Επίπεδο Μεταφοράς

Στόχοι:

- κατανόηση αρχών που διέπουν τις υπηρεσίες που παρέχονται από το επίπεδο μεταφοράς:
  - πολύπλεξη/αποπολύπλεξη multiplexing/demultiplexing
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - έλεγχος συμφόρησης
- εξέταση πρωτοκόλλων επιπέδου μεταφοράς του Διαδικτύου:
  - UDP: ασυνδεσιστρεφής μεταφορά
  - TCP: συνδεσιστρεφής μεταφορά
  - έλεγχος συμφόρησης στο TCP

Περίγραμμα

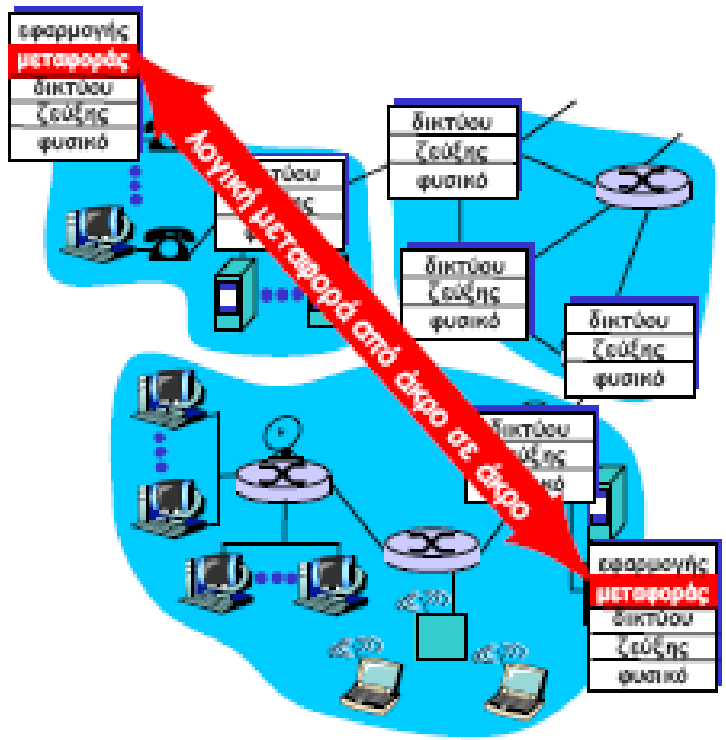
- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Περίγραμμα

- **Υπηρεσίες επιπέδου μεταφοράς**
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Υπηρεσίες και πρωτόκολλα μεταφοράς

- παρέχουν *λογική επικοινωνία* μεταξύ των διεργασιών μίας εφαρμογής που τρέχουν σε διαφορετικούς hosts
- τα πρωτόκολλα μεταφοράς τρέχουν στα τερματικά συστήματα
  - πλευρά αποστολέα: μετατροπή μηνυμάτων εφαρμογής σε segments που προωθούνται στο επίπεδο δικτύου
  - πλευρά παραλήπτη: συναρμολόγηση των segments σε μηνύματα που προωθούνται στο επίπεδο εφαρμογής
- διάφορα πρωτόκολλα μεταφοράς διαθέσιμα στις εφαρμογές
  - Διαδίκτυο: TCP και UDP



Σχέση μεταξύ επιπέδων μεταφοράς & δικτύου

- *επίπεδο μεταφοράς:* λογική επικοινωνία μεταξύ διεργασιών
- *επίπεδο δικτύου:* λογική επικοινωνία μεταξύ hosts
- το επίπεδο μεταφοράς επεκτείνει την υπηρεσία παράδοσης από host σε host του επιπέδου δικτύου σε μία υπηρεσία παράδοσης από διεργασία σε διεργασία (πολύπλεξη/αποπολύπλεξη)
- το επίπεδο μεταφοράς στηρίζεται στις υπηρεσίες του επιπέδου δικτύου και τις επαυξάνει

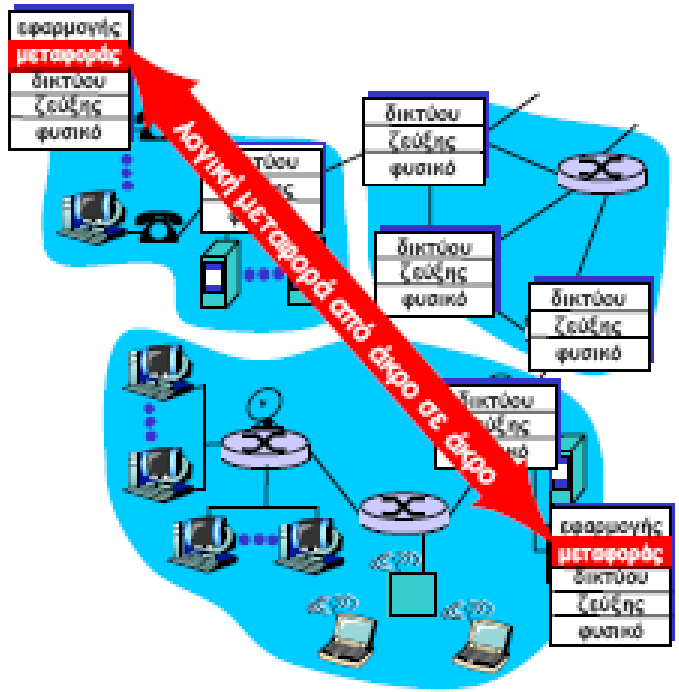
Αναλογία

12 παιδιά στέλνουν γράμματα σε 12 παιδιά

- διεργασίες = παιδιά
- μηνύματα εφαρμογής = γράμματα σε φακέλους
- hosts = σπίτια
- πρωτόκολλο μεταφοράς = οι συλλέκτες/διανομείς γραμμάτων στο σπίτι
- πρωτόκολλο επιπ. δικτύου = ταχυδρομική υπηρεσία

Πρωτόκολλα επιπέδου μεταφοράς στο Διαδίκτυο

- UDP: παρεχόμενες υπηρεσίες
  - πολύπλεξη/αποπολύπλεξη
  - έλεγχος ακεραιότητας
- UDP: αναξιόπιστη υπηρεσία
  - επέκταση της αναξιόπιστης ("best-effort") υπηρεσίας παράδοσης του IP
- TCP: πρόσθετες παρεχόμενες υπηρεσίες
  - αξιόπιστη παράδοση
  - έλεγχος ροής
  - έλεγχος συμφόρησης
- μη παρεχόμενες υπηρεσίες: εγγυήσεις ως προς
  - καθυστέρηση
  - bandwidth



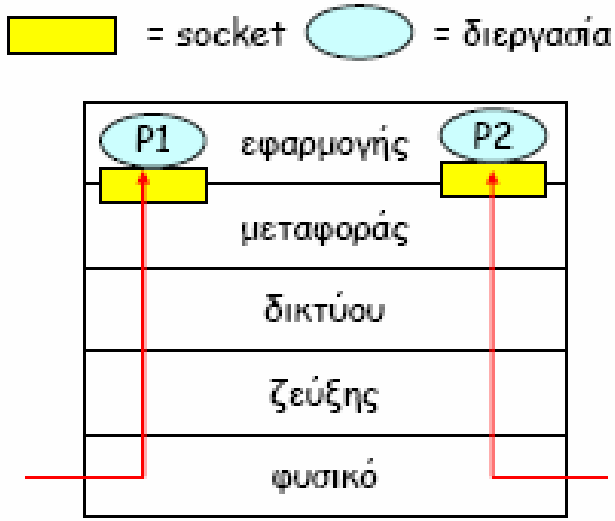
Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- **Πολύπλεξη και αποπολύπλεξη**
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Αποπολύπλεξη

host προορισμού:

- το επίπεδο μεταφοράς λαμβάνει segments από το επίπεδο δικτύου
- το επίπεδο μεταφοράς πρέπει να παραδώσει τα segments στις σωστές διεργασίες, ή καλύτερα, στα σωστά sockets
- πολλαπλά sockets στον host → απαιτείται socket identifier
- κάθε segment φέρει στην επικεφαλίδα πεδία που προσδιορίζουν το socket για το οποίο προορίζονται τα δεδομένα:
  - αριθμός θύρας πηγής (source port number)
  - αριθμός θύρας προορισμού (destination port number)



- αριθμός θύρας (port number): αριθμός από 16 bits (0 – 65535)
- καλά γνωστοί αριθμοί θύρας (well-known port numbers): 0 - 1023
  - π.χ. θύρα HTTP server: 80

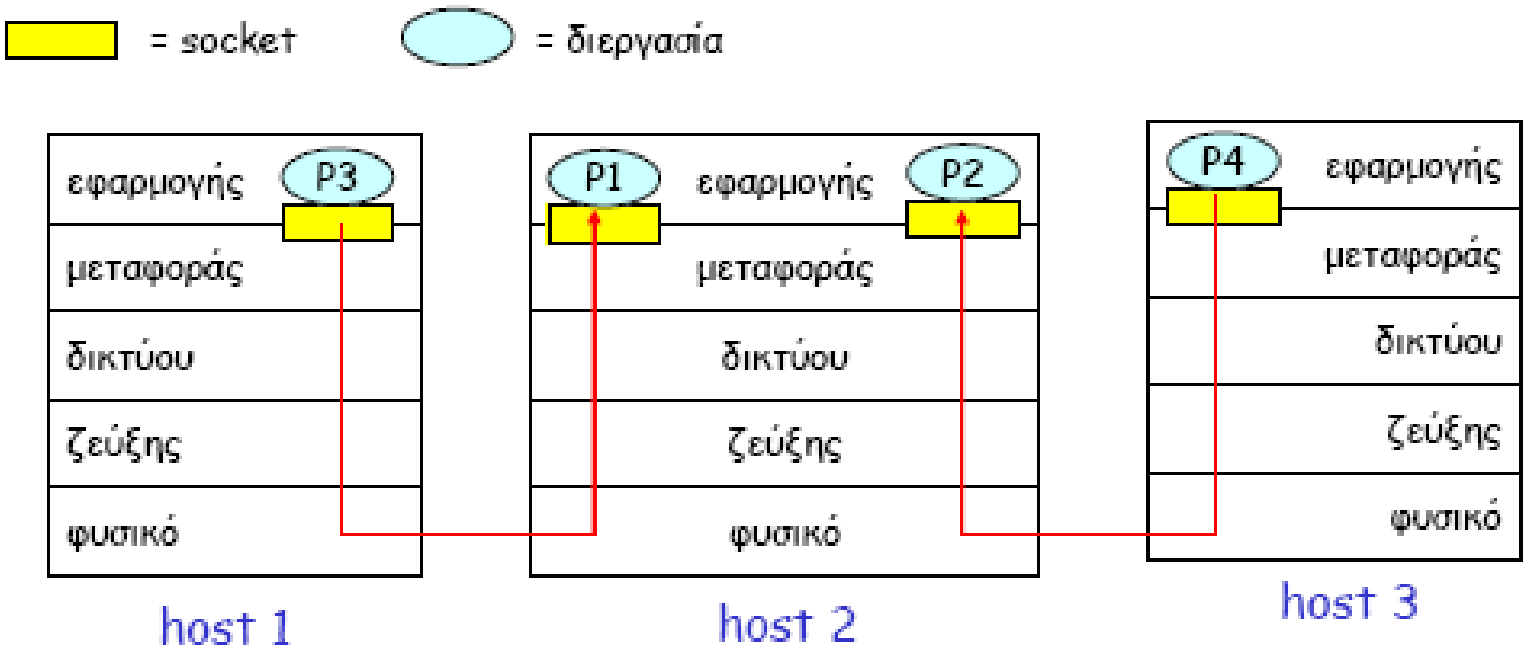
Πολύπλεξη/Αποπολύπλεξη

Αποπολύπλεξη στον παραλήπτη host:

παράδοση λαμβανομένων segments στο σωστό socket

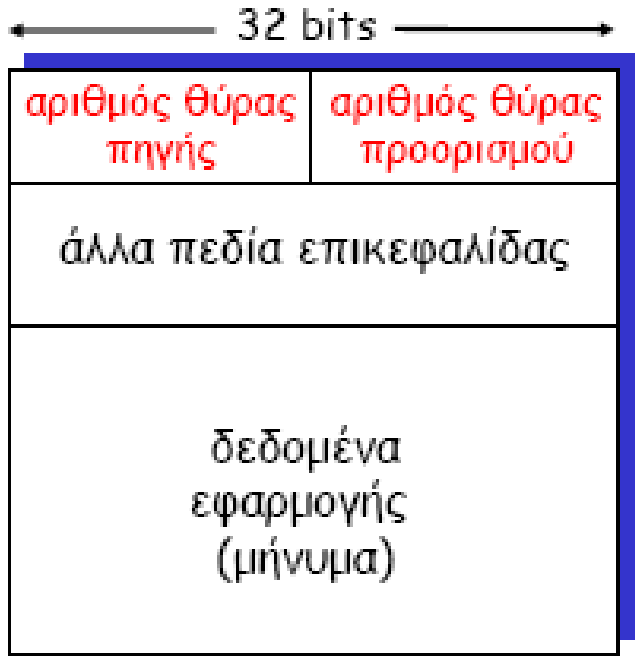
Πολύπλεξη στον αποστολέα host:

συγκέντρωση δεδομένων από πολλαπλά sockets, ενθυλάκωση δεδομένων σε segments, παράδοση segments στο επίπεδο δικτύου



Αποπολύπλεξη (συνέχεια)

- Ο host λαμβάνει IP datagrams
  - κάθε datagram έχει διεύθυνση IP πηγής, διεύθυνση IP προορισμού
  - κάθε datagram μεταφέρει ένα segment
  - κάθε segment έχει αριθμό θύρας πηγής, αριθμό θύρας προορισμού
- Ο host χρησιμοποιεί διευθύνσεις IP & αριθμούς θύρας για να κατευθύνει το segment στο κατάλληλο socket
- UDP socket identifier ≠ TCP socket identifier



δομή TCP/UDP segment

Αποπολύπλεξη στο UDP (ασυνδεσιστρεφής)

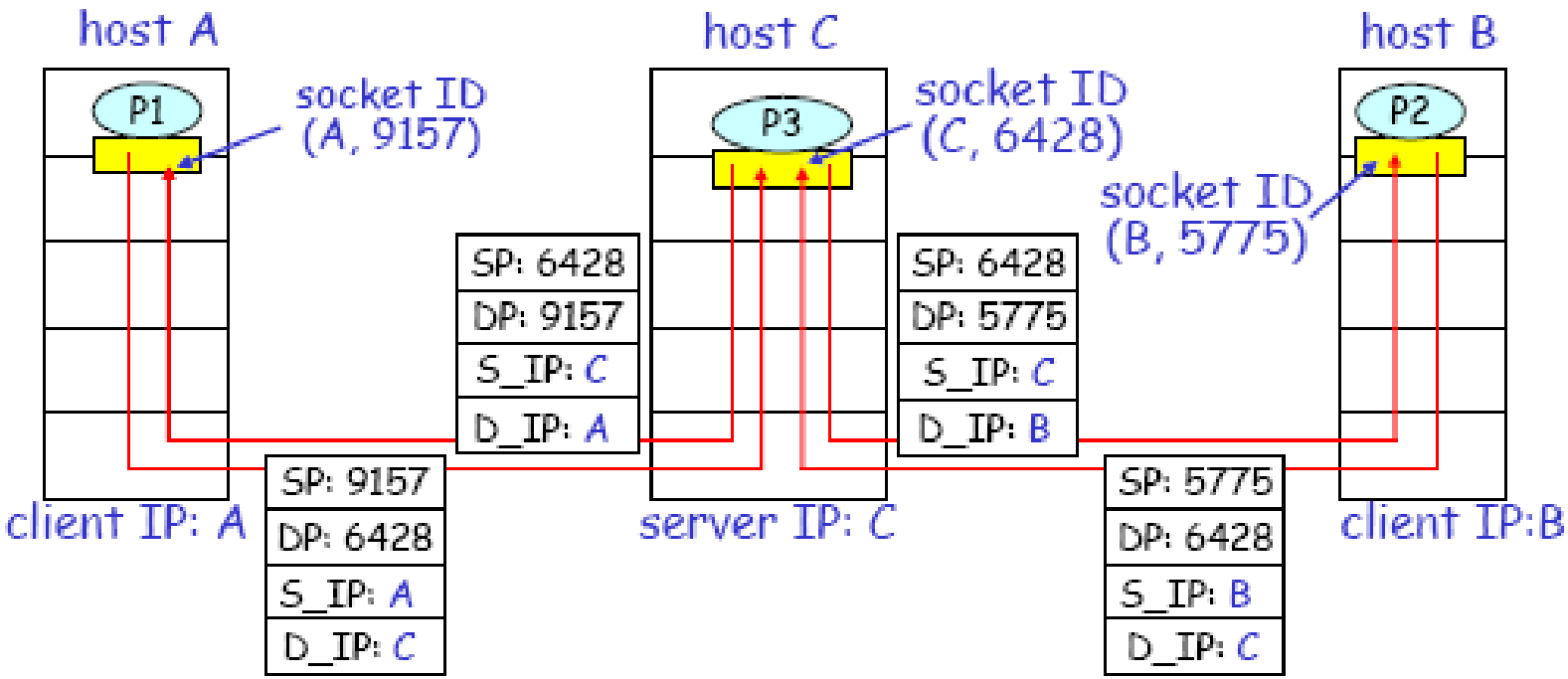
- ένα UDP socket προσδιορίζεται από το ζεύγος:

