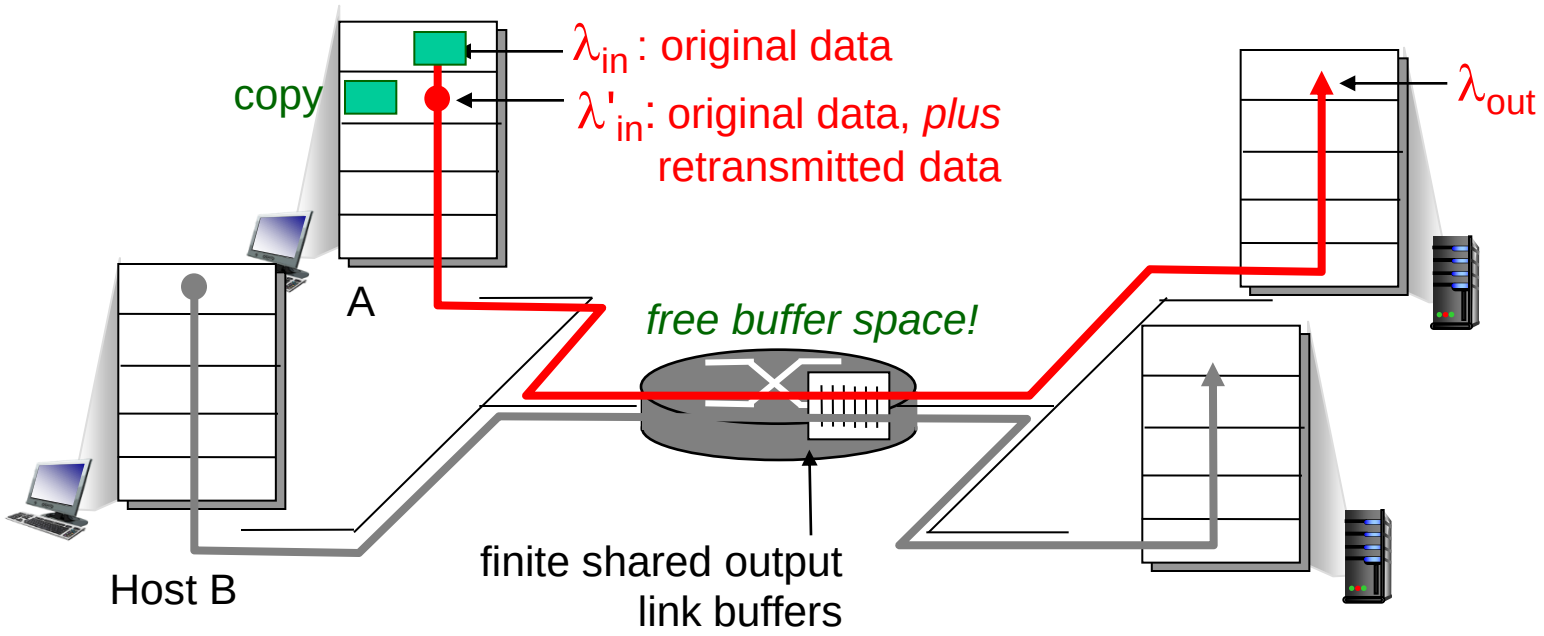


Αιτίες/κόστος Συμφόρησης: σενάριο 2

Εξιδανίκευση: πλήρης γνώση

- Ο sender στέλνει μόνο όταν οι buffers του router είναι διαθέσιμοι.



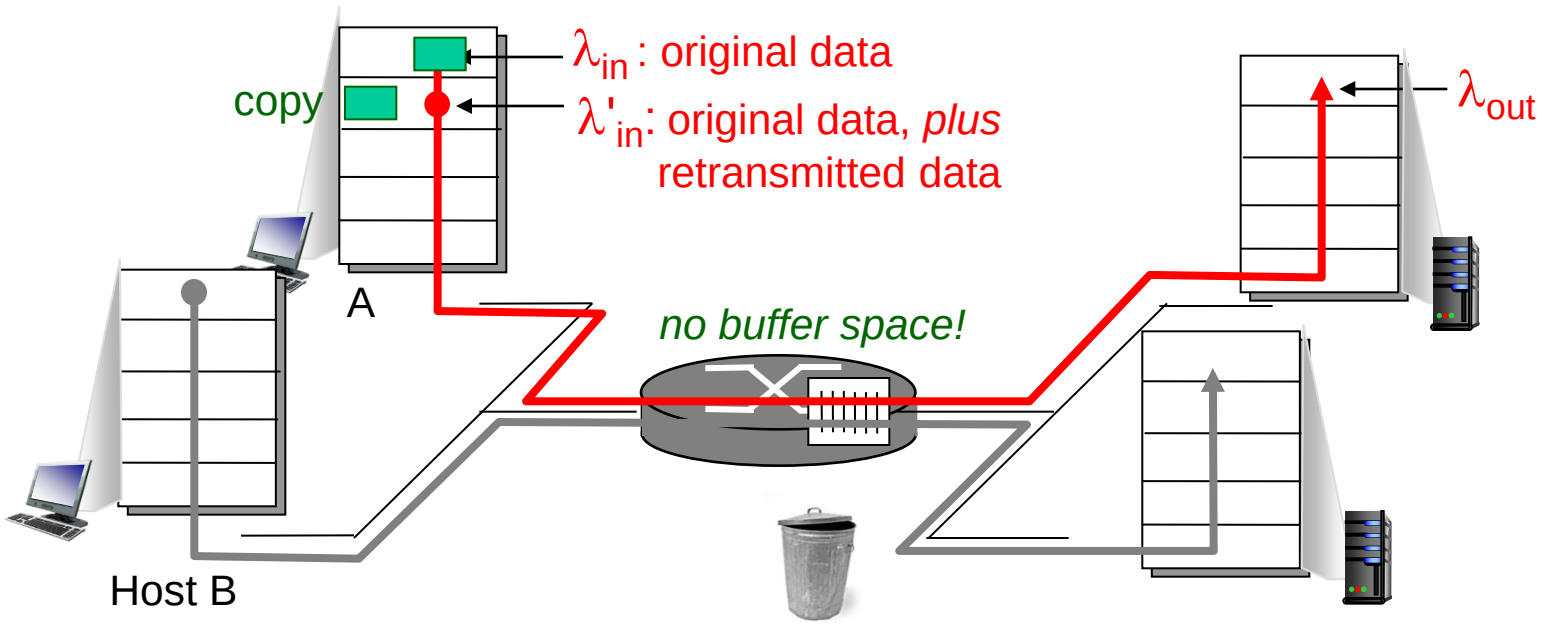


Αιτίες/κόστος Συμφόρησης: σενάριο 2

Εξιδανίκευση: γνωστές απώλειες.

