MINI IPERF TUTORIAL

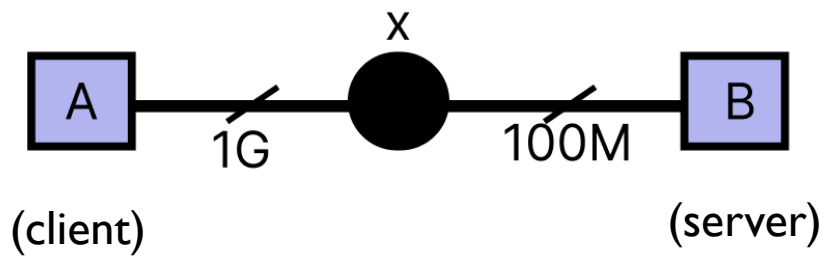
SPRING 22 TAS: MANOS LAKIOTAKIS, ELEFTHERIA PLEVRIDI

INTRODUCTION

- In this assignment you will create **your own network measuring tool → mini iperf** (or you may find a fancier name for your implementation)
- **Client – Server model**
- 2 communication channels suggested
  - TCP for communication
  - UDP for experiments

ΠΟΥ ΠΡΕΠΕΙ ΝΑ ΓΙΝΟΥΝ ΟΙ ΜΕΤΡΗΣΕΙΣ;

- Μετράμε end to end!
- Για το UDP → **Δεν έχει νόημα** να μετρήσω στον sender! (client)
- Στο TCP → Μπορώ να μετρήσω όπου θέλω (λόγω της ρύθμισης των μηχανισμών)
  - Θυμηθείτε: Πού είναι τα retransmissions/congestion window?



- **Οι μετρήσεις για τα πειράματα γίνονται όλες στον server!**
  - **Jitter**
  - **Goodput**
  - **Throughput**

HOW TO BEGIN

- Ξεκινάμε με το ένα main πρόγραμμα που να παίρνει από το command line τα κατάλληλα arguments !

→ Τηρώ το API!

- Θα χρειαστώ 2 διαφορετικά αρχεία 1 για server & 1 για client.

1. TCP CHANNEL

- **Signaling**
  - Start experiment
  - Stop experiment
  - Experiment exited
- **Reporting**
  - Να στέλνω περιοδικά στατιστικά στον client?
  - Να τα στέλνω όλα μαζί στο τέλος αφου τελειώσει το πείραμα;
  - Να σκεφτώ πως θα αποθηκεύω σε κάθε transmission τα αποτελέσματα

Γενικά το TCP κανάλι είναι ένα εργαλείο για να μας βοηθήσει να κάνουμε τα πειράματά μας!

Go wild! Do whatever you like.

ΓΙΑΤΙ ΧΡΕΙΑΖΟΜΑΙ ΤΟ TCP CONNECTION;