  (διεύθυνση IP προορισμού,
  αριθμός θύρας προορισμού)

- Όταν ο host λάβει ένα UDP segment:
  - ελέγχει τον αριθμό θύρας προορισμού στο segment
  - κατευθύνει το UDP segment στο socket με αυτόν τον αριθμό θύρας

- IP datagrams με διαφορετικές διευθύνσεις IP πηγής και/ή διαφορετικούς αριθμούς θύρας πηγής οδηγούνται στο ίδιο socket

Αποπολύπλεξη στο UDP (συνέχεια)

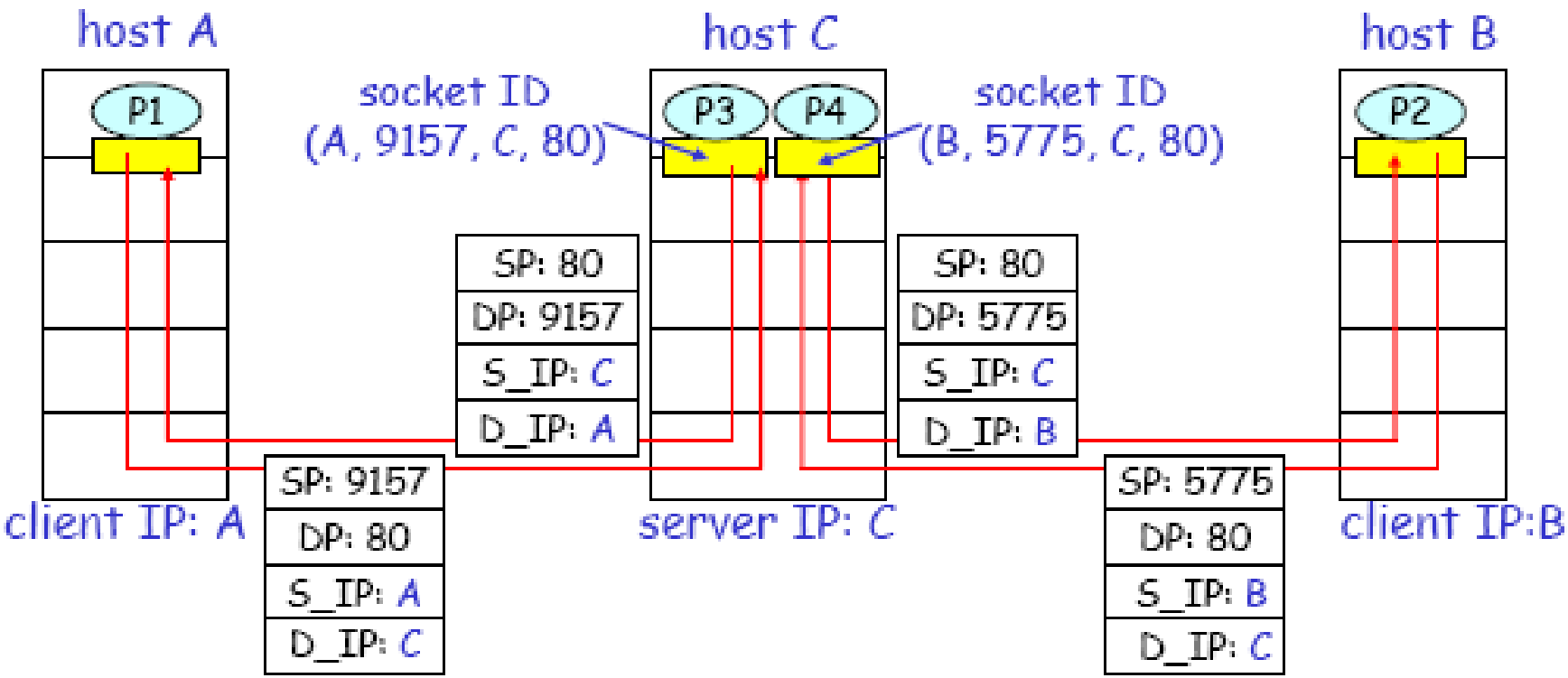


SP παρέχει "διεύθυνση επιστροφής"

Αποπολύπλεξη στο TCP (συνδεσιστρεφής)

- ένα TCP socket προσδιορίζεται από την τετράδα:
  - διεύθυνση IP πηγής
  - αριθμός θύρας πηγής
  - διεύθυνση IP προορισμού
  - αριθμός θύρας προορισμού
- Ο host που λαμβάνει ένα segment (διεύθυνση IP προορισμού) χρησιμοποιεί και τις τρεις υπόλοιπες τιμές για να κατευθύνει το TCP segment στο κατάλληλο socket
- Ένας server host μπορεί να υποστηρίξει πολλά TCP sockets ταυτόχρονα:
  - κάθε socket σε ένα host προσδιορίζεται από τη δική του τριάδα
- Οι web servers έχουν διαφορετικά sockets για κάθε συνδεδεμένο client
  - το non-persistent HTTP έχει διαφορετικό socket για κάθε αίτηση HTTP

Αποπολύπλεξη στο TCP (συνέχεια)



Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- **Ασυνδεσιστρεφής μεταφορά: UDP**
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

UDP: User Datagram Protocol [RFC 768]

- "μινιμαλιστικό" πρωτόκολλο μεταφοράς
  - πολύπλεξη/αποπολύπλεξη
  - έλεγχος σφαλμάτων
- υπηρεσία "βέλτιστης προσπάθειας (best effort)"

  Τα UDP segments ενδέχεται:
  - να χαθούν
  - να παραδοθούν εκτός σειράς
- *connectionless:*
  - δεν γίνεται χειραψία (handshaking) μεταξύ UDP αποστολέα, παραλήπτη
  - το επίπεδο μεταφοράς χειρίζεται κάθε UDP segment ανεξάρτητα από άλλα

Τι εξυπηρετεί το UDP;

- δεν απαιτεί εγκαθίδρυση σύνδεσης (που εισάγει καθυστέρηση)
- απλό: δεν διατηρεί κατάσταση σύνδεσης
  - receive & send buffers
  - παράμετροι ελέγχου συμφόρησης
  - sequence, acknowledgement #

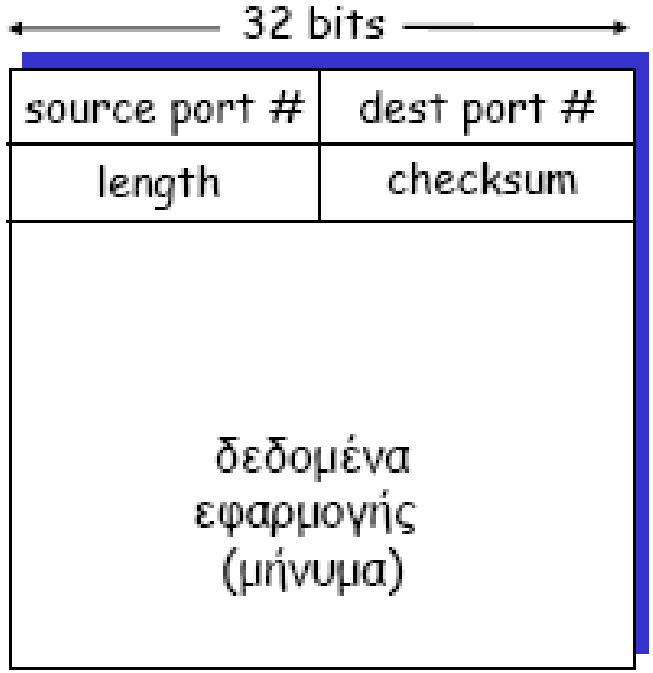
  σε αποστολέα και παραλήπτη
- μικρή επικεφαλίδα segment
  - 8 bytes (UDP) vs. 20 bytes (TCP)
- δεν παρέχει έλεγχο συμφόρησης: το UDP μπορεί να στείλει δεδομένα όσο γρήγορα μπορεί (εν δυνάμει πρόβλημα)
- δεν έχει καθυστερήσεις λόγω επαναμεταδόσεων

UDP

Εφαρμογές που χρησιμοποιούν UDP

- πολυμεσικές εφαρμογές συνεχούς ροής (streaming multimedia)
  - με ανοχή στις απώλειες
  - ευαίσθητες ως προς το bandwidth
- άλλες εφαρμογές π.χ.
  - DNS
  - SNMP
- αξιόπιστη μεταφορά με UDP: με μηχανισμούς ανάκαμψης από σφάλματα στο επίπεδο εφαρμογής
  - ο μηχανισμός εξαρτάται από τις απαιτήσεις της εφαρμογής

Δομή UDP segment



length: μήκος UDP segment σε bytes (συμπεριλαμβανομένης επικεφαλίδας)

UDP checksum

Στόχος: ανίχνευση σφαλμάτων (ανεστραμμένων bits) στο λαμβανόμενο segment

Αποστολέας:

- χειρίζεται το περιεχόμενο του segment ως ακολουθία ακεραίων των 16 bits
- checksum: συμπλήρωμα ως προς 1 του αθροίσματος των περιεχομένων του segment
- ο αποστολέας τοποθετεί την τιμή του checksum στο πεδίο checksum του UDP segment

Παραλήπτης:

- υπολογίζει το checksum του λαμβανόμενου segment
- ελέγχει εάν η υπολογισθείσα τιμή του checksum είναι ίση με την τιμή στο πεδίο checksum:
  - Όχι - ανίχνευση σφάλματος
    - απόρριψη segment
    - προώθηση με προειδοποίηση
  - Ναι - δεν έχει ανιχνευθεί κανένα σφάλμα

UDP checksum

Έστω segment:

0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

Αποστολέας:

πρόσθεση

0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0

\+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

1 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0

CKS 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1

Παραλήπτης:

πρόσθεση

0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

\+ 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

σφάλμα 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1

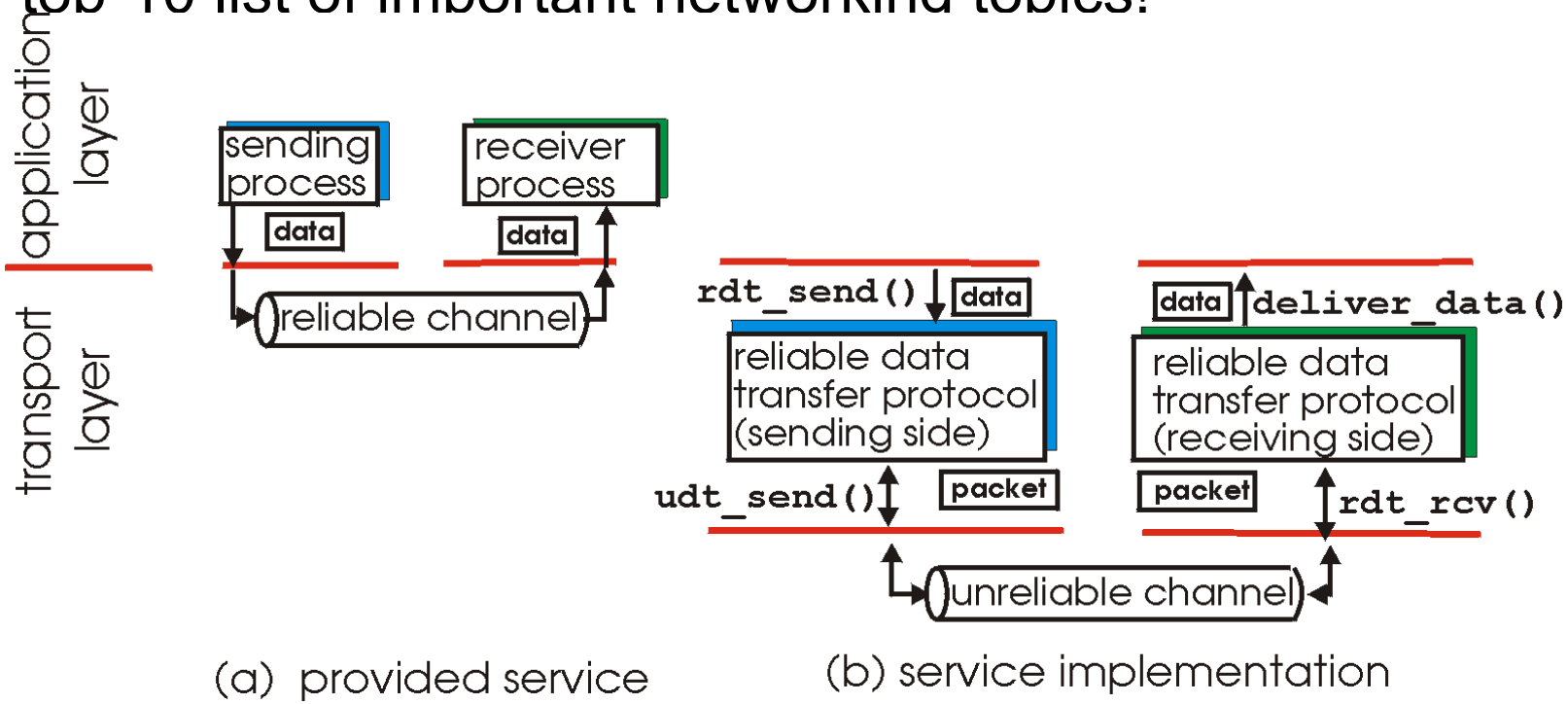
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



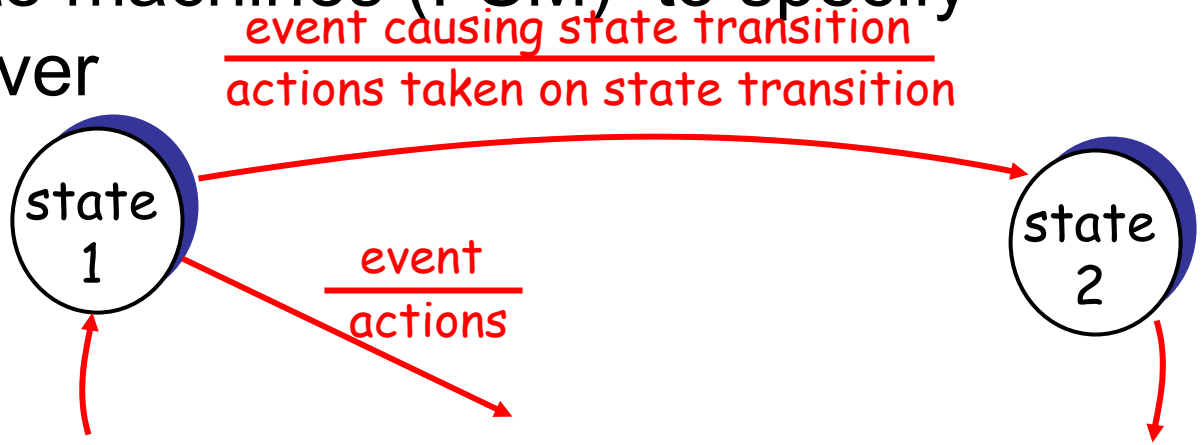
- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

We'll:

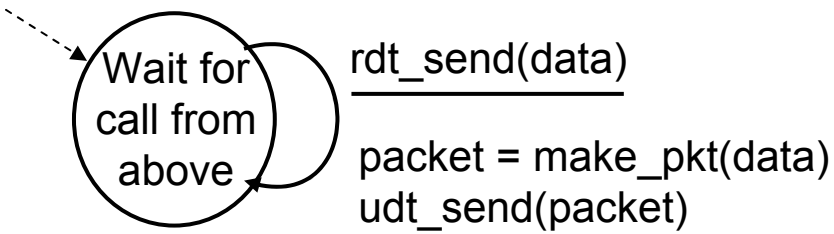
- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event