πακέτα μπορεί να χαθούν, να απορριφθούν στο router λογω πληρότητας των buffers

- Ο sender ξαναστέλνει πακέτα μόνο αν ξέρει ότι έχουν χαθεί.

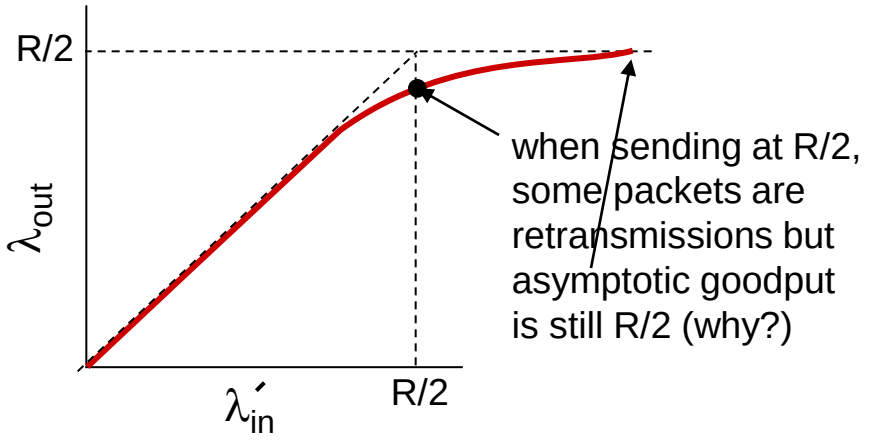


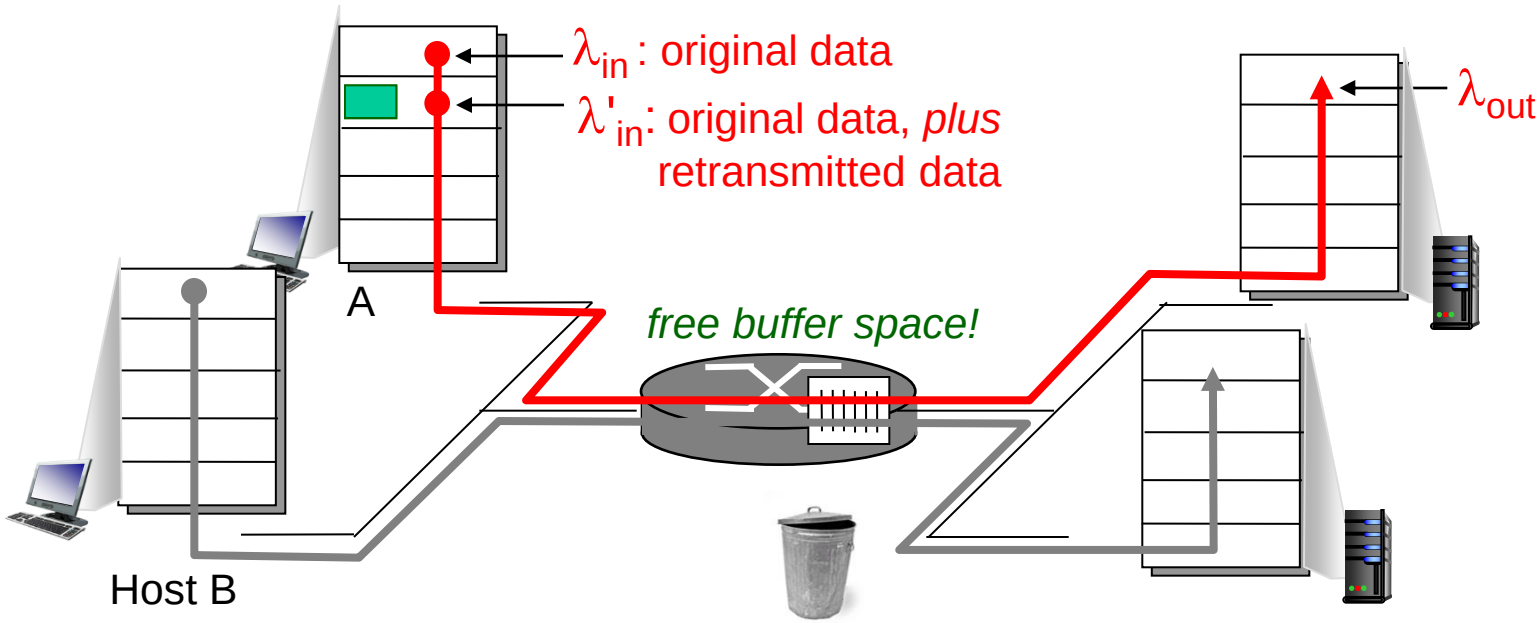
Αιτίες/κόστος Συμφόρησης: σενάριο 2

Εξιδανίκευση: γνωστές απώλειες.

πακέτα μπορεί να χαθούν, να απορριφθούν στο router λογω πληρότητας των full buffers

- Ο sender ξαναστέλνει πακέτα μόνο αν ξέρει ότι έχουν χαθεί.