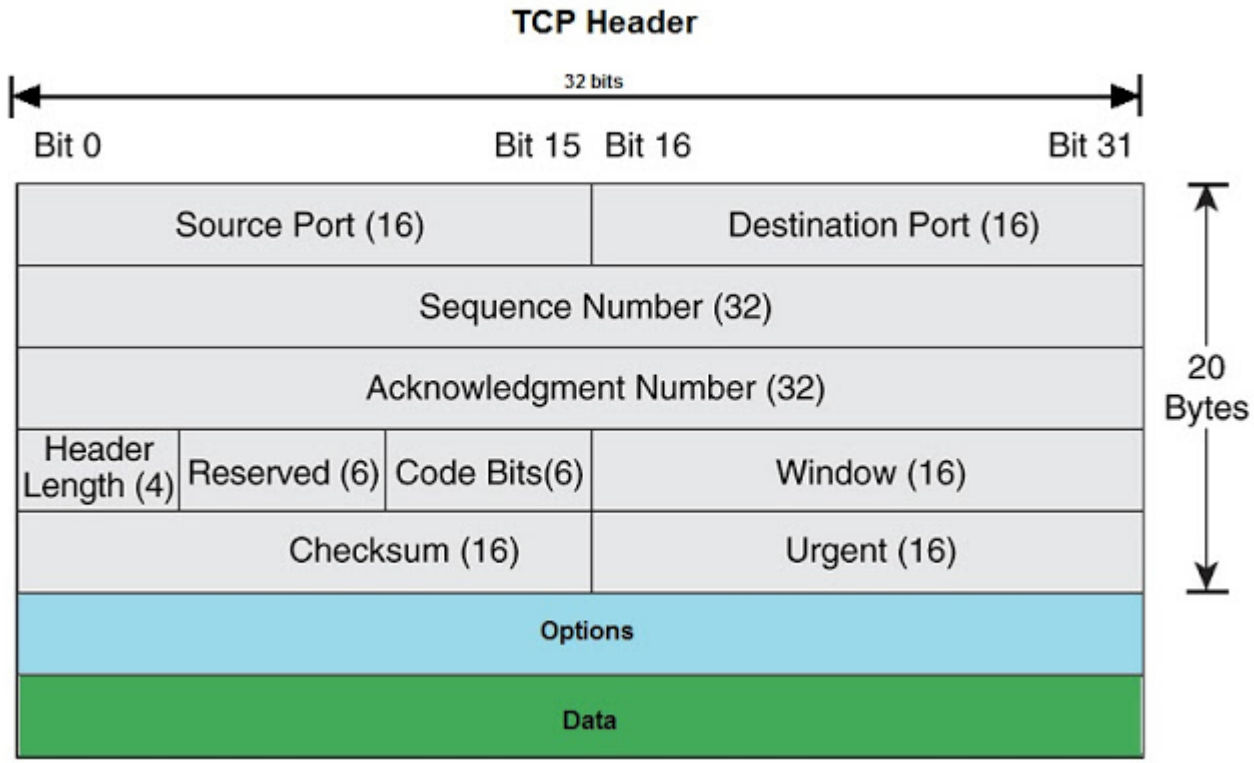
- Port
  - Τα 2 end points πρέπει να συμφωνήσουν σε ποια πόρτα θα μιλήσουν
  - Συνήθως ο server επιλέγει, και ενημερώνει τον client.
- Packet length
  - Ο client αποφασίζει ότι θέλει ένα πείραμα, με συγκεκριμένο packet length, και ενημερώνει τον server.
- Pararell streams (user param)
  - Το αποφασίζει ο client και ενημερώνει τον server
- Το TCP είναι stream! Δεν δουλεύει με πακέτα.
  - Αν θέλω συγκεκριμένα πακέτα χρησιμοποιώ UDP with fixed size.

ΤΙ ΚΑΝΩ;

- Δημιουργώ ένα δικό μου πρωτόκολλο επικοινωνίας που να είναι **self contained!**
- Από «μπροστά» θα βάλω κάποια **meta data**
- Χρειάζομαι ένα είδους **constant size header** για να μπορεί η εφαρμογή μου να κάνει **handle** τα **streams!**
  - Ετσι ο **receiver (server)** θα μπορεί να πει: «Όσο τα **x bytes** δεν είναι αυτά που θέλω, **discard**»
  - Όταν το βρει, και αφου θα ξέρει το **length** (πχ 1024) κάνει πολλαπλές φορές **recv** μέχρι να τα λάβει όλα.
  - Έπειτα μολις τα λάβει τα διαχειρίζεται.!
- Προσοχή, το –**l size** → **UDP packet size in bytes!**

HEADER SIZE?