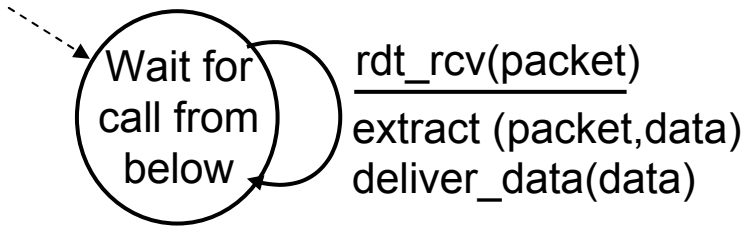


Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



sender

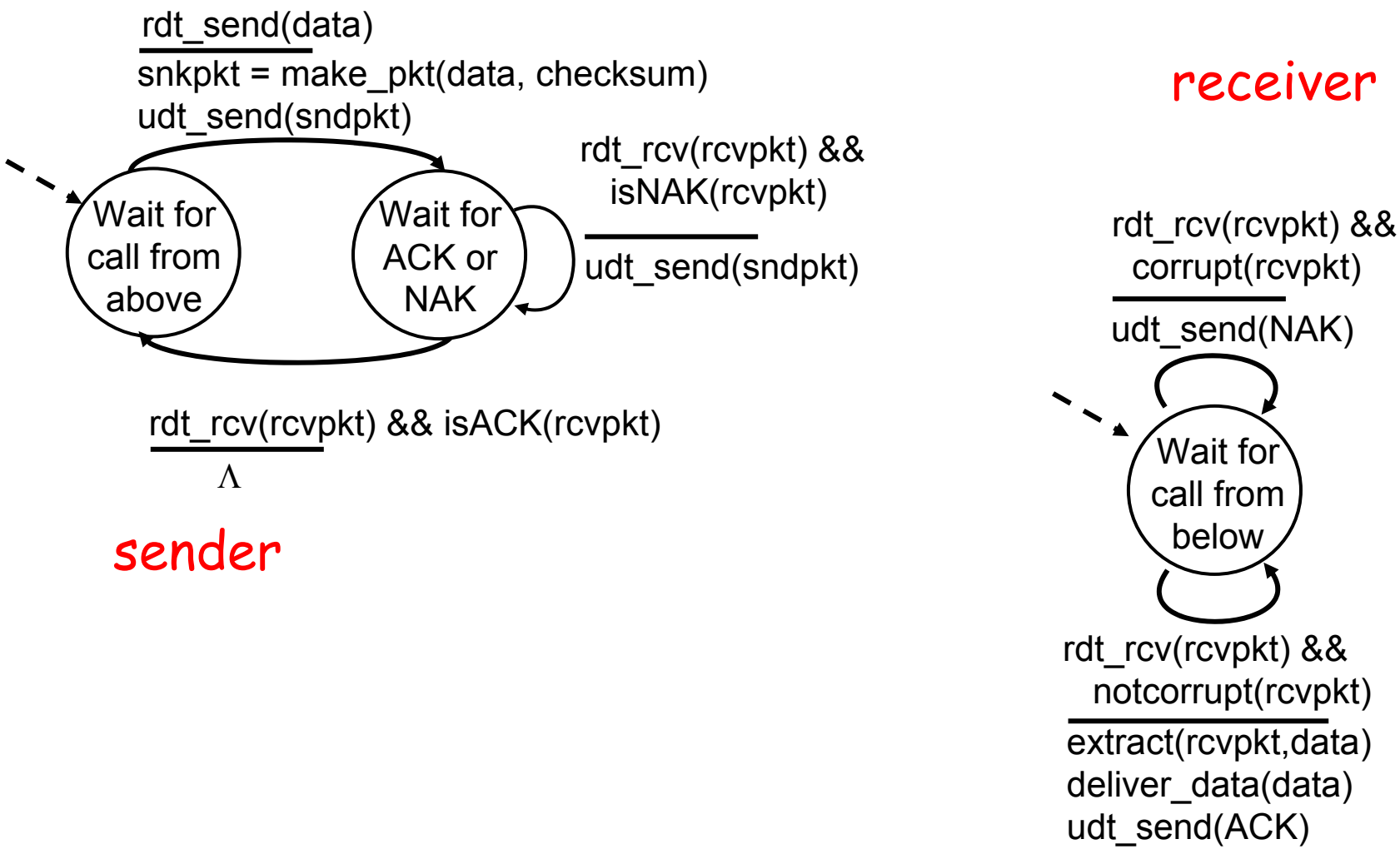


receiver

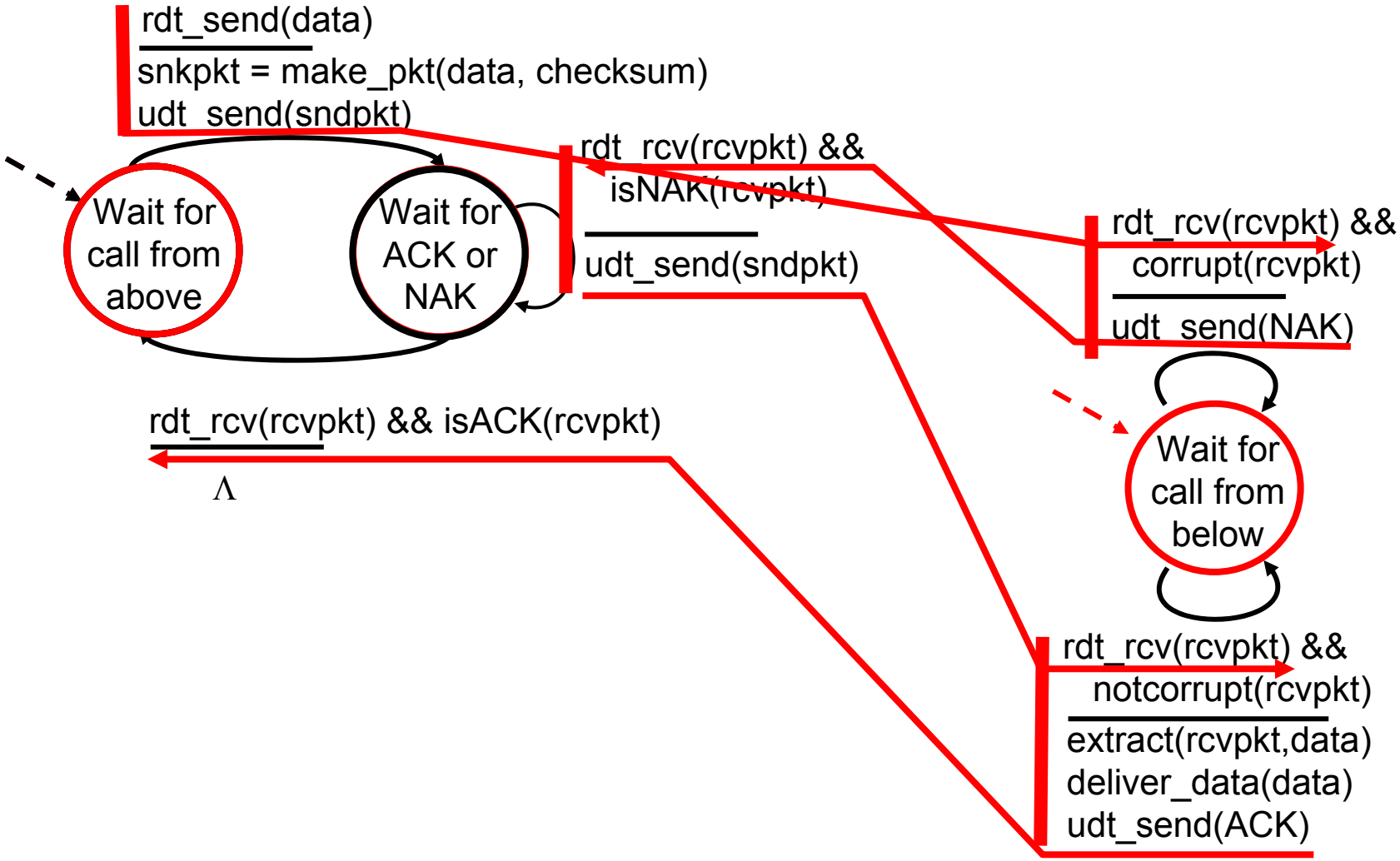
Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - recall: UDP checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
  - human scenarios using ACKs, NAKs?
- new mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender

rdt2.0: FSM specification



rdt2.0: error scenario



rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

What to do?

- sender ACKs/NAKs receiver's ACK/NAK? What if sender ACK/NAK lost?
- retransmit, but this might cause retransmission of correctly received pkt!

Handling duplicates:

- sender adds *sequence number* to each pkt
- sender retransmits current pkt if ACK/NAK garbled
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

Sender sends one packet, then waits for receiver response

rdt2.1: discussion

Sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice.
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "current" pkt has 0 or 1 seq. #

Receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

rdt3.0: channels with errors *and* loss

New assumption: underlying channel can also lose packets (data or ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

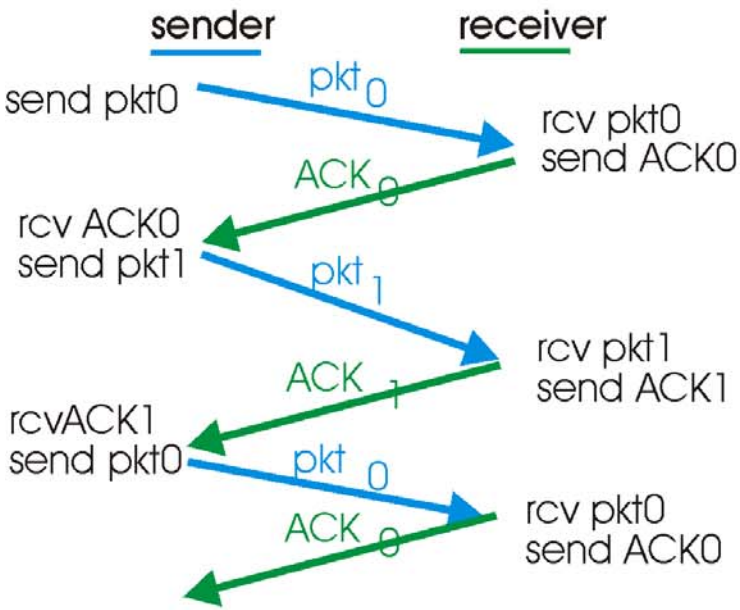
Q: how to deal with loss?

- sender waits until certain data or ACK lost, then retransmits

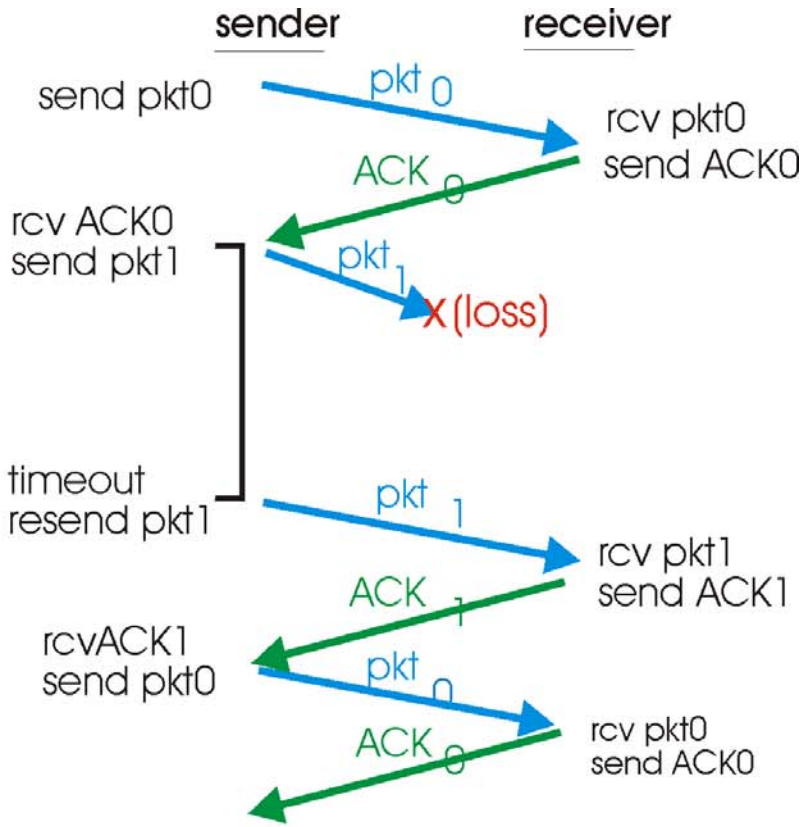
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