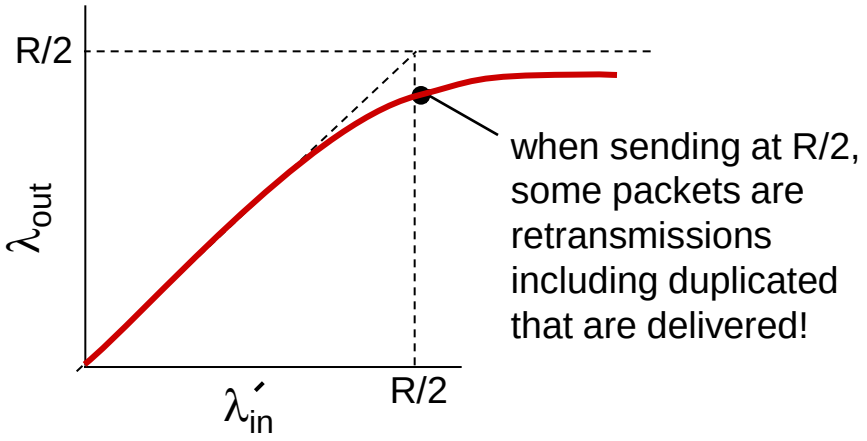


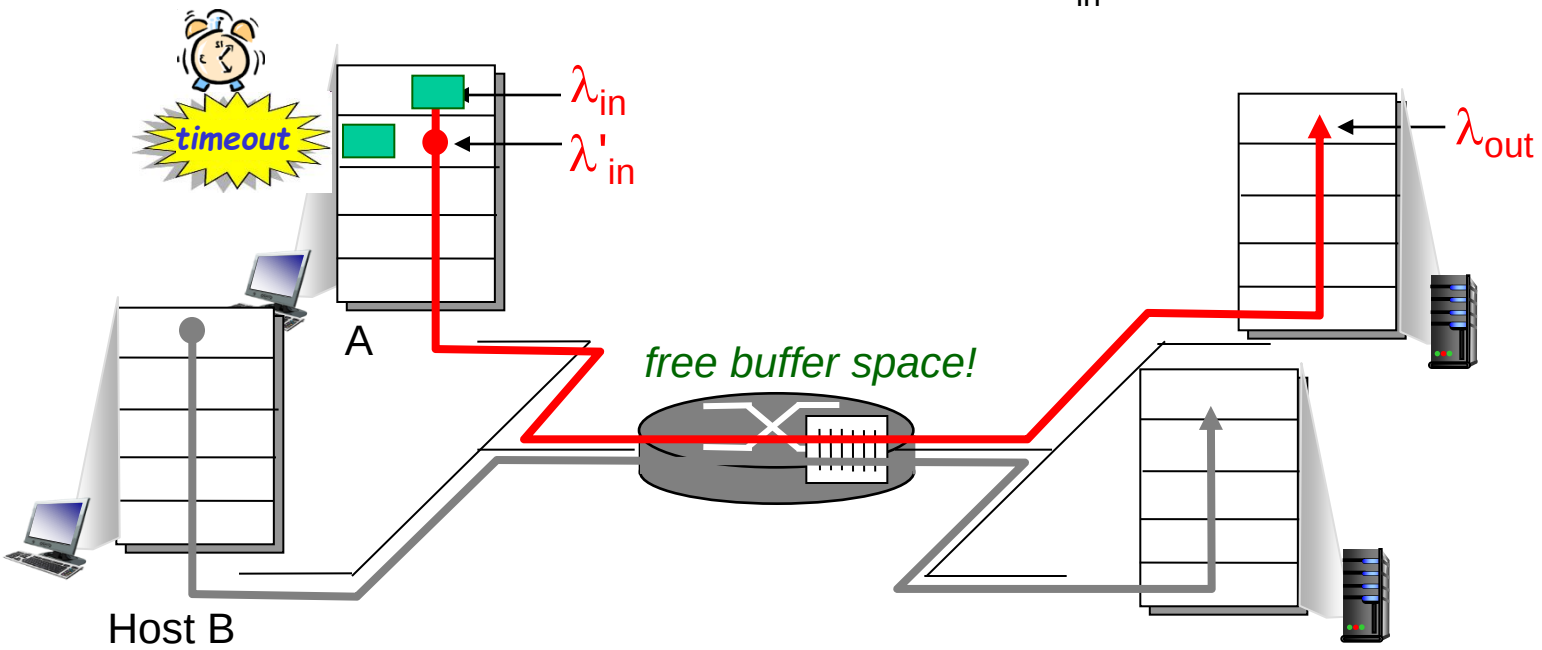
Αιτίες/κόστος Συμφόρησης: σενάριο 2

Ρεαλιστικά: duplicates

Πακέτα μπορεί να χαθούν, να απορριφθούν στο router λογω πληρότητας των full buffers

- Πρώιμο time-out στον sender, και αποστολή δύο αντιγράφων τα οποία και παραδίδονται



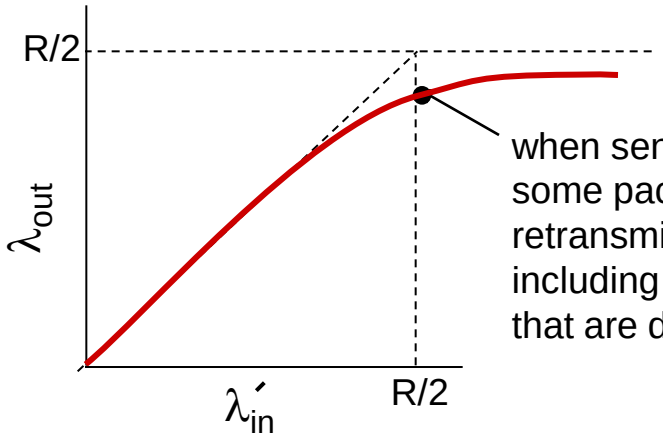


Αιτίες/κόστος Συμφόρησης: σενάριο 2

Ρεαλιστικά: duplicates

Πακέτα μπορεί να χαθούν, να απορριφθούν στο router λογω πληρότητας των full buffers

- Πρώιμο time-out στον sender, και αποστολή δύο αντιγράφων τα οποία και παραδίδονται



"κόστος" συμφόρησης:

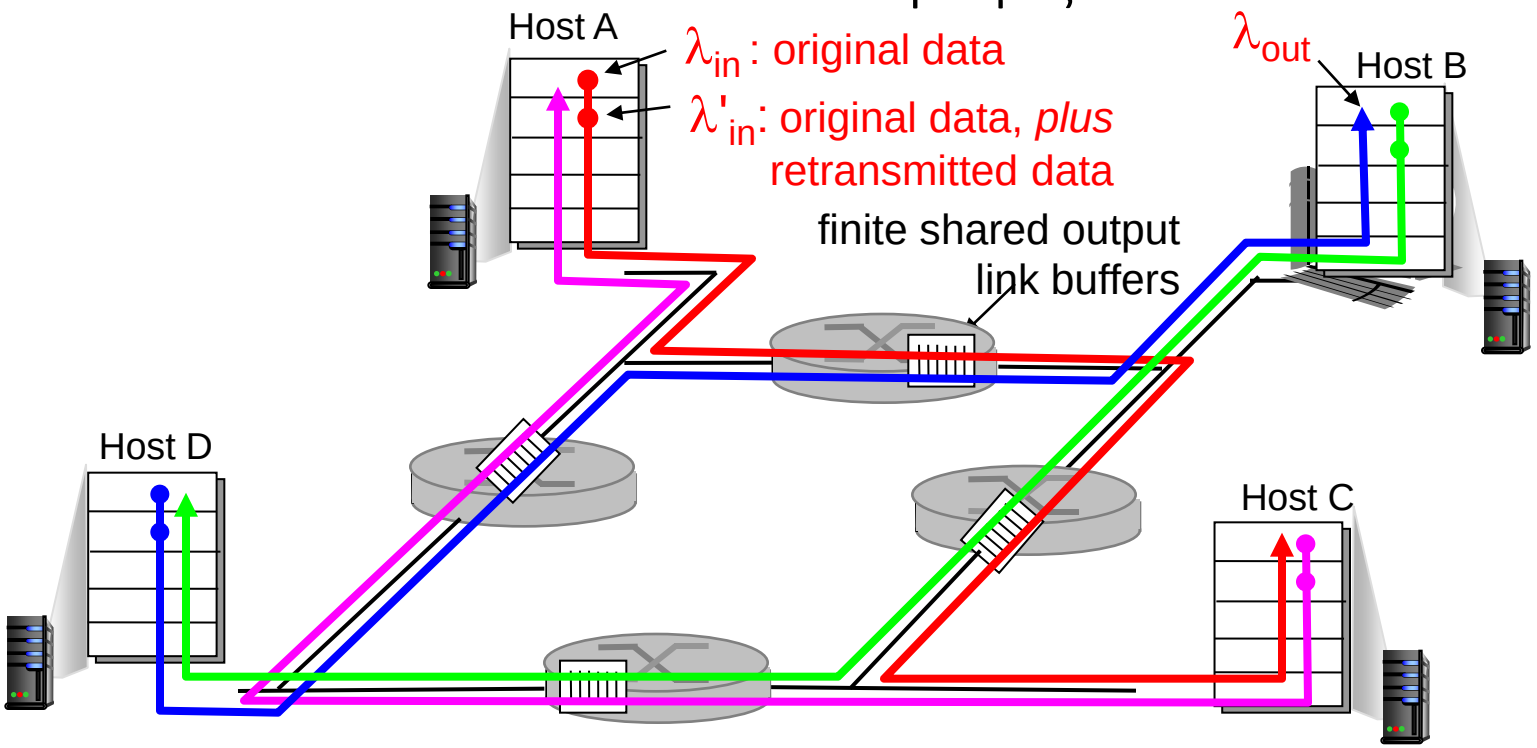
- Περισσότερος φόρτος (retransmissions) για δεδομένο ρυθμό
- Μη απαραίτητες επαναλήψεις: Η γραμμή μεταφέρει πολλαπλά αντίγραφα ενός πακέτου
  - Μειωμένος ρυθμός μετάδοσης.

Αιτίες/κόστος Συμφόρησης: σενάριο 3

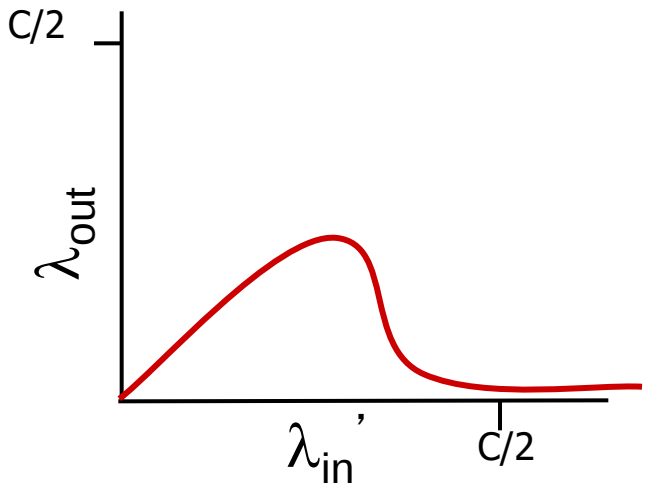
- τέσσερεις senders
- Διαδρομές με πολλαπλούς routers
- timeout/retransmit

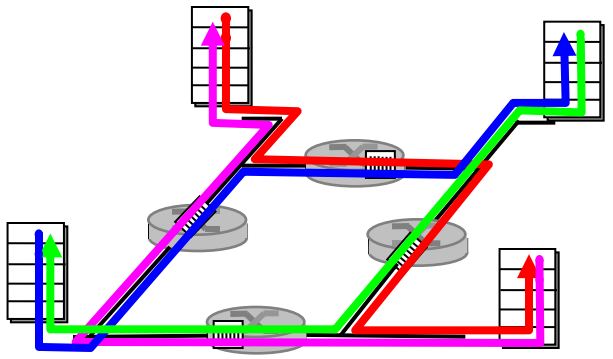
E: Τι συμβαίνει καθώς λ_{in} and λ_{in}' αυξάνουν ?

A: Καθώς αυξάνει η κοκκινη λ_{in}', όλα τα μπλέ πακέτα που φτάνουν στην επάνω γραμμή απορρίπτονται, και ο μπλε ρυθμός τείνει στο 0.



Αιτίες/κόστος Συμφόρησης: σενάριο 3





Άλλο ένα κόστος συμφόρησης

- Όταν ένα πακέτο απορρίπτεται, η χωρητικότητα της γραμμής που χρησιμοποιήθηκε μέχρι την απόρριψη πάει χαμένη!

Προσεγγίσεις για έλεγχο συμφόρησης

Δυο γενικές προσεγγίσεις:

end-end congestion control:

- Δεν υπάρχει ανατροφοδότηση από το δίκτυο.
- Συμφόρηση συμπεραίνεται στα άκρα από απώλεια πακέτων και καθυστέρηση.
- Εφαρμόζεται στο TCP

network-assisted congestion control:

- Οι routers παρέχουν ανατροφοδότηση στα τελικά συστήματα.
  - Ένα bit δείχνει συμφόρηση (SNA, DECbit, TCP/IP ECN, ATM)
  - Αποστέλλεται στον sender ο ρυθμός στον οποίο πρέπει να στέλνει.