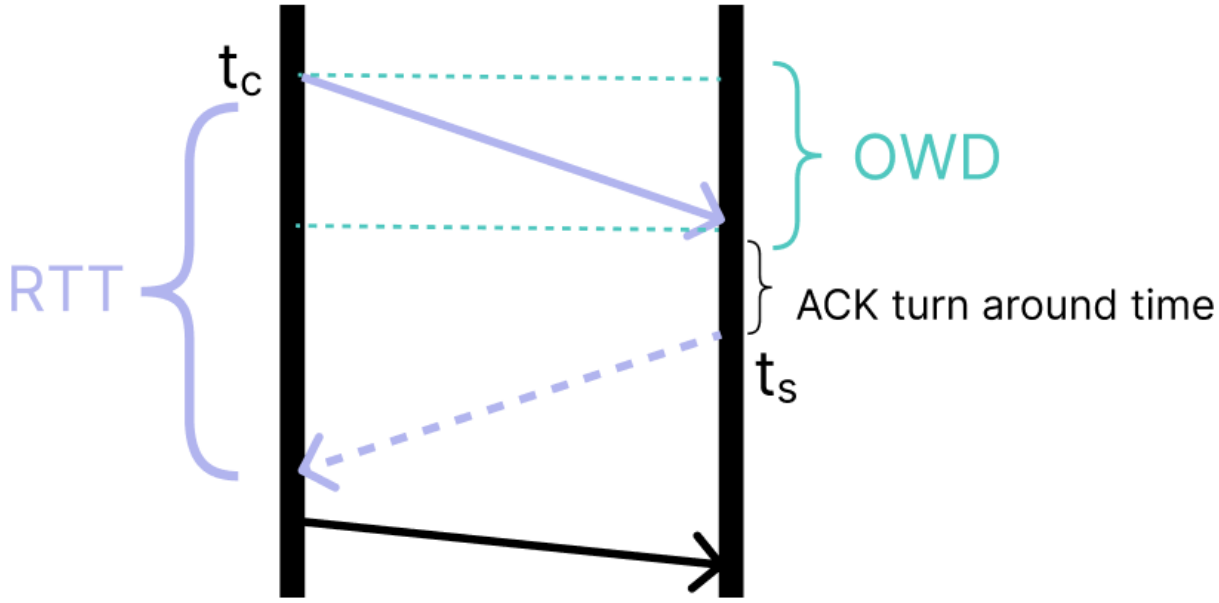
- Τα περισσότερα IP πακέτα εχουν 20-byte header (with max of 60 bytes)
- Τα TCP segments έχουν κ αυτά συνήθως 20 bytes στον header και max 60 bytes
- ➔Βλέποντάς τα μαζί, most TCP/IP datagram have 40 bytes of header data.
- Όταν φτιάχνουμε segments στο TCP πρέπει να υπάρχει χώρος για αυτούς τους headers! Αλλιώς θα γίνει exceed το MTU (<= 1500) και θα οδηγηθούμε σε fragmentation.
  - Source port – 16 bits
  - Destination port – 16 bits
  - Sequence identifier – 32 bits
  - Ack identifier – 32 bits



Προσοχή στις μεταβλητές → unint32
Και με τι χρόνους παίζετε! Πχ nano seconds!

ONE WAY DELAY

- OWD: είναι ο χρόνος που κάνει ένα πακέτο να φτάσει από την πηγή στον προορισμό.
- RTT: Round Trip Time
- Λύσεις;
  - Ατομικά ρολόγια (χάνουν 1sec /100.000.000 χρόνια) ακρίβεια ~ 1 pico sec
  - NTP protocol → allows sync of system clocks (ntp) ακρίβεια ~ 1-10 msec
  - PTP precision time → used to sync clocks in a computer network .
    Ακρίβεια ~ 20 μsec
  - RTT/2
  - Μέσος όρος
  - Μπορώ να δω το standar deviation, αν κάτι ξεφέυγει δεν το υπολογίζω



Fun fact!

Their newest atomic clock is predicted to become inaccurate by an amount of 1.6 seconds of time after running for a total of 10^{18} seconds—or, in other words, it loses **one full second** over the course of about 50.8 billion years.

2. UDP CHANNEL

- Το UDP έχει αυτοδύναμα πακέτα, δεν σπάει τίποτα στην μέση
- Connectionless → όχι accept , όχι connect, Απλά στέλνω δεδομένα
- Recvfrom() και sendto()
- Η sendto() Δεν μπλοκάρει αφου δεν περιμένει ack, όμως κάνει Block τον ρυθμό που στέλνει δεδομένα!

- →With setsockopt() μπορώ να αλλάξω τα default settings των sockets

UDP TRAFFIC 1

- **Το packet loss, πώς θα το καταλάβει ο receiver ?**
  - → Το κάθε πακέτο έχει πάνω του **ένα sequence number.**
  - Για το throughput, εκτος από το sequence number, δεν χρειάζομαι κάτι άλλο.