rdt3.0 in action

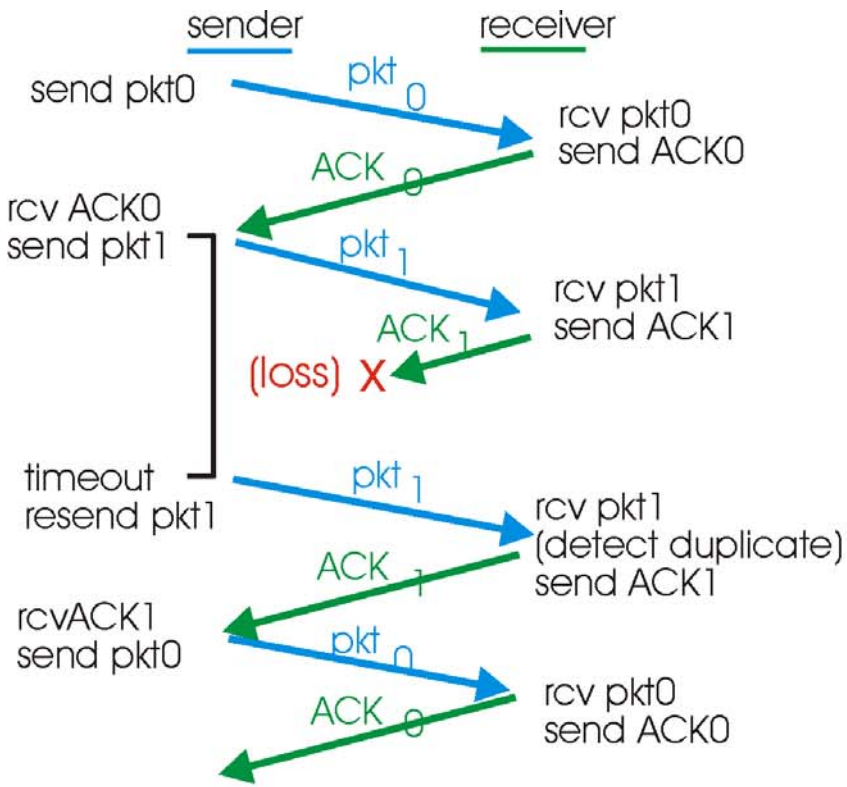


(a) operation with no loss

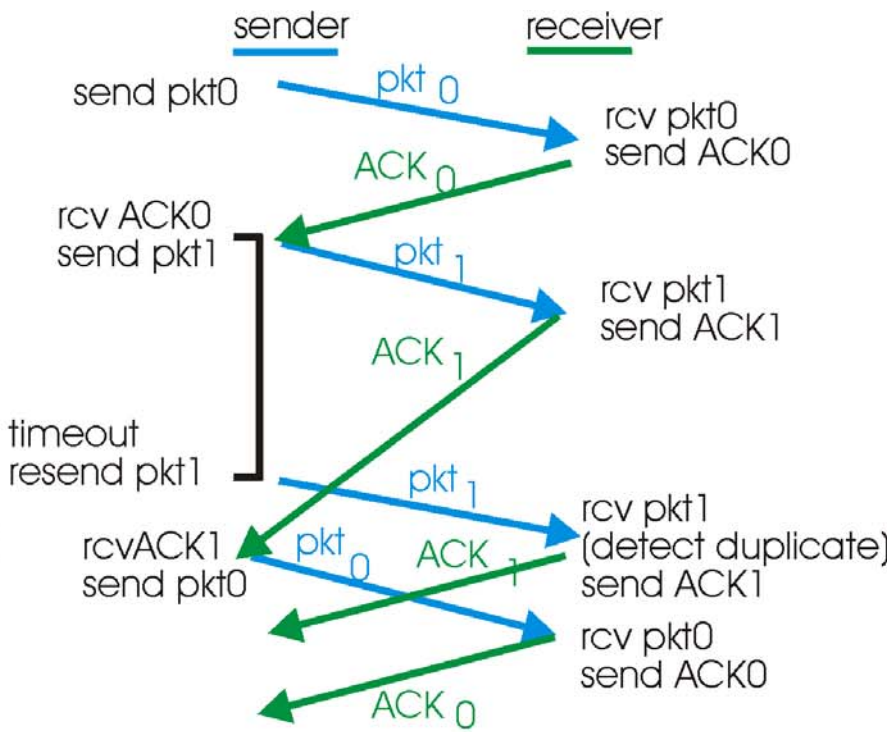


(b) lost packet

rdt3.0 in action



(c) lost ACK



(d) premature timeout

Performance of rdt3.0

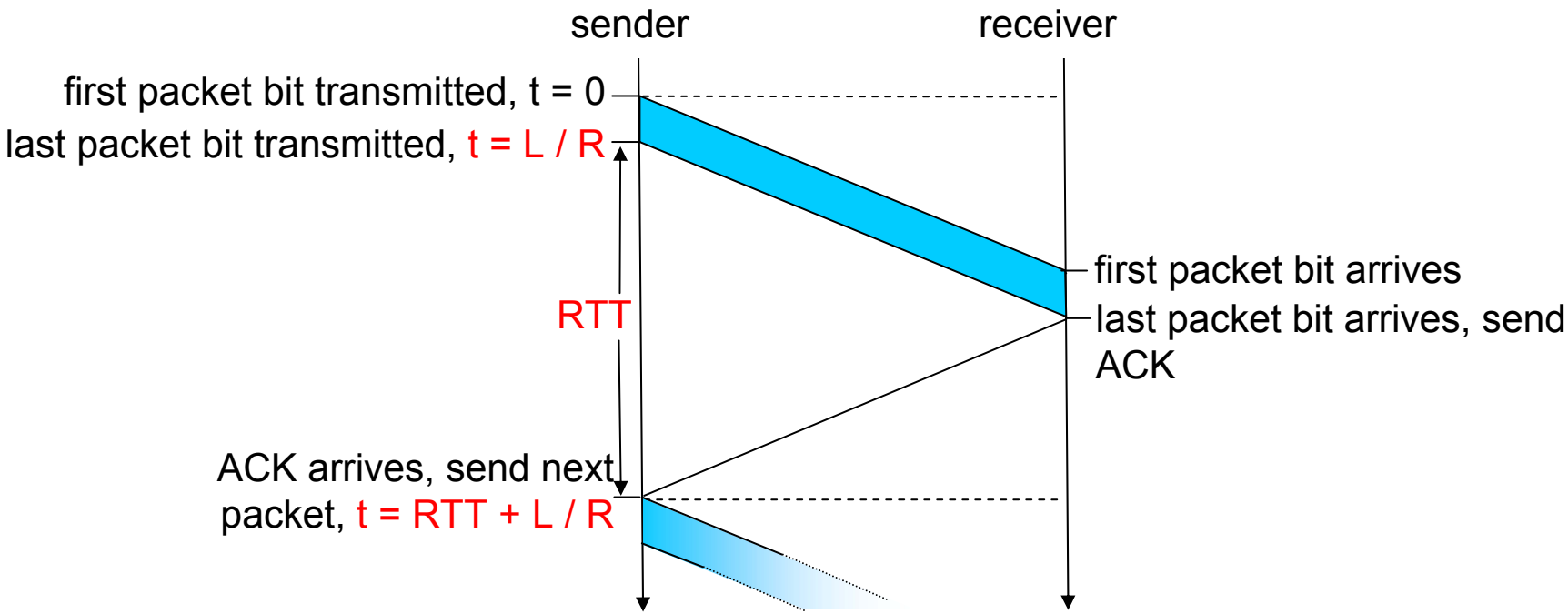
- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

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

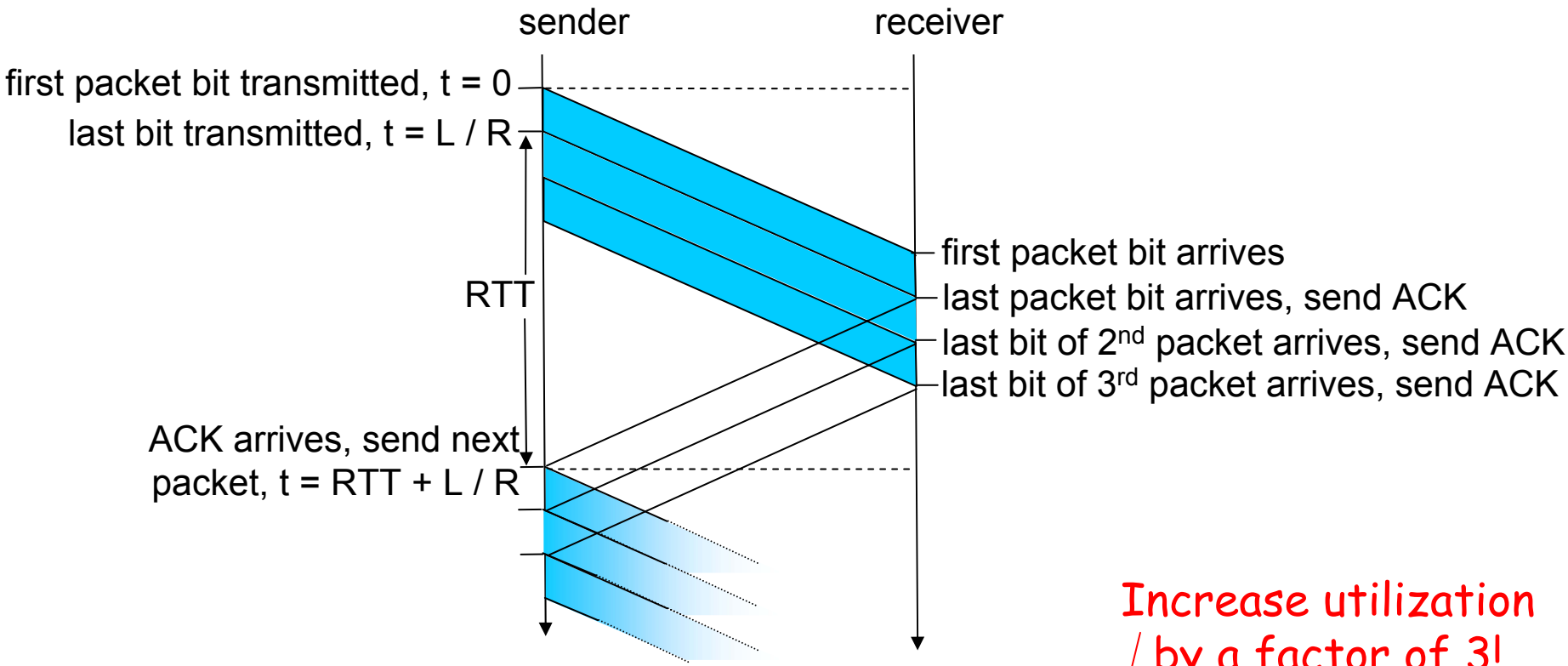
  - U_{sender}: utilization – fraction of time sender busy sending
  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation



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

Pipelining: increased utilization



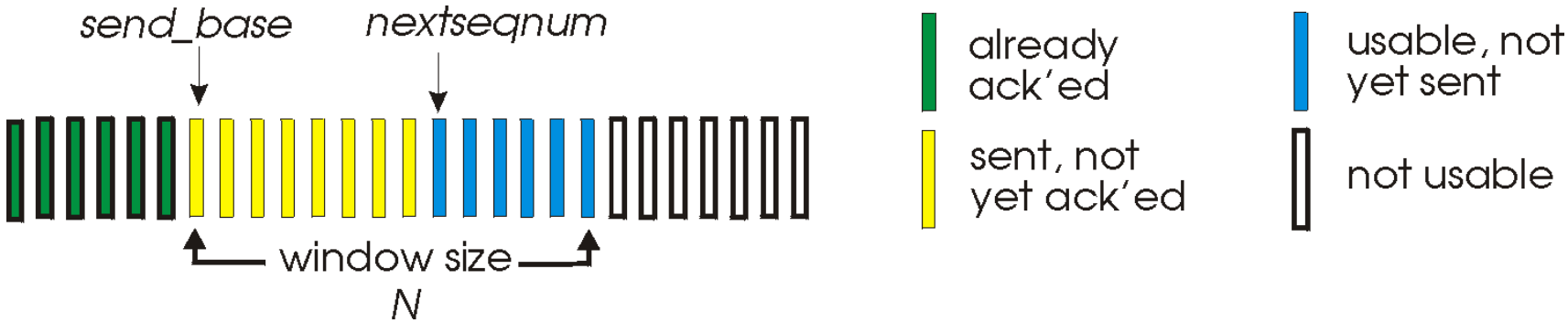
Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Go-Back-N

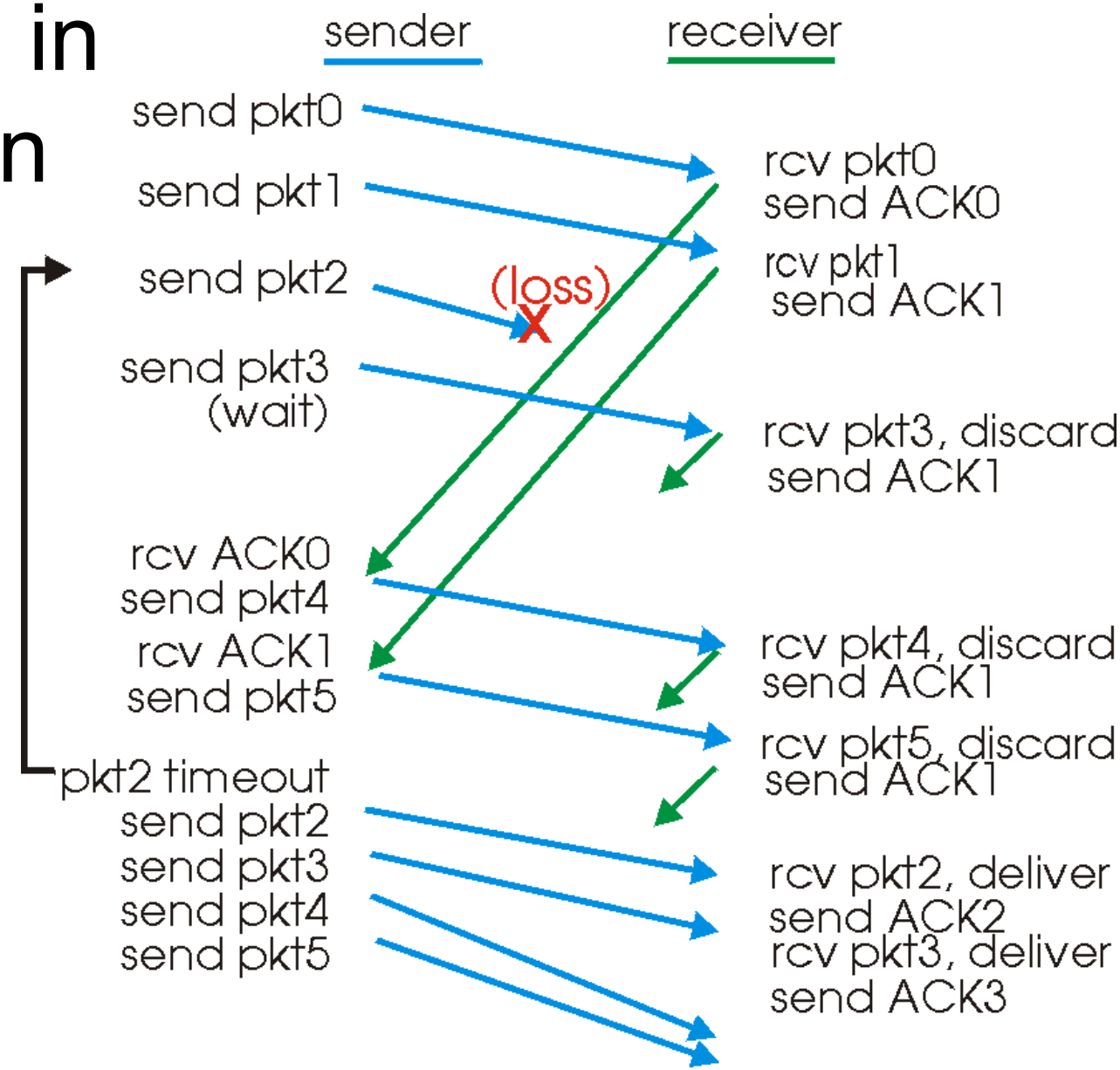
Sender:

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may deceive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