Case study: ATM ABR congestion control

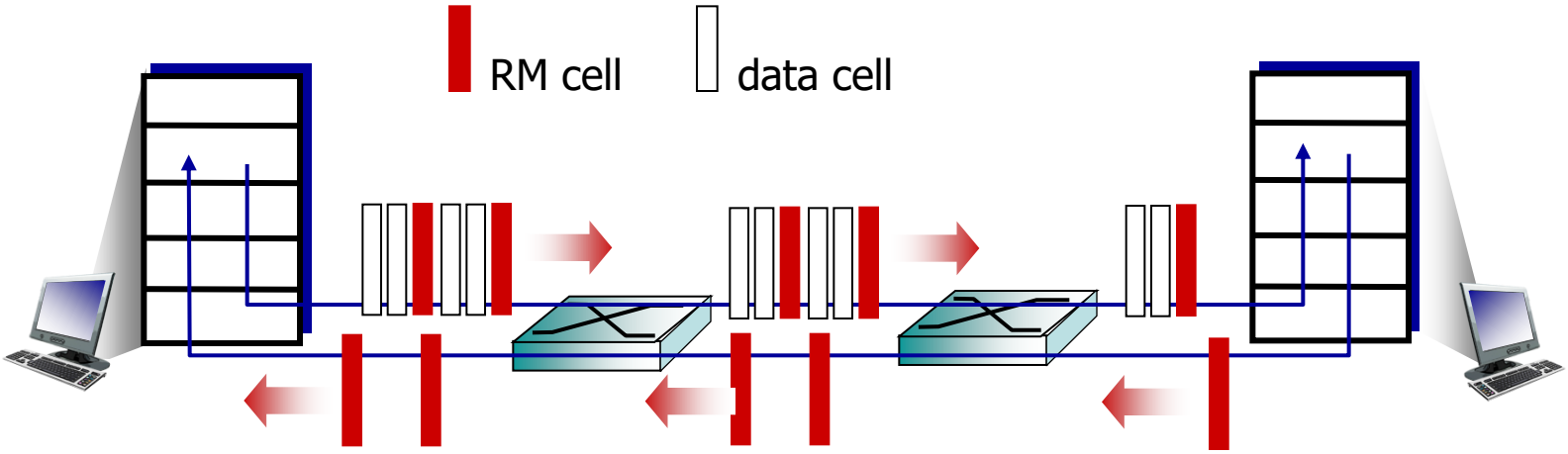
ABR: available bit rate:

- "elastic service"
- if sender's path "underloaded":
  - sender should use available bandwidth
- if sender's path congested:
  - sender throttled to minimum guaranteed rate

RM (resource management) cells:

- sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted*")
  - *NI bit:* no increase in rate (mild congestion)
  - *CI bit:* congestion indication
- RM cells returned to sender by receiver, with bits intact

Case study: ATM ABR congestion control



- two-byte ER (explicit rate) field in RM cell
  - congested switch may lower ER value in cell
  - senders' send rate thus max supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - if data cell preceding RM cell has EFCI set, receiver sets CI bit in returned RM cell

Κεφάλαιο 3: Περίγραμμα

3.1 υπηρεσίες επιπέδου μεταφοράς

3.2 πολύπλεξη και αποπολύπλεξη

3.3 μη συνδεσμοστραφής μεταφορά: UDP

3.4 αρχές αξιόπιστης μεταφοράς δεδομένων

3.5 συνδεσμοστραφής μεταφορά (connection-oriented reliable transport): TCP

- Δομή του segment
- Αξιόπιστη μεταφορά δεδομένων
- Έλεγχος ροής
- Διαχείριση συνδέσμου

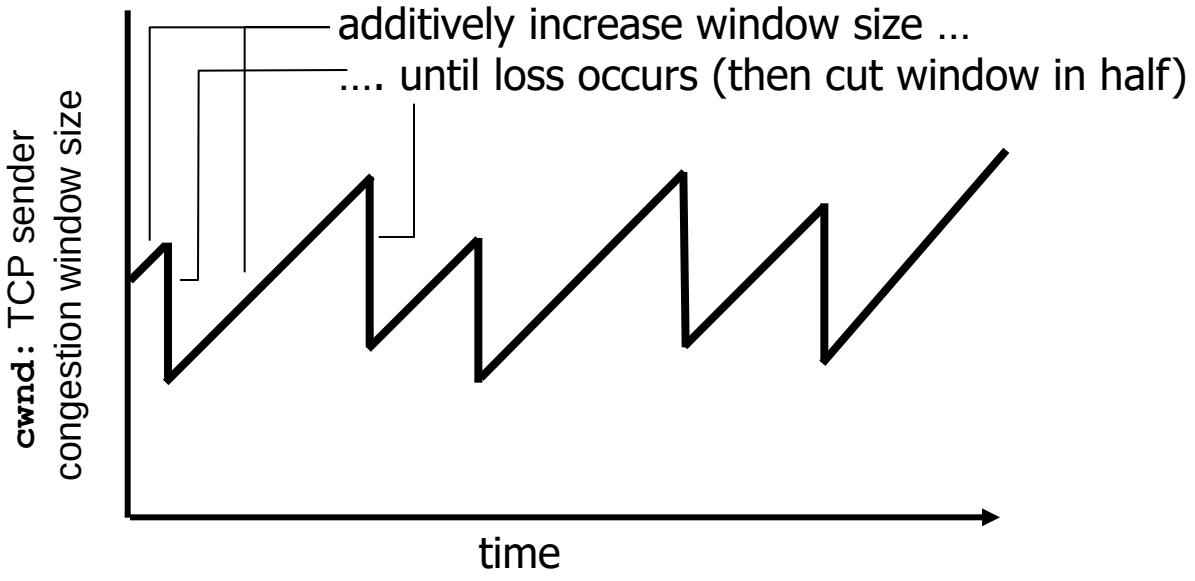
3.6 Αρχές έλεγχου συμφόρησης

3.7 TCP έλεγχος συμφόρησης

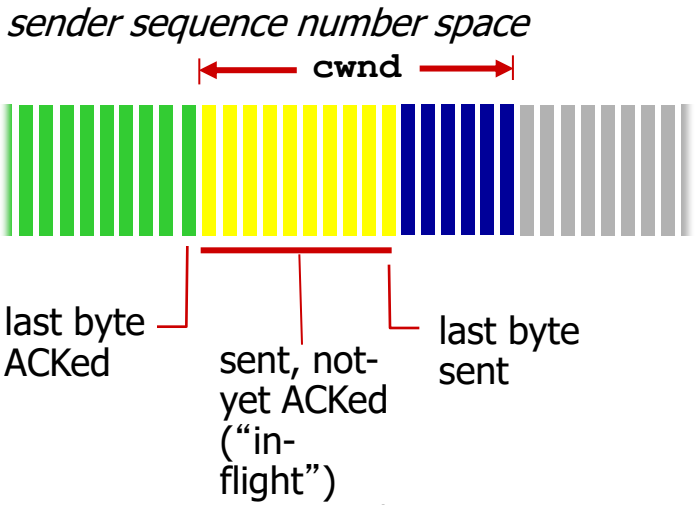
TCP έλεγχος συμφόρησης: προσθετική αύξηση πολλαπλασιαστική μείωση

- *Προσέγγιση:* Ο sender *αυξάνει το ρυθμό μετάδοσης* (window size), δοκιμάζοντας την γραμμή, μέχρι να συμβεί απώλεια
  - *additive increase:* αύξηση του **cwnd** κατά 1 MSS κάθε RTT μέχρι να ανιχνευθεί απώλεια
  - *multiplicative decrease*: μείωση του **cwnd** στο μισό μετά από απώλεια.