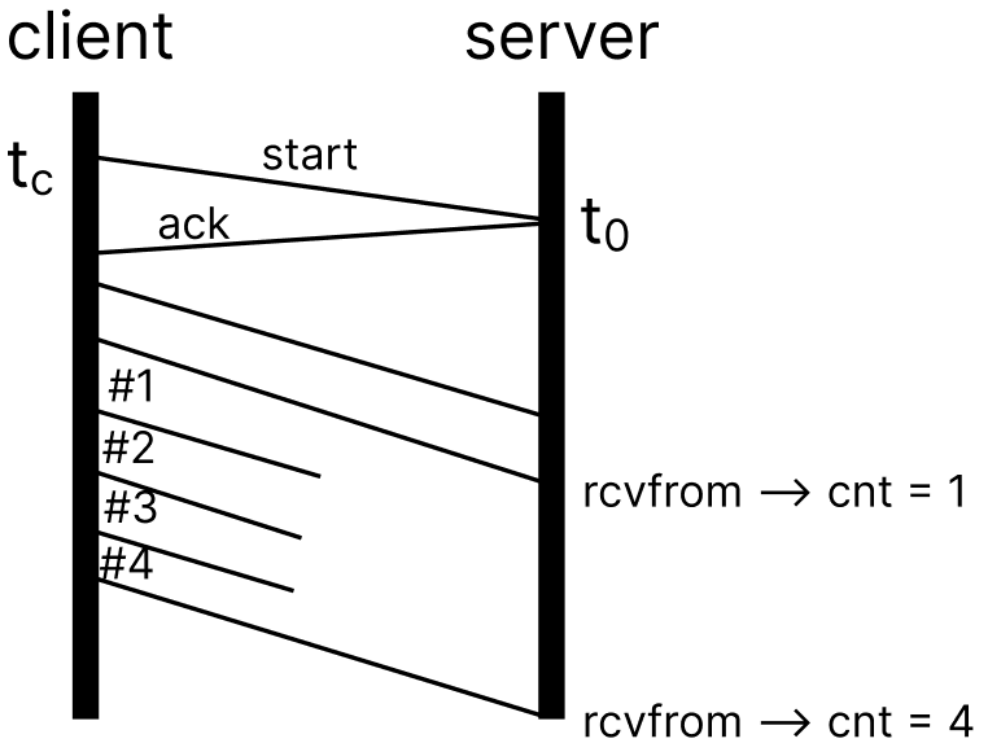
? **Πως θα γίνει generate το throughput όμως;;** → Ο χρήστης ορίζει το packet length + throughput.

? Ερώτηση: **ΠΩΣ παράγω συγκεκριμένο throughput μέσω μιας UDP πόρτας;**

UDP PACKET

- Sequence number (32bits)
  - Και αν πάνε εκτός σειράς; Out of order ?? → **Το διαχειρίζομαι σαν packet loss**!
  - → Ετσι ο server μπορεί να κάνει estimate το packet loss.
- **Το υπόλοιπο πακέτο το γεμίζω με ότι πληροφορία θέλω! (+κάποια δεδομένα)**
- Pre allocated buffers που θα έχουν συγκεκριμένο content/value

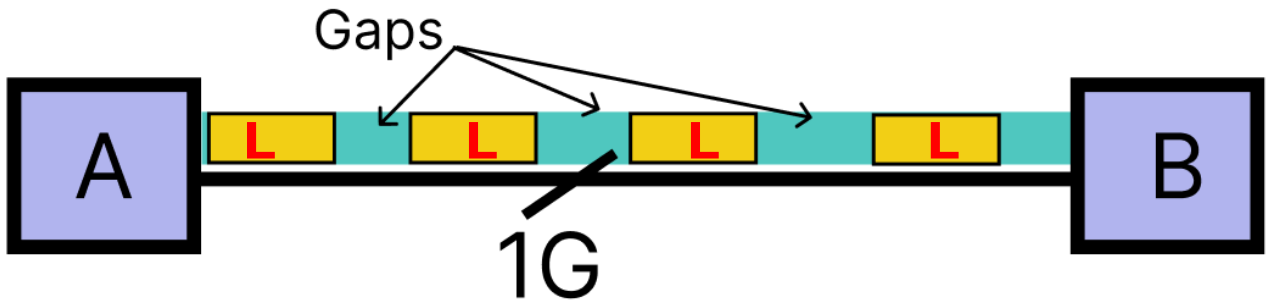
PACKET LOSS



- Αρα θα έχουν χαθεί 2 !