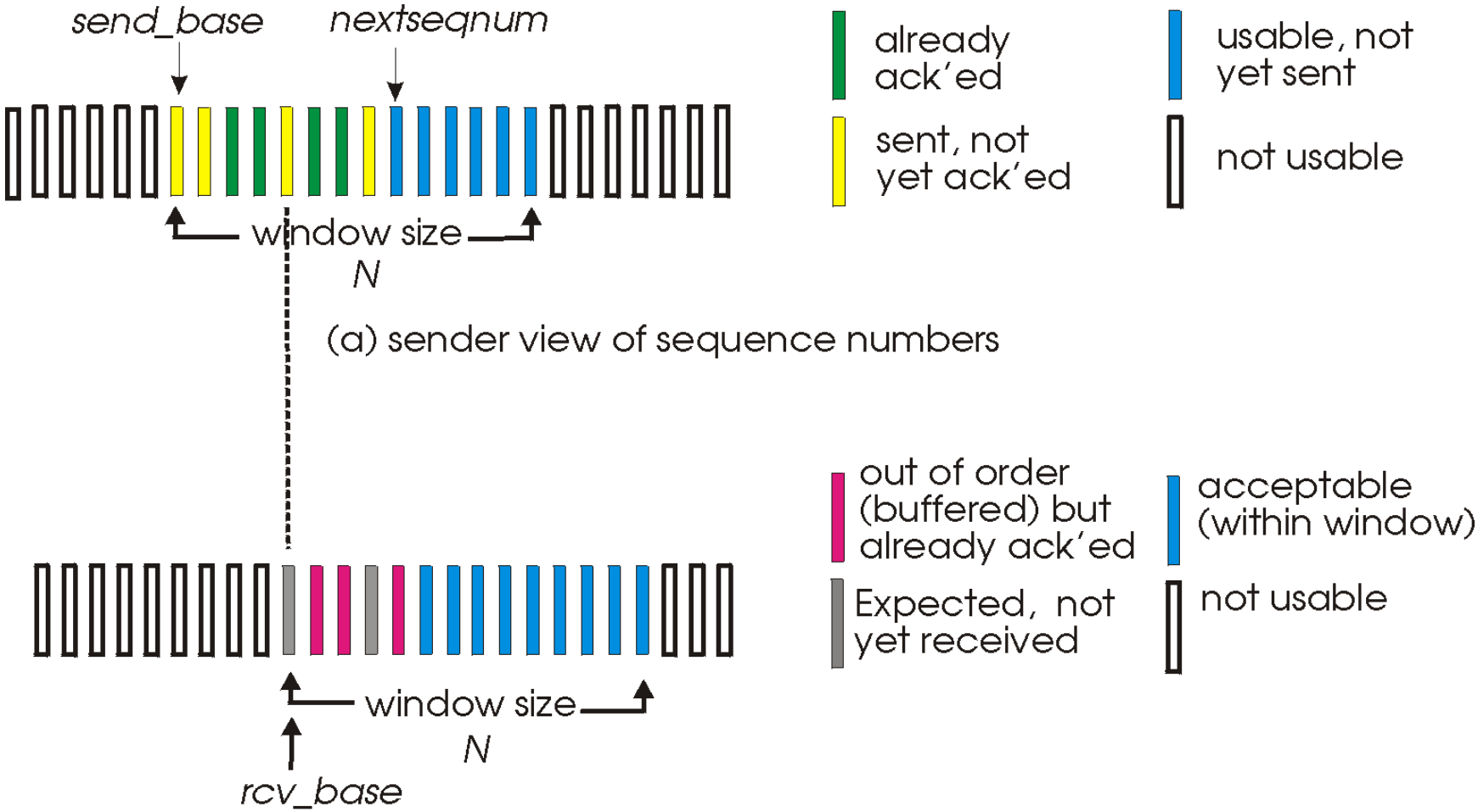
GBN in action



Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts

Selective repeat: sender, receiver windows



Selective repeat

sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

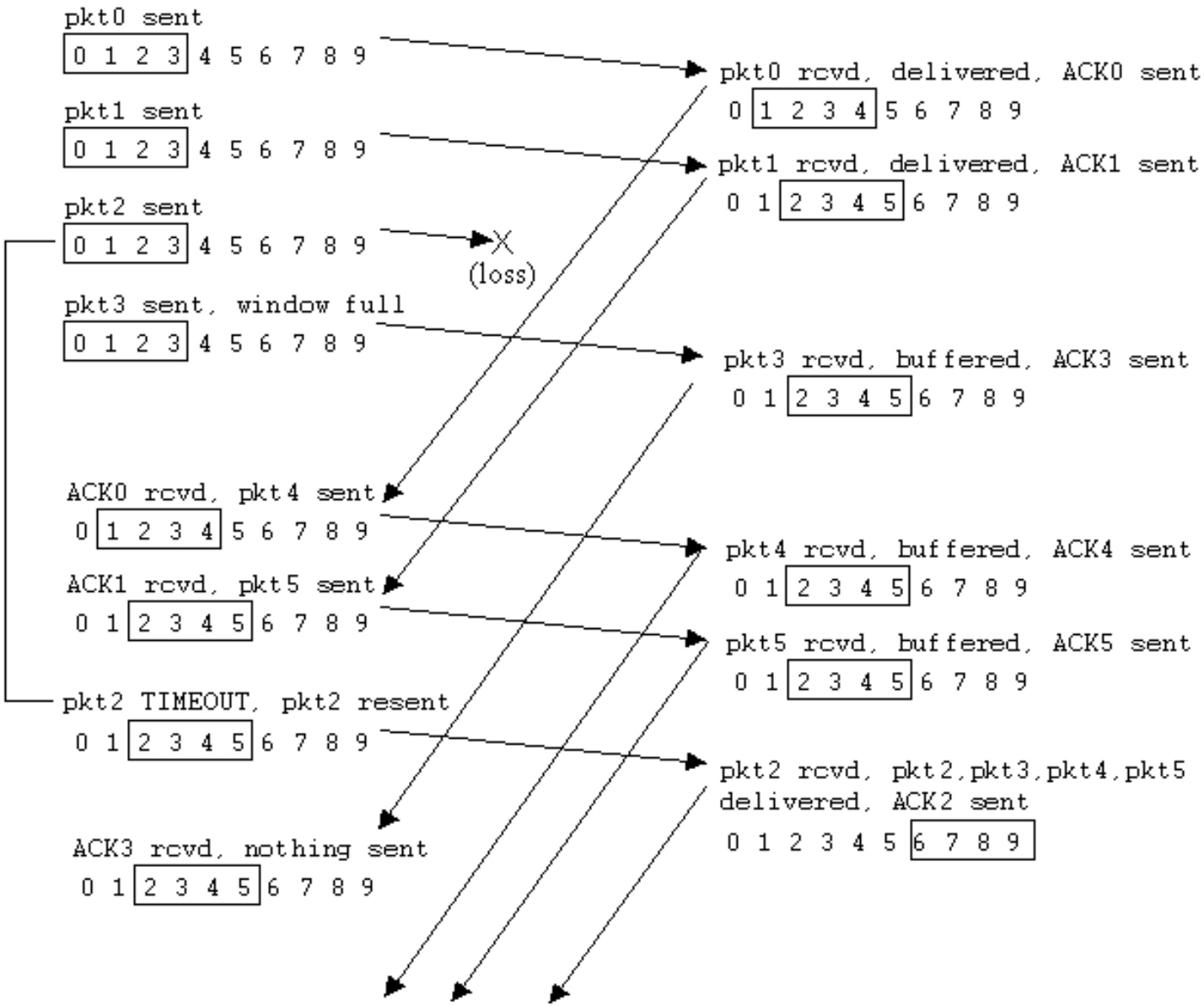
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat in action



Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- **Συνδεσιστρεφής μεταφορά: TCP**
  - **δομή segment**
  - **αξιόπιστη μεταφορά δεδομένων**
  - **έλεγχος ροής**
  - **διαχείριση συνδέσεων**
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

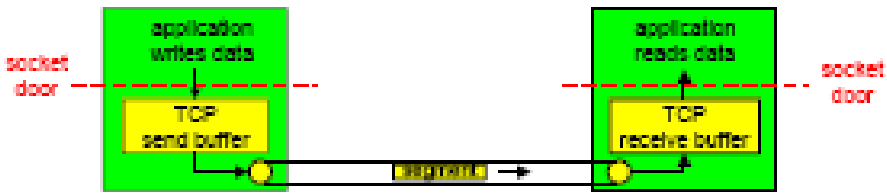
Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- **Συνδεσιστρεφής μεταφορά: TCP**
  - **δομή segment**
  - **αξιόπιστη μεταφορά δεδομένων**
  - **έλεγχος ροής**
  - **διαχείριση συνδέσεων**
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

TCP: Επισκόπηση

RFCs: 793, 1122, 1323, 2018, 2581

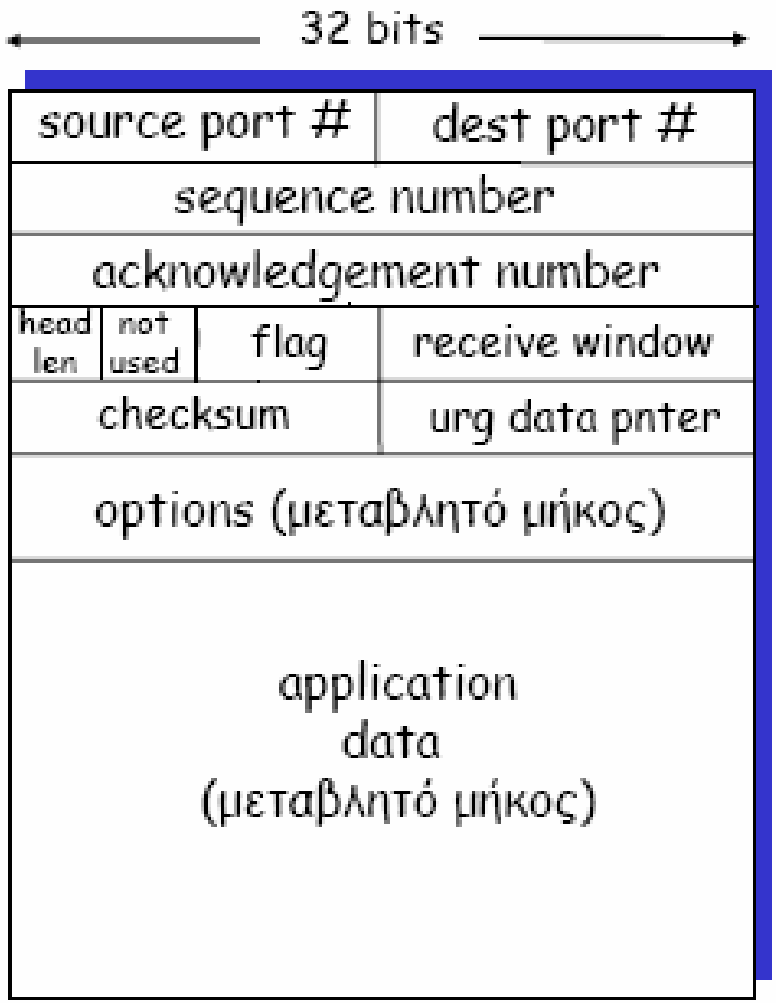
- connection-oriented:
  - με τη χειραψία (ανταλλαγή μηνυμάτων ελέγχου) γίνεται αρχικοποίηση της κατάστασης σε αποστολέα & παραλήπτη πριν αρχίσει η μεταφορά δεδομένων
- full duplex data:
  - αμφίδρομη ροή δεδομένων στην ίδια σύνδεση
- point-to-point:
  - ένας αποστολέας, ένας παραλήπτης
- *send & receive buffers*
  - MSS: maximum segment size
- αξιόπιστη, εν σειρά ροή από bytes (*byte stream*):
  - δεν υπάρχουν "όρια μηνυμάτων"
- έλεγχος ροής:
  - ώστε ο αποστολέας να μην υπερφορτώνει τον παραλήπτη
- pipelining:
  - ο έλεγχος ροής και συμφόρησης ορίζουν το μέγεθος του παραθύρου στο TCP



Δομή TCP segment

Πεδία:

- source, destination port #: πολύπλεξη/αποπολύπλεξη
- sequence, acknowledgment #: υλοποίηση αξιόπιστης μεταφοράς δεδομένων
- header length: μήκος επικεφαλίδας σε λέξεις των 32 bits
- receive window: αριθμός bytes που είναι διατεθειμένος να λάβει ο παραλήπτης
- checksum: όπως στο UDP
- options: προαιρετικό – δεν χρησιμοποιείται συνήθως
  - διαπραγμάτευση MSS μεταξύ αποστολέα & παραλήπτη
- application data: δεδομένα εφαρμογής (0 ≤ μήκος ≤ MSS)



Δομή TCP segment (συνέχεια)

Flag bits:

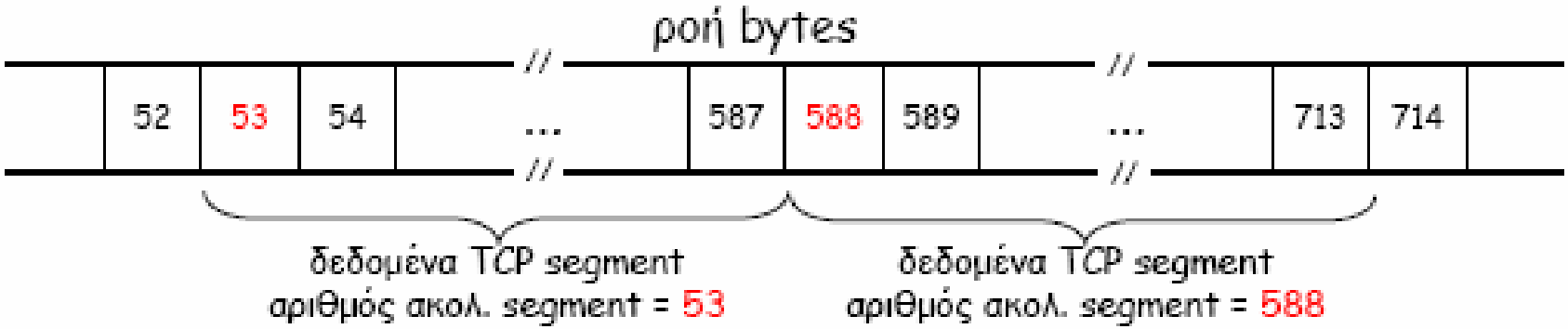
- URG: υπάρχουν δεδομένα στο segment που η εφαρμογή (πλευρά αποστολέα) έχει χαρακτηρίσει ως επείγοντα (δεν χρησιμοποιείται, εν γένει)
- ACK: ACK number έγκυρο
- PSH: άμεση προώθηση δεδομένων στην εφαρμογή από τον παραλήπτη
- RST (reset), SYN (synchronize), FIN (finish): χρησιμοποιούνται για την εγκαθίδρυση και τερματισμό σύνδεσης TCP (περισσότερα σε λίγο)

Πεδίο Flag

urgent data pointer: εάν URG = 1, τότε το πεδίο αυτό δείχνει τη θέση του τελευταίου byte των επειγόντων δεδομένων

Αριθμός ακολουθίας TCP

- Το TCP αντιμετωπίζει τα δεδομένα που λαμβάνει από την εφαρμογή ως μία διατεταγμένη ακολουθία (ροή) από bytes
- Τα bytes μίας ροής είναι αριθμημένα ακολουθιακά (αφανώς)
- Αριθμός ακολουθίας (sequence number) ενός segment: ο αριθμός του πρώτου byte δεδομένων στο segment στη ροή των bytes
- Ο αρχικός αριθμός της ακολουθίας των bytes επιλέγεται τυχαία



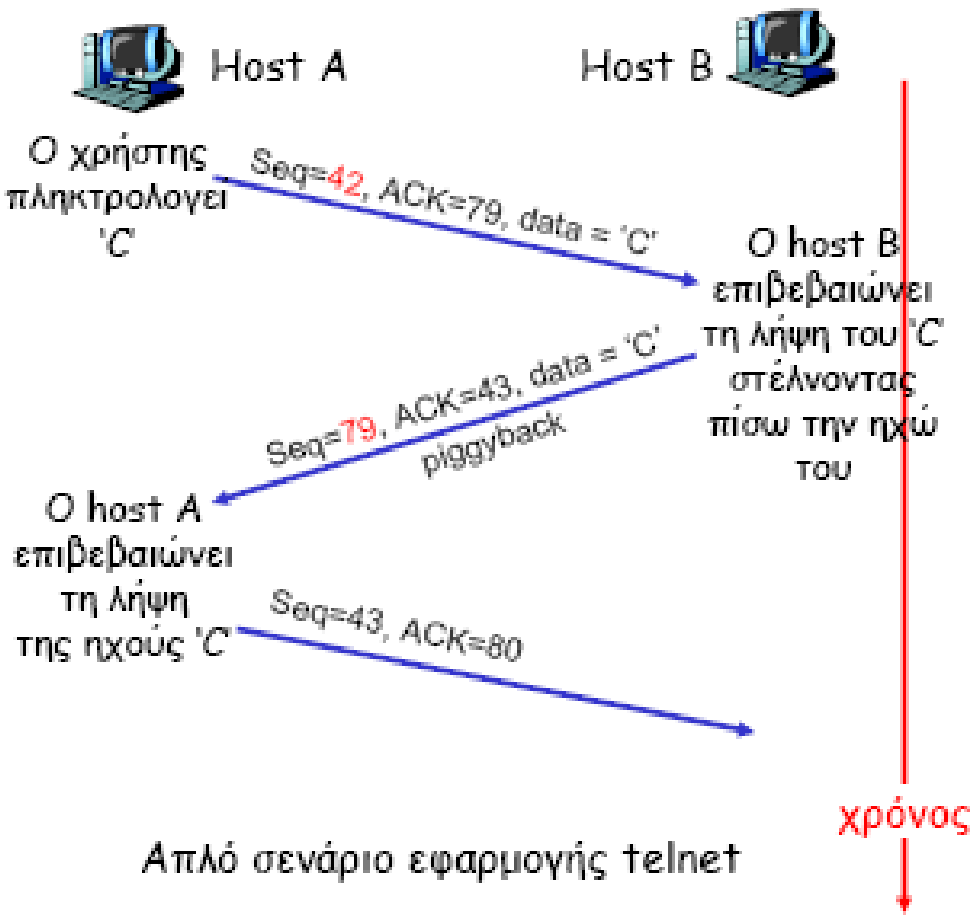
Αριθμός επιβεβαίωσης TCP

- αριθμός επιβεβαίωσης (acknowledgement number): αριθμός ακολουθίας του επόμενου αναμενόμενου byte
  π.χ. ο host A λαμβάνει το segment με bytes δεδομένων (0 - 535)
  ο host A στέλνει επιβεβαίωση (ACK) με αριθμό 536
- αθροιστικές επιβεβαιώσεις (cumulative acknowledgements): το TCP στέλνει επιβεβαίωση για όλα τα bytes που έχει λάβει στη σωστή σειρά μέχρι το πρώτο byte που εκλείπει
  π.χ. ο host A λαμβάνει το segment_1 (0 - 535) και το segment_3 (900 – 1000) αλλά όχι το segment_2 (536 - 899)
  ο host A στέλνει επιβεβαίωση (ACK) με αριθμό 536
- segments εκτός σειράς: segments που λαμβάνονται εκτός σειράς
  - είτε απορρίπτονται
  - είτε παραμένουν σε buffers έως ότου φθάσουν τα υπολειπόμενα

  ανάλογα με την υλοποίηση

TCP seq #, ack # (παράδειγμα)

- Μετά την αρχικοποίηση κατά την εγκαθίδρυση σύνδεσης TCP:
  - ο host B περιμένει το segment με αριθμό ακολουθίας 42
  - ο host A περιμένει το segment με αριθμό ακολουθίας 79



Απλό σενάριο εφαρμογής telnet

Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- **Συνδεσιστρεφής μεταφορά: TCP**
  - **δομή segment**
  - **αξιόπιστη μεταφορά δεδομένων**
  - **έλεγχος ροής**
  - **διαχείριση συνδέσεων**
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Αξιόπιστη μεταφορά δεδομένων στο TCP

- Το TCP δημιουργεί υπηρεσία αξιόπιστης μεταφοράς δεδομένων πάνω από την αναξιόπιστη υπηρεσία του IP χρησιμοποιώντας επαναμεταδόσεις
- Το TCP χρησιμοποιεί:
  - pipelining των segments
  - αθροιστικές επιβεβαιώσεις (cumulative ACKs)
  - ένα μόνο timer επαναμετάδοσης
- Γεγονότα που προκαλούν επαναμεταδόσεις:
  - timeouts
  - διπλότυπες επιβεβαιώσεις (duplicate ACKs)
- Θεωρούμε αρχικά απλοποιημένο αποστολέα TCP αγνοώντας:
  - διπλότυπες επιβεβαιώσεις
  - έλεγχο ροής, έλεγχο συμφόρησης

Γεγονότα στον TCP αποστολέα:

Γεγονότα που συμβαίνουν στην πλευρά του TCP αποστολέα:

- παραλαβή δεδομένων από εφαρμογή
- timeout
- λήψη επιβεβαίωσης ACK

παραλαβή δεδομένων από εφαρμογή:

- δημιουργία segment που φέρει αριθμό ακολουθίας
  - αριθμός ακολουθίας: αριθμός του πρώτου byte δεδομένων στο segment όπως προκύπτει από την ακολουθιακή αρίθμηση των bytes της ροής
- εκκίνηση timer εφόσον δεν βρίσκεται σε λειτουργία (θεωρείστε τον timer ως τον timer για το παλαιότερο ανεπιβεβαίωτο segment)
  - προθεσμία λήξης timer: `TimeOutInterval`
- προώθηση segment στο IP