Πριονωτή συμπεριφορά AIMD: Δοκιμάζοντας το εύρος της γραμμής



TCP Έλεγχος συμφόρησης: λεπτομέρειες



- Ο sender περιορίζει την εκπομπή:

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** είναι δυναμικό, συνάρτηση της εκτιμώμενης συμφόρησης του δικτύου

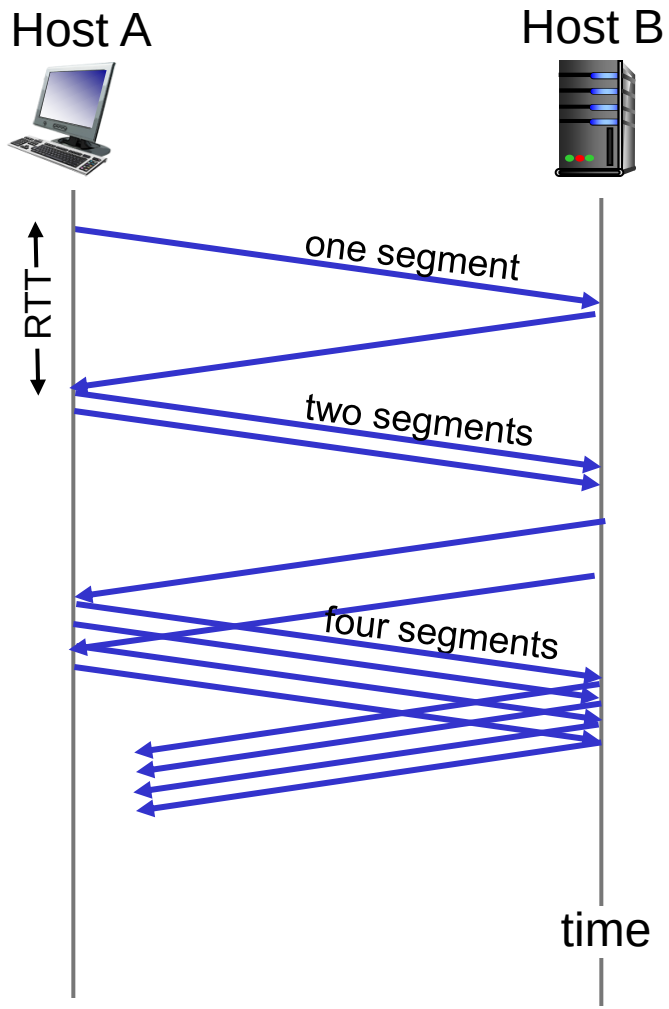
Ρυθμός αποστολής TCP :

- *χοντρικά:* στέλνω cwnd bytes, περιμένω RTT για ACKS, μετά στέλνω και άλλα bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

TCP Αργή Αρχή

- Όταν αρχίζει μια σύνδεση, αυξάνει ο ρυθμός εκθετικά μέχρι την πρώτη απώλεια :
  - αρχικά **cwnd** = 1 MSS
  - διπλασιασμός **cwnd** κάθε RTT
  - Αύξηση του **cwnd** για κάθε ACK που λαμβάνεται
- *Συνοπτικά:* Ο αρχικός ρυθμός είναι αργός αλλά αυξάνει εκθετικά



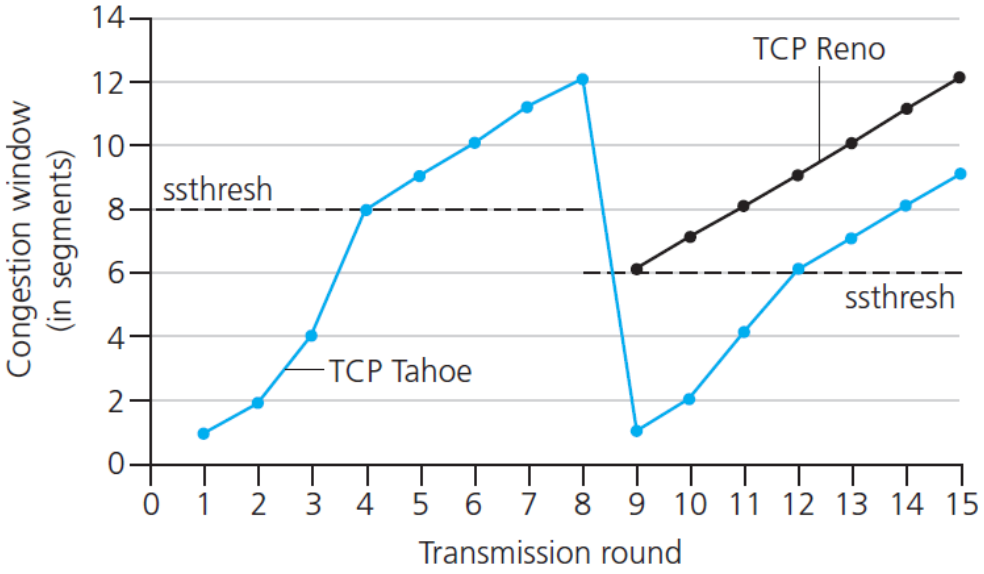
TCP: Ανίχνευση, αντίδραση σε απώλεια

- Η απώλεια υποδεικνύεται από timeout:
  - **cwnd** ορίζεται σε 1 MSS;
  - Το παράθυρο αυξάνεται εκθετικά (όπως στην αργή αρχή) μέχρι ένα κατώφλι και μετα αυξάνει γραμμικά.
- Η απώλεια υποδεικνύεται από 3 διπλότυπα ACKs:
  - διπλότυπα ACKs δείχνουν ότι το δίκτυο μπορεί να παραδίδει κάποια segments
  - **cwnd** μειώνεται στο μισο και μετά αυξάνει γραμμικά (TCP RENO)
- TCP Tahoe πάντα αρχίζει με **cwnd** 1 (είτε timeout είτε 3 διπλότυπα acks)