UDP TRAFFIC - THROTTLING

- Δεν έχω καμμία επιρροή πάνω στην ζεύξη!
- **Προσπαθώ έμμεσα να επηρεάσω το long term throughput!**
- **Εισαγωγή delays!**



- Τι γνωρίζω;;
  - Packet size
  - Link speed
  - Και φυσικά ξέρω πόσο throughput θέλω να πετύχω!

1 Mbit → 1 χιλιοστό του GB

THROTTLING

- Τι είναι; Η διαδικασία του να **περιορίζουμε το bandwidth** που μπορούν να έχουν οι χρήστες μας !
- **Αρχικά ΜΗΝ το υλοποιήσετε** !
- Αφήστε το πρόγραμμα να τρέχει και παράλληλα τρέξτε IFTOP (bandwidth monitoring tool)
- Δείτε αν είναι ίδια/παρόμοια τα αποτελέσματα στο throughput, αν όχι, κάτι θα έχει πάει λάθος με τον υπολογισμό του!
- Έπειτα προσπαθώ για το throttling
- Θυμάμαι→ Delays ανάμεσα στο packet size
- Μετά τσεκάρω και μετράω και βάζοντας το jitter, packet loss.
- **ΠΩΣ θα το ελεγξω ;; → Busy wait/ spinning στον client!**

MEASUREMENTS

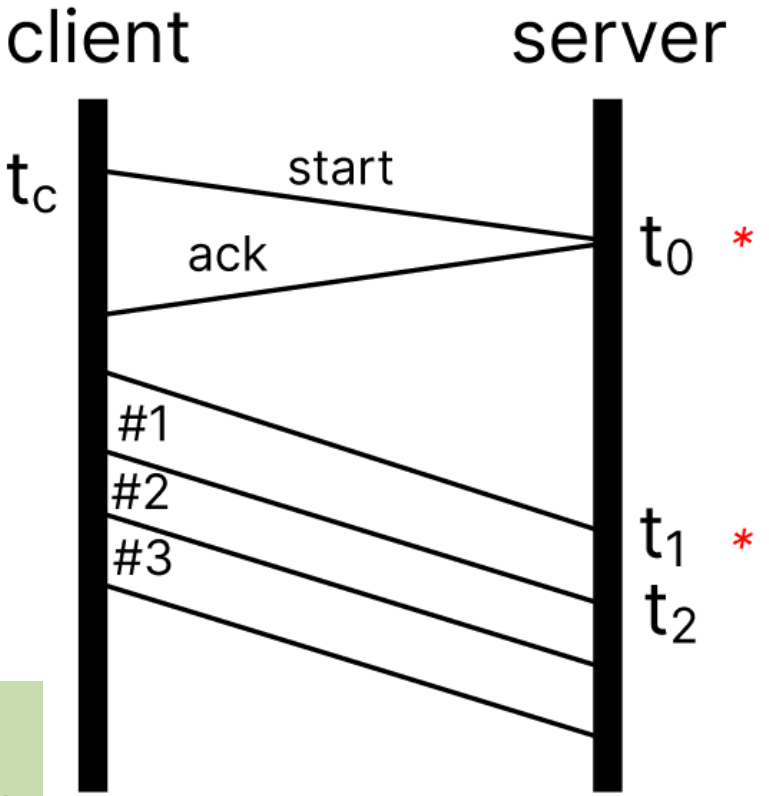
- Ως εδώ εχουμε καταφέρει να φτιάξουμε το throughput... **Τώρα πως θα το μετρήσουμε;;**
- Throughput = δεδομένα ανα second.
- Έχω την clock_get_time() = μονοτονικό ρολόι **Πού πρέπει να μετρήσω;;**
- **Αν χρησιμοποιήσω timers κλπ, → threading**

Η recvfrom() είναι blocking και περιμένει το 1ο πακέτο.