Γεγονότα στον αποστολέα TCP: (συνέχεια)

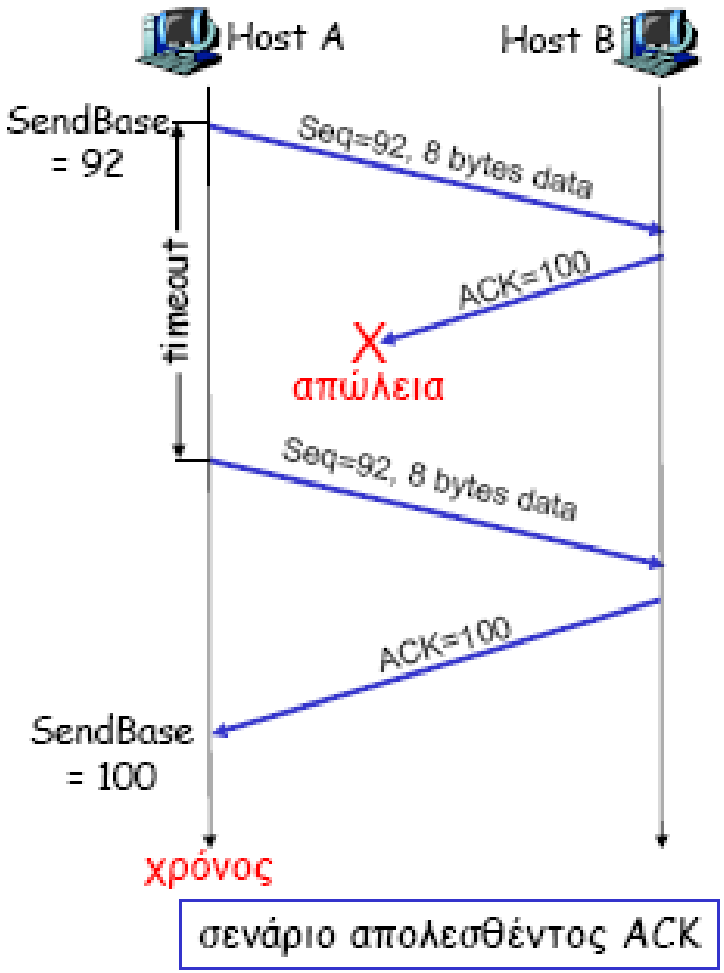
timeout:

- επαναμετάδοση ανεπιβεβαίωτου segment με το μικρότερο αριθμό ακολουθίας
- εκκίνηση timer

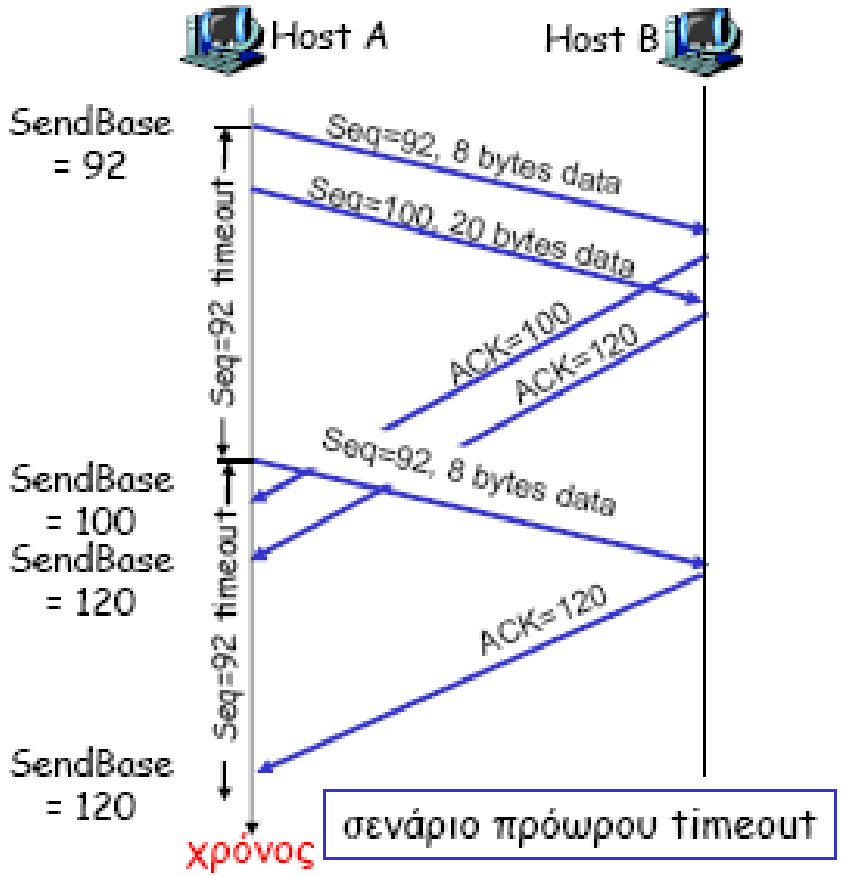
Λήψη επιβεβαίωσης ACK:

- Εάν πρόκειται για επιβεβαίωση segments που ήταν ανεπιβεβαίωτα προηγουμένως (πριν από τη λήψη του ACK):
  - ενημέρωση ως προς τα segments που επιβεβαιώνονται με το παρόν ACK
  - εκκίνηση timer εφόσον παραμένουν ανεπιβεβαίωτα segments

TCP: σενάρια επαναμεταδόσεων

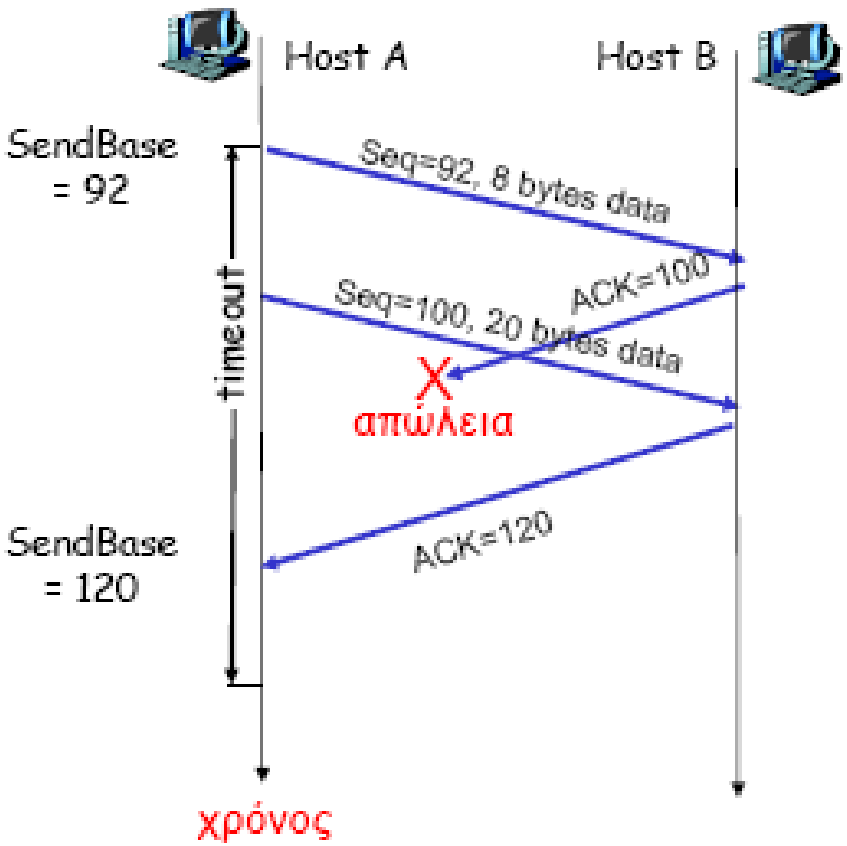


σενάριο απολεσθέντος ACK



σενάριο πρόωρου timeout

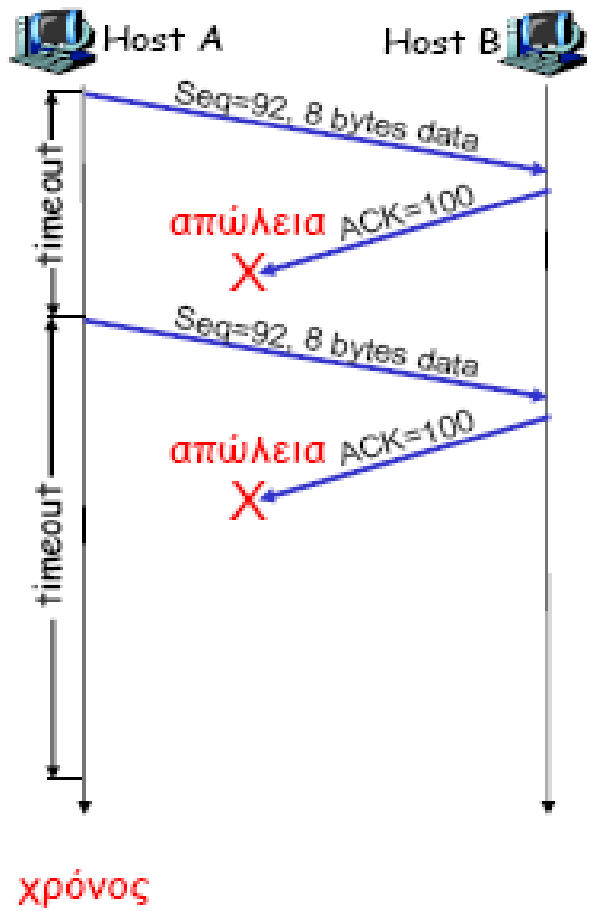
TCP: σενάρια επαναμεταδόσεων (συνέχεια)



σενάριο αθροιστικών ACK

TCP: διπλασιασμός της διάρκειας του timeout

- Μετά από timeout, το segment επαναμεταδίδεται
- Το timeout ενδεχομένως να οφείλεται σε συμφόρηση
- Προς αποφυγή περαιτέρω συμφόρησης, το `TimeOutInterval` για το επαναμεταδιδόμενο segment υπολογίζεται ως
  `TimeOutInterval =`
  `2×TimeOutInterval`
- Το `TimeOutInterval` διπλασιάζεται για κάθε διαδοχικό επαναμεταδιδόμενο segment (εκθετική αύξηση)
- Η υπαναχώρηση παύει με την εκκίνηση του timer για μη επαναμεταδιδόμενο segment



Αποστολή ACK στο TCP [RFC 1122, RFC 2581]

Γεγονός στον παραλήπτη	Δράση TCP παραλήπτη
Άφιξη segment εν σειρά με αναμενόμενο αριθμό ακολουθίας. Όλα τα δεδομένα έως τον αναμενόμενο αριθμό ακολουθίας είναι ήδη επιβεβαιωμένα.	Καθυστερημένο ACK. Αναμονή έως 500 ms για το επόμενο segment, αποστολή ACK.
Άφιξη segment εν σειρά με αναμενόμενο αριθμό ακολουθίας. Εκκρεμεί το ACK και ενός ακόμη segment.	Άμεση αποστολή ενός αθροιστικού ACK που επιβεβαιώνει αμφότερα τα εν σειρά segments.
Άφιξη segment εκτός σειράς με αριθμό ακολουθίας μεγαλύτερο του αναμενόμενου. Ανίχνευση κενού.	Άμεση αποστολή ενός διπλότυπου ACK που υποδεικνύει τον αριθμό ακολουθίας του επόμενου αναμενόμενου byte.
Άφιξη segment το οποίο συμπληρώνει, μερικώς ή εξ ολοκλήρου, το κενό.	Άμεση αποστολή ACK, υπό την προϋπόθεση ότι το segment αρχίζει στο κατώτερο άκρο του κενού.

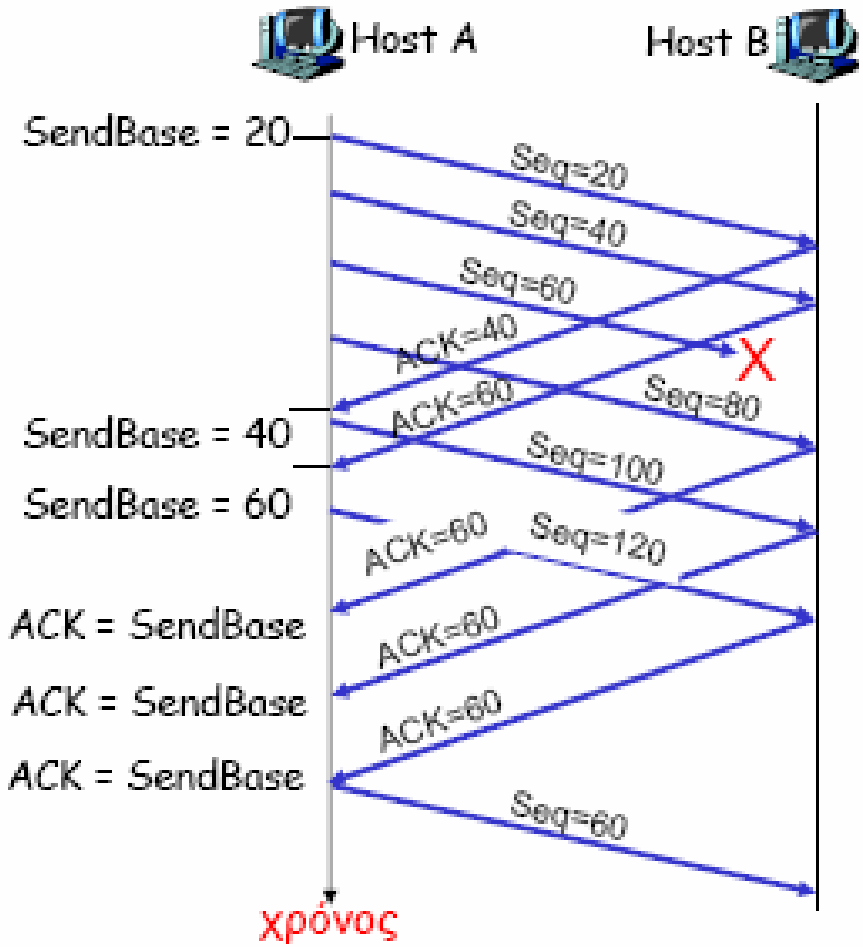
Ταχεία Επαναμετάδοση (Fast Retransmit)

- Η διάρκεια του timeout είναι συχνά σχετικά μεγάλη:
  - μεγάλη καθυστέρηση πριν από την επαναμετάδοση απολεσθέντος segment
- Ανίχνευση απολεσθέντων segments με διπλότυπα ACK
  - Διπλότυπο ACK: ACK που επιβεβαιώνει ένα segment για το οποίο ο αποστολέας έχει ήδη λάβει επιβεβαίωση
  - Ο αποστολέας συχνά στέλνει πολλά segments το ένα αμέσως μετά το άλλο
  - Εάν ένα segment χαθεί, ο αποστολέας θα λάβει πολλά διπλότυπα ACK
- Εάν ο αποστολέας λάβει 3 διπλότυπα ACK για το ίδιο segment, υποθέτει ότι το ακόλουθο segment (που μεταδόθηκε μετά από αυτό που έχει επιβεβαιωθεί) χάθηκε:
  - ταχεία επαναμετάδοση: επαναμετάδοση του segment που θεωρείται χαμένο πριν από τη λήξη του timer επαναμετάδοσης

Ταχεία επαναμετάδοση

Παράδειγμα:

- Κάθε segment φέρει 20 bytes δεδομένων
- Το τρίτο segment με αριθμό ακολουθίας 60 χάνεται
- Μετά τη λήψη 3 διπλότυπων ACK = 60 για το δεύτερο segment με αριθμό ακολουθίας 40, το τρίτο segment με αριθμό ακολουθίας 60 επαναμεταδίδεται



TCP Round Trip Time και Timeout

Επιλογή τιμής timeout στο TCP

- timeout > RTT
  - όμως RTT είναι τυχαία μεταβλητή
- τιμή timeout πολύ μικρή: πρόωρο timeout
  - άσκοπες επαναμεταδόσεις
- τιμή timeout πολύ μεγάλη:
  - αργή αντίδραση στις απώλειες segments
- η τιμή του timeout καθορίζεται ως συνάρτηση της εκτιμώμενης, από μετρήσεις, τιμής του RTT

Εκτίμηση του RTT:

βασίζεται στις τιμές δειγμάτων

- `SampleRTT`: μετρούμενος χρόνος από τη μετάδοση ενός segment έως τη λήψη της επιβεβαίωσής του
  - το TCP αγνοεί δείγματα που προέρχονται από επαναμεταδιδόμενα segments (Γιατί;)
- Το TCP δεν λαμβάνει δείγματα `SampleRTT` για κάθε segment
  - ένα `SampleRTT` κάθε RTT περίπου
- Για την εκτίμηση του RTT επιλέγεται ένας μέσος όρος των πρόσφατων δειγμάτων `SampleRTT`