TCP: αλλαγή από slow start γραμμική αύξηση

E: πότε πρέπει η εκθετική αυξηση να αλλάξει σε γραμμική;

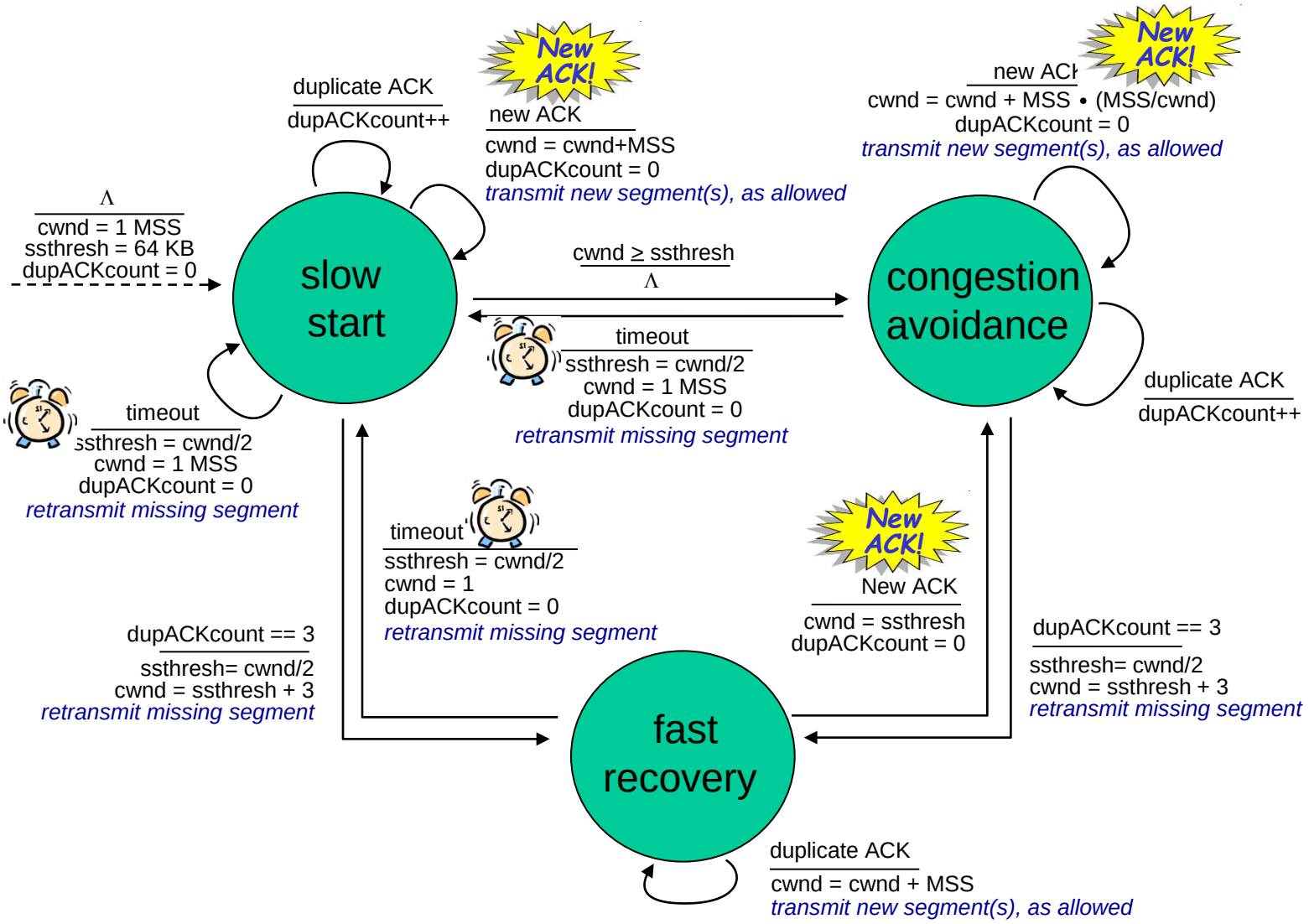
A: Όταν το **cwnd** φθάσει στο 1/2 της τιμής του πριν το timeout.



Υλοποίηση:

- παράμετρος **ssthres**
- Σε απώλεια, το **ssthresh** οριζεται στο 1/2 του **cwnd** αμέσως πριν την απώλεια

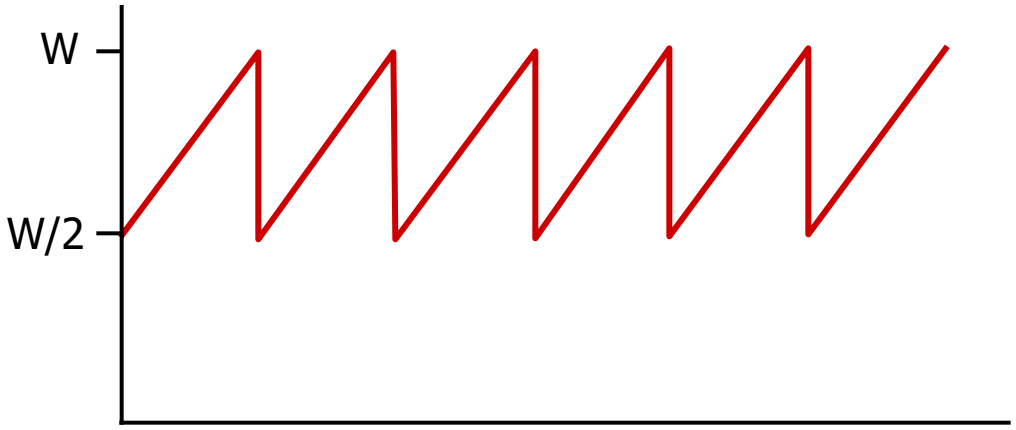
Περίληψη: TCP Έλεγχος Συμφόρησης



TCP Ταχύτητα μεταγωγής (throughput)