→ ΑΡΑ ξεκινάω να μετράω από το 2ο + πακέτο!

Για να μην χρησιμοποιήσω 100% την CPU μου όταν μετράω, μπορώ να εναλλάξω μεταξύ: poll, sleep, poll sleep

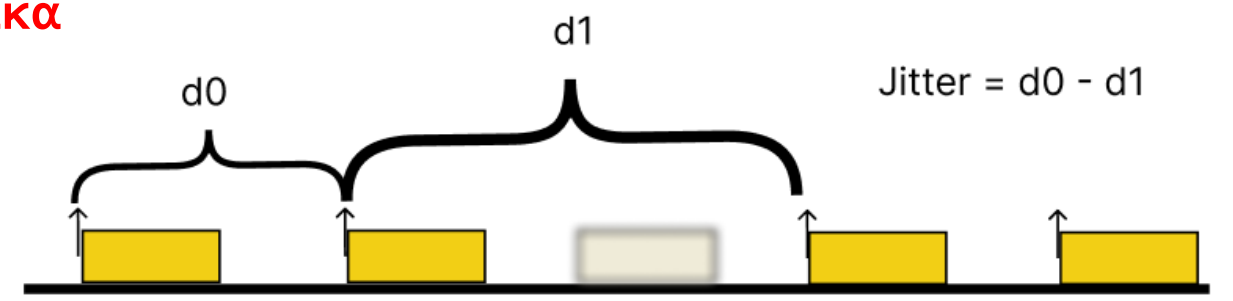
→ Busy waiting/spinning



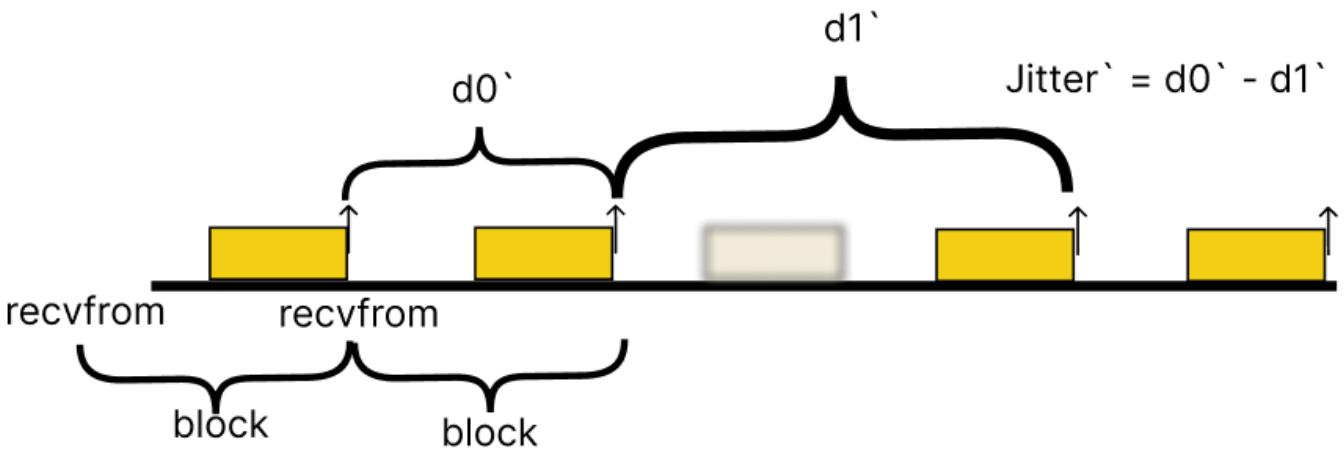
- Αν ξέρω πόσος είναι αυτός ο χρόνος, και ξέρω και πόσα είναι τα δεδομένα L