Εκτίμηση του RTT

- Για την εκτίμηση του RTT επιλέγεται ένας μέσος όρος των προσφάτων δειγμάτων SampleRTT με εκθετικά βάρη (exponential weighted moving average):

$$\texttt{EstimatedRTT}(n) = (1-\alpha)\,\texttt{EstimatedRTT}(n-1) + \alpha\,\texttt{SampleRTT}(n),\ \alpha < 1$$

- η επίδραση των παλαιοτέρων δειγμάτων φθίνει εκθετικά με το n
- όσο μεγαλύτερη η τιμή του α τόσο μεγαλύτερο το βάρος που δίνεται στις πιο πρόσφατες μετρήσεις
  - τυπική τιμή: $\alpha = 0.125$ (1/8)

Η τιμή του χρονικού διαστήματος timeout τίθεται σε:

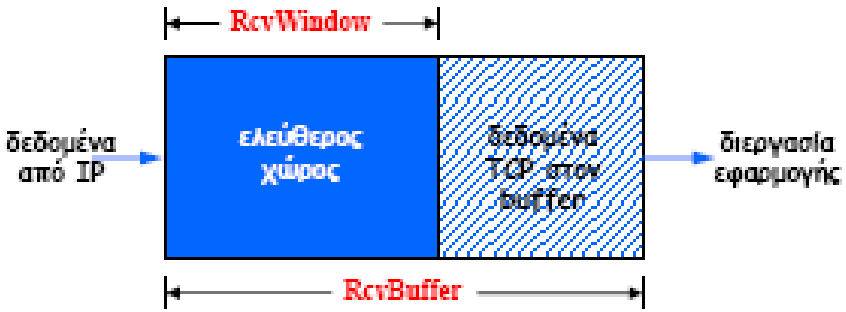
```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- **Συνδεσιστρεφής μεταφορά: TCP**
  - **δομή segment**
  - **αξιόπιστη μεταφορά δεδομένων**
  - **έλεγχος ροής**
  - **διαχείριση συνδέσεων**
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Έλεγχος ροής στο TCP

- η σύνδεση TCP τοποθετεί τα bytes που λαμβάνει σωστά και εν σειρά στο receive buffer:



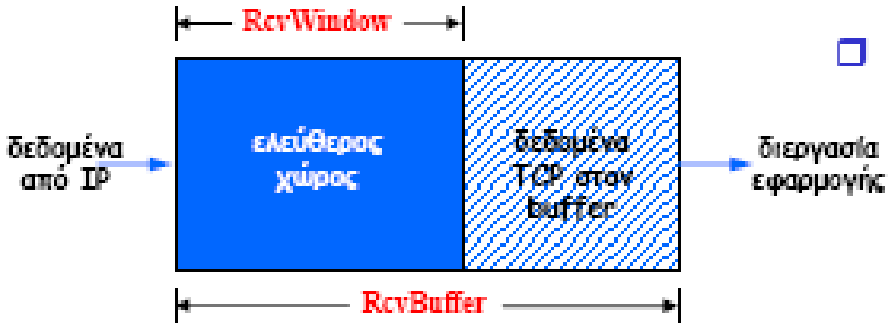
- όταν η διεργασία της εφαρμογής αργεί να διαβάσει τα δεδομένα από το receive buffer, τότε ενδέχεται να υπερχειλίσει ο buffer

έλεγχος ροής

ο αποστολέας δεν υπερχειλίζει τον buffer του παραλήπτη στέλνοντας μεγάλο όγκο δεδομένων με μεγάλο ρυθμό

- υπηρεσία προσαρμογής του ρυθμού αποστολής στο ρυθμό με τον οποίο διαβάζονται τα δεδομένα από τον παραλήπτη

Έλεγχος ροής στο TCP: υλοποίηση



(Έστω ότι ο παραλήπτης απορρίπτει τα segments που λαμβάνει εκτός σειράς)

- ο ελεύθερος χώρος στο buffer

= `RcvWindow`

= `RcvBuffer-[LastByteRcvd - LastByteRead]`

μεταβάλλεται με δυναμικό τρόπο

- Ο παραλήπτης ενημερώνει τον αποστολέα για τον ελεύθερο χώρο στο buffer περιλαμβάνοντας την τρέχουσα τιμή του `RcvWindow` σε κάθε segment (πεδίο receive window) που στέλνει στον αποστολέα
- Ο αποστολέας περιορίζει τα ανεπιβεβαίωτα δεδομένα στην τιμή του `RcvWindow`:

`LastByteSent - LastByteAcked` ≤ `RcvWindow`

  - εγγύηση ότι ο receive buffer δεν θα υπερχειλίσει
- deadlock όταν `RcvWindow = 0`

Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- **Συνδεσιστρεφής μεταφορά: TCP**
  - **δομή segment**
  - **αξιόπιστη μεταφορά δεδομένων**
  - **έλεγχος ροής**
  - **διαχείριση συνδέσεων**
- Αρχές ελέγχου συμφόρησης
- Έλεγχος συμφόρησης στο TCP

Διαχείριση σύνδεσης TCP

- Εγκαθίδρυση σύνδεσης μεταξύ TCP αποστολέα και παραλήπτη πριν από την ανταλλαγή δεδομένων
- αρχικοποίηση μεταβλητών:
  - sequence numbers
  - buffers, πληροφορία ελέγχου ροής (π.χ. **RcvWindow**)

- Ο *client* ξεκινά τη σύνδεση με τον *server* ακολουθώντας

Χειραψία τριών βημάτων:

Βήμα 1: Ο client στέλνει TCP SYN segment (SYN = 1) στον server

- προσδιορίζει το αρχικό seqnum του client: seqnum = client_isn
- data = ∅

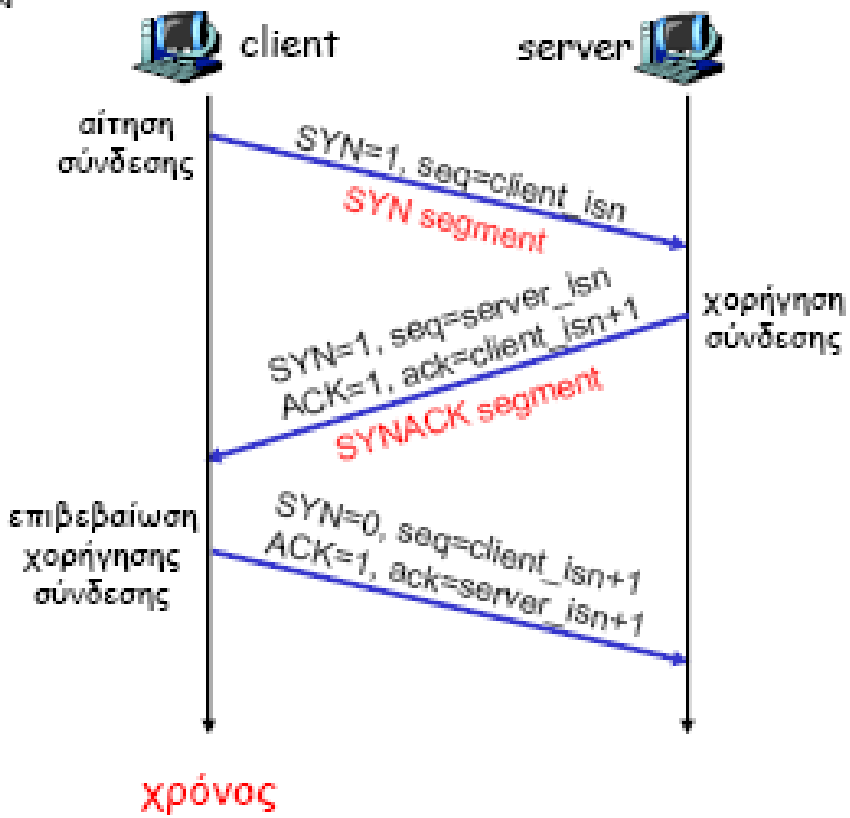
Διαχείριση σύνδεσης TCP (συνέχεια)

Βήμα 2: Ο server λαμβάνει το SYN segment, απαντά με SYNACK segment (SYN = 1, ACK = 1)

- Ο server εκχωρεί buffers
- προσδιορίζει αρχικό seqnum seqnum = server_isn
- acknum = client_isn+1
- data = ∅

Βήμα 3: Ο client λαμβάνει SYNACK, απαντά με ACK segment (SYN = 0, ACK = 1) το οποίο μπορεί να περιέχει data

- Ο client εκχωρεί buffers
- seqnum = client_isn+1
- acknum = server_isn+1



Διαχείριση σύνδεσης TCP (συνέχεια)

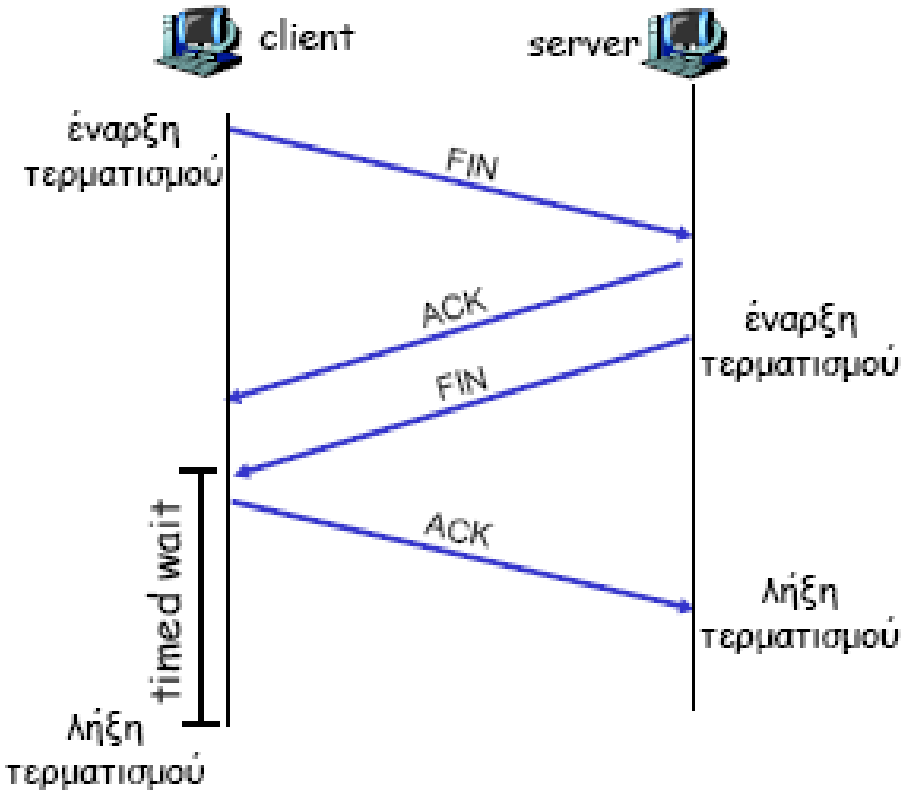
Τερματισμός σύνδεσης:

Ο client αποφασίζει να τερματίσει τη σύνδεση

Βήμα 1: Ο client στέλνει TCP FIN segment (FIN = 1) στον server

Βήμα 2: Ο server λαμβάνει το FIN segment, απαντά με ACK

Ο server αρχίζει διαδικασία τερματισμού της σύνδεσης στέλνοντας FIN segment στον client

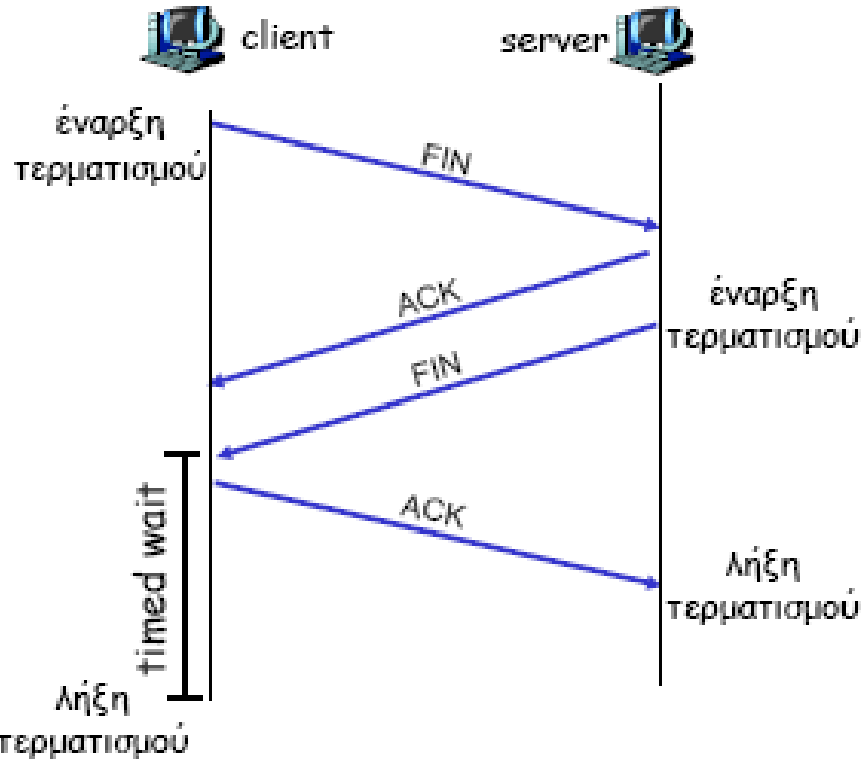


Διαχείριση σύνδεσης TCP (συνέχεια)

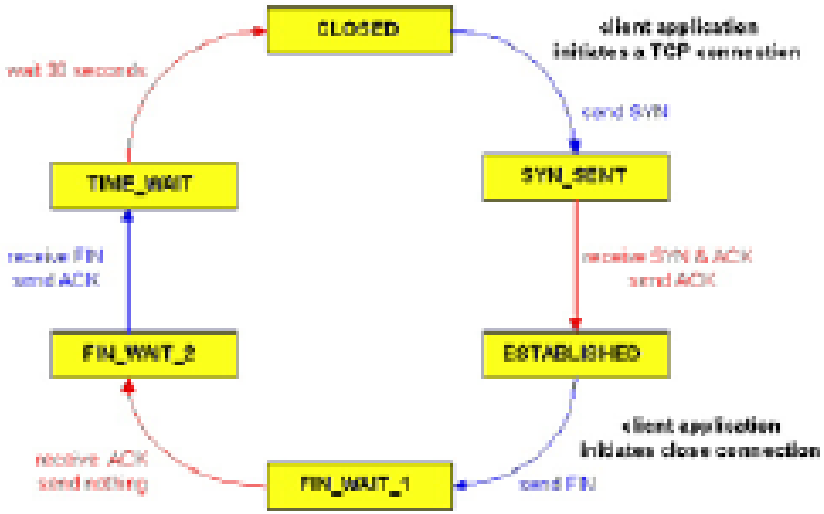
Βήμα 3: Ο client λαμβάνει το FIN segment, απαντά με ACK

- Κάνει μετάβαση στην κατάσταση "timed wait" – απαντά με ACK σε λαμβανόμενα FIN segments

Βήμα 4: Ο server λαμβάνει ACK – Η σύνδεση τερματίζεται

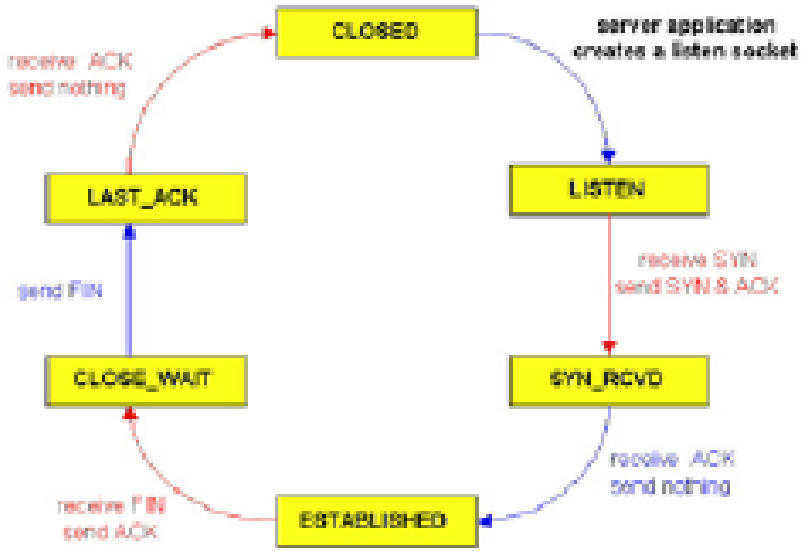


Διαχείριση σύνδεσης TCP (συνέχεια)

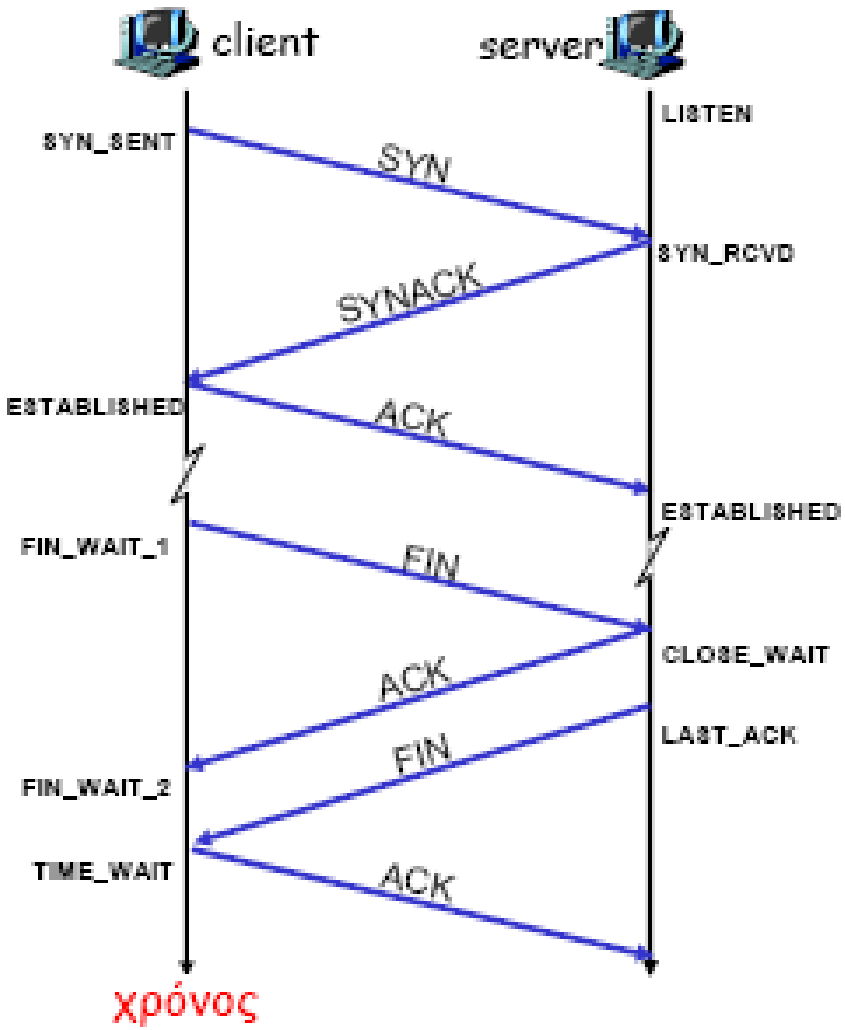


κύκλος ζωής
TCP client

κύκλος ζωής
TCP server



Διαχείριση σύνδεσης TCP (συνέχεια)



Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- **Αρχές ελέγχου συμφόρησης**
- Έλεγχος συμφόρησης στο TCP

Αρχές ελέγχου συμφόρησης

Συμφόρηση:

- αίτια: το δίκτυο δεν μπορεί να ανταπεξέλθει στο φόρτο της κίνησης (μεγάλος αριθμός πηγών, μεγάλος όγκος δεδομένων, υψηλός ρυθμός μετάδοσης των πηγών)
- αποτελέσματα:
  - απώλειες πακέτων (υπερχείλιση των buffers των δρομολογητών)
  - μεγάλες καθυστερήσεις (αναμονή στους buffers των δρομολογητών)

Έλεγχος συμφόρησης:

- μηχανισμοί αποφυγής συμφόρησης ή αντίδρασης στη συμφόρηση
- διαφέρει από τον έλεγχο ροής

Διαφορετικές προσεγγίσεις στο πρόβλημα του ελέγχου συμφόρησης

Δύο ευρείες κατηγορίες μεθόδων ελέγχου συμφόρησης:

Έλεγχος συμφόρησης από άκρο σε άκρο:

- κανένα άμεσο feedback από το δίκτυο
- συμφόρηση συνάγεται από τις παρατηρούμενες απώλειες, καθυστερήσεις στα τερματικά συστήματα
- η προσέγγιση αυτή έχει υιοθετηθεί στο TCP

Έλεγχος συμφόρησης επι-βοηθούμενος από το δίκτυο:

- οι δρομολογητές παρέχουν feedback στα τερματικά συστήματα
  - ένα bit που υποδεικνύει συμφόρηση (DECbit, ATM EERC)
  - ρητός ρυθμός για τον αποστολέα (ATM EERC)
- ακριβέστερος έλεγχος ρυθμού μετάδοσης
- αυξημένη πολυπλοκότητα

Περίγραμμα

- Υπηρεσίες επιπέδου μεταφοράς
- Πολύπλεξη και αποπολύπλεξη
- Ασυνδεσιστρεφής μεταφορά: UDP
- Αρχές αξιόπιστης μεταφοράς δεδομένων
- Συνδεσιστρεφής μεταφορά: TCP
  - δομή segment
  - αξιόπιστη μεταφορά δεδομένων
  - έλεγχος ροής
  - διαχείριση συνδέσεων
- Αρχές ελέγχου συμφόρησης
- **Έλεγχος συμφόρησης στο TCP**

Έλεγχος συμφόρησης στο TCP

- Το TCP αναγκαστικά χρησιμοποιεί έλεγχο από άκρο σε άκρο: το IP δεν παρέχει πληροφορία συμφόρησης **πάντα**
- Κάθε αποστολέας ελέγχει το ρυθμό μετάδοσης ως συνάρτηση της συμφόρησης που αντιλαμβάνεται

Τρόπος μεταβολής ρυθμού:

- με τη μεταβλητή congestion window (CongWin):

  `LastByteSent-LastByteAcked ≤ min{CongWin, RcvWindow}`

- Χονδρικά, για το ρυθμό αποστολής ισχύει:

$$\text{ρυθμός} = \frac{\text{CongWin}}{\text{RTT}} \quad \text{bytes/sec}$$

- Το παράθυρο CongWin μεταβάλλεται δυναμικά, ως συνάρτηση της συμφόρησης που αντιλαμβάνεται ο αποστολέας

Έλεγχος συμφόρησης στο TCP

Τρόπος με τον οποίο ο αποστολέας αντιλαμβάνεται τη συμφόρηση:

- γεγονός απώλειας = timeout ή 3 διπλότυπα ack
- ο αποστολέας μειώνει το ρυθμό (**CongWin**) μετά από ένα γεγονός απώλειας

Αλγόριθμος ελέγχου συμφόρησης

Τρεις κύριοι μηχανισμοί:

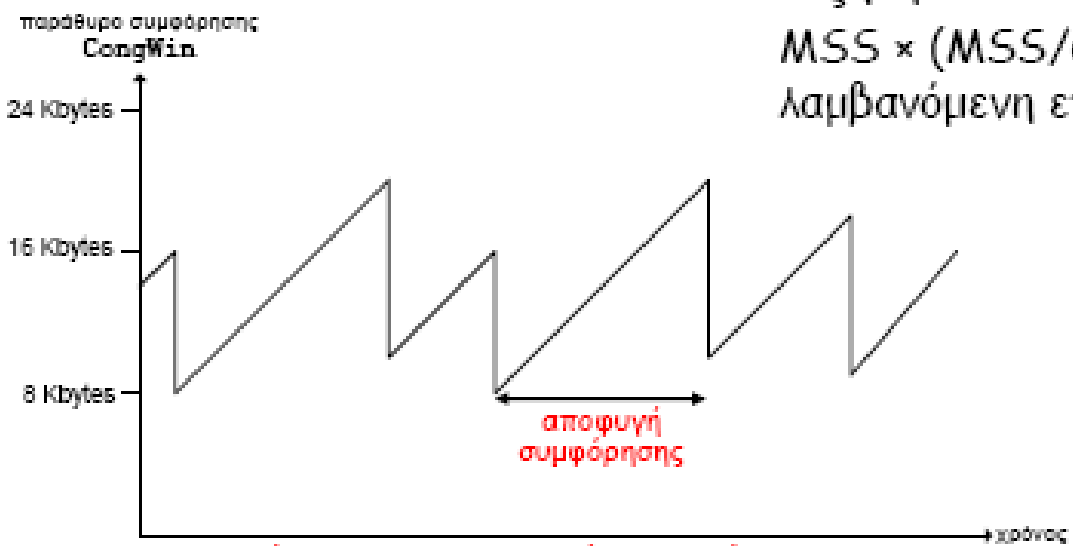
- προσθετική αύξηση – πολλαπλασιαστική μείωση (Additive Increase Multiplicative Decrease - AIMD)
- αργή εκκίνηση (slow start)
- αντίδραση σε γεγονότα timeout

Προσθετική αύξηση – πολλαπλασιαστική μείωση στο TCP

πολλαπλασιαστική μείωση:
υποδιπλασιασμός του CongWin μετά από κάθε γεγονός απώλειας

προσθετική αύξηση:
αύξηση του CongWin κατά 1 MSS κάθε RTT όσο δεν συμβαίνουν γεγονότα απώλειας: *ανίχνευση (probing)*

αύξηση του CongWin κατά MSS × (MSS/CongWin) με κάθε λαμβανόμενη επιβεβαίωση



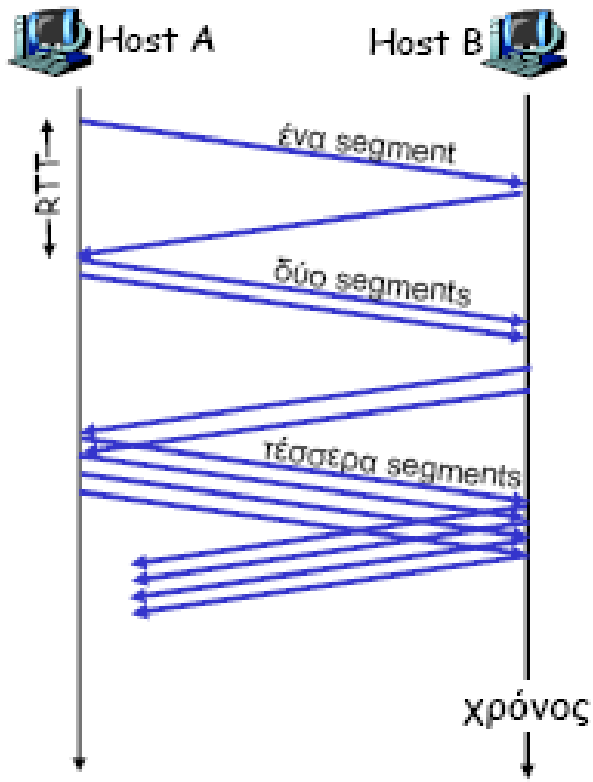
Σύνδεση TCP μεγάλης διάρκειας

Αργή εκκίνηση στο TCP

- αρχική τιμή `CongWin` = 1 MSS ⇒
  αρχικός ρυθμός αποστολής ≈ MSS/RTT
  - π.χ. MSS = 500 bytes, RTT = 200 msec
    ⇒ αρχικός ρυθμός MSS/RTT = 20 kbps
- ενδεχομένως διαθέσιμη χωρητικότητα >> MSS/RTT
  - γραμμική αύξηση ρυθμού πολύ αργή σε σχέση με διαθέσιμη χωρητικότητα
  - επιθυμητή η ταχεία αύξηση του ρυθμού σε επιτρεπτό επίπεδο
- στο ξεκίνημα της σύνδεσης ο αποστολέας αυξάνει το ρυθμό του εκθετικά μέχρι το πρώτο γεγονός απώλειας (αργή εκκίνηση αντί αποφυγή συμφόρησης)

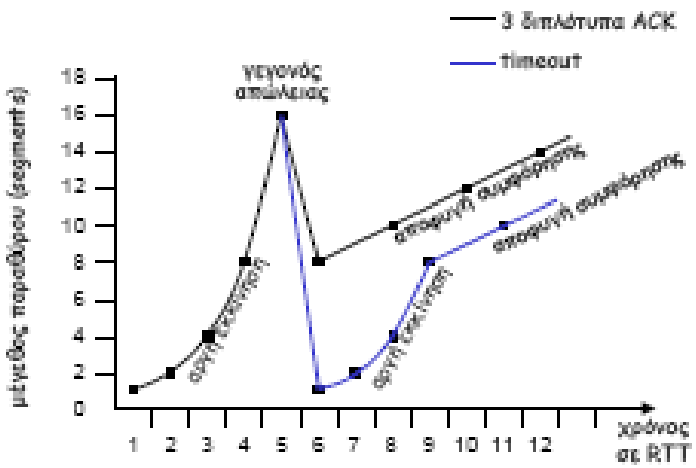
Αργή εκκίνηση στο TCP (συνέχεια)

- στο ξεκίνημα της σύνδεσης ο ρυθμός αυξάνει εκθετικά μέχρι το πρώτο γεγονός απώλειας
  - διπλασιασμός της τιμής CongWin κάθε RTT
  - επιτυγχάνεται με την αύξηση του CongWin κατά 1 MSS με κάθε λαμβανόμενη επιβεβαίωση ACK
- Αργή εκκίνηση (slow start): μικρός αρχικός ρυθμός αποστολής που όμως αυξάνεται εκθετικά



Διευκρίνιση

- Μετά από 3 διπλότυπα ACK:
  - **CongWin** μειώνεται στο μισό
  - στη συνέχεια το παράθυρο αυξάνεται γραμμικά
- Όμως μετά από timeout:
  - **CongWin** παίρνει τιμή 1 MSS
  - στη συνέχεια το παράθυρο αυξάνεται εκθετικά
  - έως ένα κατώφλι και μετά αυξάνεται γραμμικά



Φιλοσοφία:

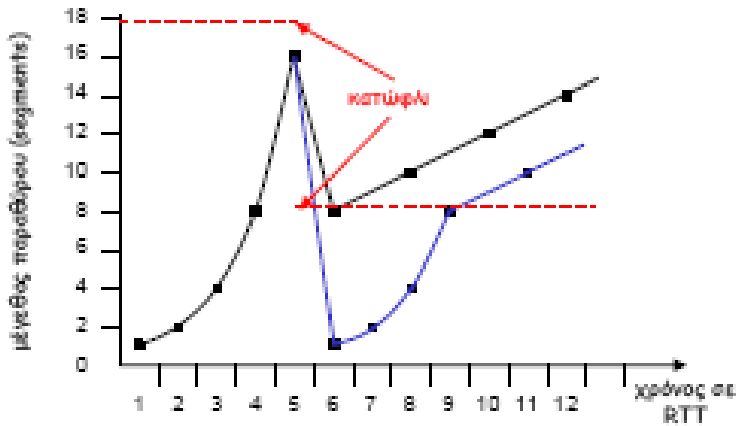
- 3 διπλότυπα ACK είναι ένδειξη ότι το δίκτυο είναι σε θέση να παραδώσει μερικά segments
- ένα timeout πριν από 3 διπλότυπα ACK είναι πιο "ανησυχητικό"

Διευκρίνιση (συνέχεια)

- η αύξηση μεταβάλλεται από εκθετική σε γραμμική όταν
- η τιμή του **CongWin** φθάσει το μισό της τιμής πριν από το timeout

Υλοποίηση:

- Μεταβλητή **Threshold**
- Όταν συμβεί ένα γεγονός απώλειας, η μεταβλητή **Threshold** παίρνει τιμή ίση με το μισό της τιμής που είχε το **CongWin** ακριβώς πριν από το γεγονός



Σύνοψη: Έλεγχος συμφόρησης στο TCP

- Όταν **CongWin < Threshold**, ο αποστολέας βρίσκεται στη φάση αργής εκκίνησης και **CongWin** αυξάνεται εκθετικά
- Όταν **CongWin > Threshold**, ο αποστολέας βρίσκεται στη φάση αποφυγής συμφόρησης και **CongWin** αυξάνεται γραμμικά
- Όταν ληφθούν τρία διπλότυπα ACK,

  Threshold ← **CongWin/2** και **CongWin** ← **Threshold**
- Όταν συμβεί ένα timeout,

  Threshold ← **CongWin/2** και **CongWin** ← 1 MSS

Μέσο throughput σύνδεσης TCP

- αγνοούμε τις φάσεις αργής εκκίνησης μετά από timeout (μικρή διάρκεια)
- έστω W η τιμή του `CongWin` όταν συμβαίνει ένα γεγονός απώλειας
- θεωρούμε ότι W ≈ σταθ., RTT ≈ σταθ.
- ο ρυθμός της σύνδεσης αυξάνεται γραμμικά από W/2RTT σε W/RTT (αύξηση κατά MSS/RTT κάθε RTT)

- μέσο throughput σύνδεσης = $\frac{0.75W}{RTT}$