- Μέση Ταχύτητα μεταγωγής TCP ως συνάρτηση του μεγέθους του παράθυρου και του RTT?
  - Αγνοούμε slow start, υποθέτουμε ότι πάντα υπάρχουν δεδομένα για αποστολή
- W: μέγεθος παράθυρου (σε bytes) στο οποίο συμβαίνουν απώλειες.
  - μέσο μέγεθος παράθυρου (# in-flight bytes) είναι ¾ W
  - Μέση ταχύτητα είναι 3/4W ανά RTT

$$\text{Μέση ταχύτητα μεταγωγής TCP} = \frac{3}{4}\ \frac{W}{RTT} \quad \text{bytes/sec}$$



TCP στο Μέλλον: TCP σε "γραμμές μεγάλου ευρους και μήκους"

- παράδειγμα: 1500 byte segments, 100ms RTT, ζητούμενη 10 Gbps throughput
- απαιτεί παράθυρο W = 83.333 segments
- Ταχύτητα σε σχέση με πιθανότητα απώλειας, L [Mathis 1997]:

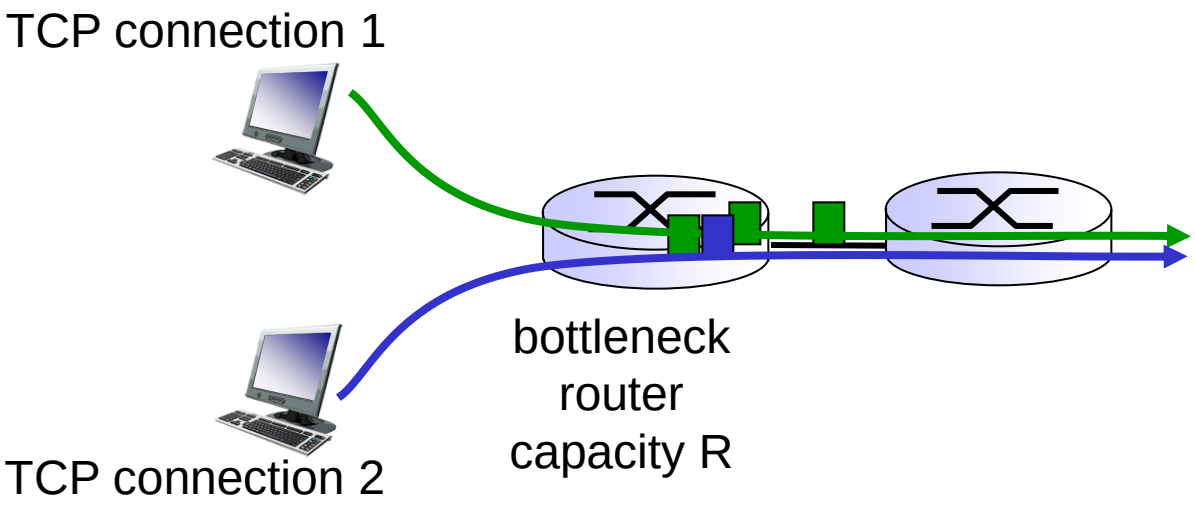
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ Για να επιτευχθεί ταχύτητα 10 Gbps, η πιθανότητες απώλειας πρέπει να είναι $L = 2 \cdot 10^{-10}$ *– πολύ μικρή!*

- Νέες εκδόσεις του TCP για υψηλές ταχύτητες

TCP Fairness

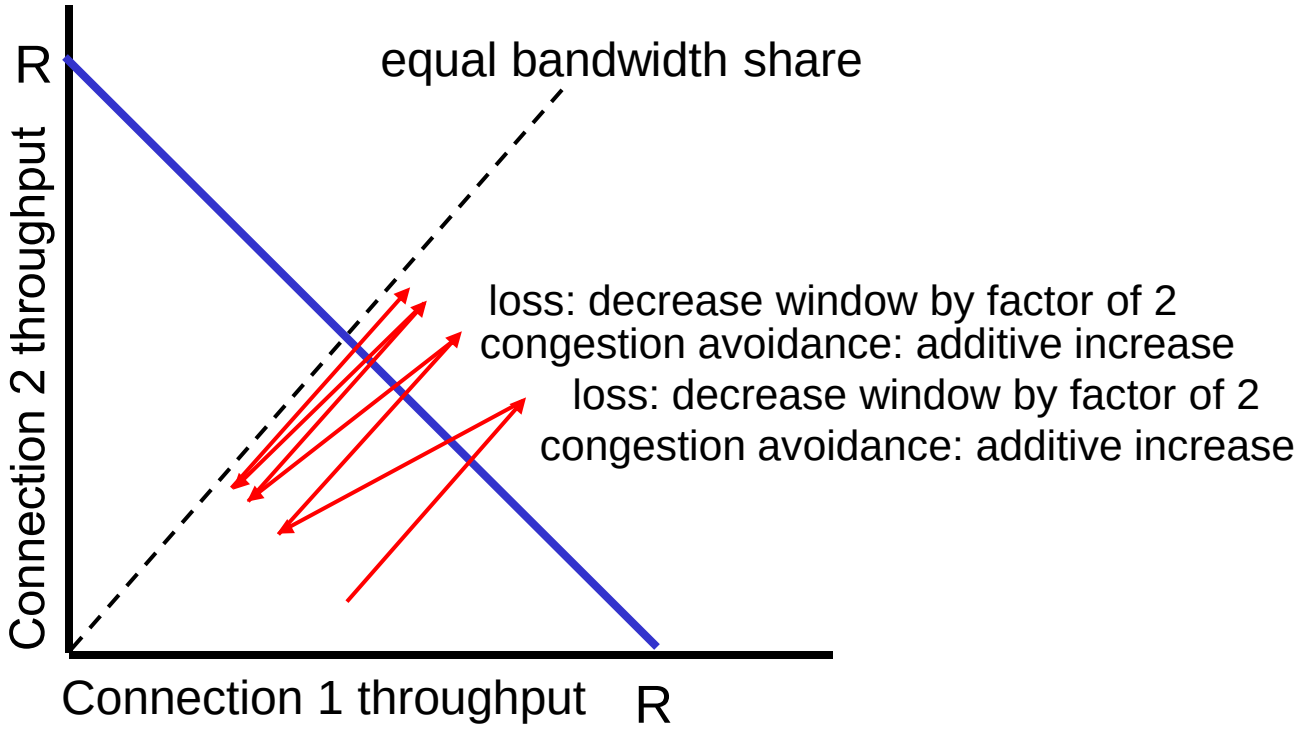
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

two competing sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Fairness (more)

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"