JITTER = Η ΔΙΑΦΟΡΑ ΤΩΝ INTERARRIVALS

Ιδανικά



t0= 0ms **t1 = 20 ms**



- Ιδανικό jitter θα είχαμε αν
  - t1 – t0 = 20ms
  - t2 – t1 = 20 ms
- Η receivfrom() επιστρέφει όταν πάρει ένα frame.
- Τι γίνεται αν χαθεί ένα πακέτο; Το jitter το μετράω αν χαθεί πακέτο ή πρέπει να κάνω reset counters κλπ;;
- **➔UP TO YOU!**

SEND AND RECEIVE TIPS – ΙΣΧΥΟΥΝ ΚΑΙ ΓΙΑ ΤΑ 2 KINDS OF SOCKETS

- Χρησιμοποιώ **send() & receive()** όχι **write & read**
- Η **send** εγγυάται ότι, ό,τι μου επέστρεψε, έχει πάει σωστά, τα άλλα είναι ευθύνη του προγραμματιστή!
- Πάντα να τσεκάρω τι επιστρέφει η **send()**
- Η **send** μπλοκάρει? ΝΑΙ!
  - "περιμένει **ack**" (αν και στην πραγματικότητα οι **buffers** του **kernel** το περιμένουν)
  - Από τα πιο κάτω **layers** που έχουν **buffering** και έχουν μπλοκάρει αυτά πρώτα.
- Για να πάρω δεδομένα → **receive()** – είναι **Blocking!**
- Η **receive** δεν κάνει κάποιο **allocation**, πρέπει να το κάνω εγώ.
- Αν δεν θέλεω να κάνω **block** σε κάποιο **socket** (πχ αν δεν υπάρχουν δεδομένα) → **use poll()**

THREADS 1

- Why threads?
- → παράλληλα streams
- Για να κάνω accept ένα καινούριο connection χρησιμοποιώ την accept(int sock, struct sockaddr, socklen)..
- Η accept είναι Blocking!! Περιμένει μέχρι κάποιος να κάνει connect.
- Για να μπορεί ο server να διαχειριστεί άλλα requests και εγώ σαν client να συνεχίσω να μιλάω εκεί που ήμουν,
  - Το OS, δίνει στην accept ένα καινούριο sd και κάνει ένα Mapping μεταξύ port & addr
  - Από το struct sockaddr_in παίρνω πληροφορία για πόρτες κ διευθύνσεις.
- Προσοχή να γίνονται όλα reset!! – **use memset**

THREADS 2

- Τι είναι το **thread?** Μια **lightweight** διεργασία.
- Τα **threads** του ίδιου **process** μπορούν να συγχρονίζονται .
- Εμείς το **thread** που καλούμαστε να φτιάξουμε, είναι όταν έρθει ένα καινούριο **connection**
- Θα είναι το **socket descriptor** που μας επιστρέφει η **accept()**, ώστε να μιλήσω με τον **client.**
- Για να μπορεί ο **server** να εξυπηρετήσει από την **accept()** και μετά, μπορώ να τον βάλω σε συνάρτηση,
  Να αφήσω ένα **thread** να χειριστεί τον **client** που μόλις ήρθε και
  το **main thread** πάει πάλι να κάνει καινούρια **connect.**

GENERAL TIPS

- Ο ένας μπορεί να ξεκινήσει με το **handshake – tcp** και ο άλλος με το **udp**
- Μην τα υλοποιήσετε όλα μαζί.
- Φτιάξτε συναρτήσεις για **reusability** και **clean code**
- Προσοχή στις μονάδες ! (**bits/bytes/sec/nsec/uint32/...**)
- Φτιάξτε προσεκτικά τον **parser** και **be aware** από τις μονάδες μέτρησης!!

RECAP

1. Main programm for reading arguments
2. TCP communication channel
    1. Tcp costum header
3. UDP communication channel
    1. Best way to calculate OWD
    2. Throttling. (Πως θα τηρήσω το bandwidth που παίρνω σαν Input) Που θα το υλοποιήσω;
    3. How to measure bandwidth
    4. How to measure packet loss / jitter (τι κάνω σε περιπτώσεις που χάνονται πακέτα)
    5. Λαμβάνω υποψιν headers των άλλων επιπέδων.
4. Parallel streams

GOOD LUCK!!!