Oggi vediamo il primo laboratorio. Il primo laboratorio si chiama Ping-Pong, e lo scopo è di mettere in comunicazione due processi (possibilmente su macchine diverse).   
Dovremo utilizzare il socket di tipo STREAM (quindi servirà usare la system call corretta). Ping funziona da client e pong funziona come server. Ping dovrà creare un socket di tipo stream e effettuare l’operazione di connessione (connect()), avviando la three way handshake (verrà automaticamente segnata la porta effimera per il client ping).  
Il server pong dovrà definire un primo socket ed effettuare la bind, poi la listen e infine la accept (si ricorda che il file descriptor ritornato da accept è quello da usare).

A questo punto l’idea è di fare delle misure di prestazioni sul canale di comunicazione: prima di tutto ci si mette d’accordo sulle modalità di effettuare l’esperimento. La richiesta sarà una stringa di caratteri che indica che cosa il client vorrebbe fare: a questo punto il server risponderà con un’altra stringa di caratteri (che è l’OK) e quando il client sa che il server è OK questo manda il primo messaggio come una sequenza di byte arbitraria, dopodiché si mette in attesa della risposta. Appena ricevuto il primo messaggio il server lo rinvia tale quale (Ping Pong).

La cosa che ci interessa è misurare il tempo che passa (round trip time) tra trasmissione e risposta.

I due programmi saranno scritti in C. Il client Ping è stato scritto per primo (bisogna completare i sorgenti). Il primo file su cui lavorare si chiamerà tcp\_ping.c.

L’analisi sarà di tipo statistico (perché i valori non sono esattamente ripetibili). Sarà quindi da calcolare il valore medio e mediano di rtt completando il file statistics.c.

Per provare la connessione, c’è il server del dipartimento, che ha l’indirizzo “webdev.dibris.unige.ite” che è in ascolto sulla porta 1491, perché questa porta è quasi libera (è assegnata a un protocollo ma che non è attualmente in uso).  
L’idea è quindi di lanciare il programma ping e dare come parametri il numero di byte che devono essere inviati e ricevuti, il numero di ripetizioni e l’indirizzo ip e il numero di porta del server a cui connettersi.

Dal punto di vista applicativo i nomi vengono risolti in un modo piuttosto complicato: la funzione che risolve gli indirizzi è infatti getaddrinfo() -> man gettadrinfo.

Le statistiche diventano significative dopo almeno 30 (ma anche centinaia) di ripetizioni. Alla fine l’output sarà di tipo testuale e in esso verranno stampate le statistiche con valore medio e valore mediano (più un istogramma della distribuzione della probabilità dei valori di rtt). Per calcolare il valore mediano servirà ordinare il vettore dei dati misurati e prendere il valore che ha indice metà della lunghezza del vettore (quindi conviene fare un numero di esperimenti dispari).

Cosa succede se faccio l’esperimento sulla stessa macchina senza la rete? In tal caso bisogna mandare in esecuzione entrambi i processi sullo stesso host, in tal caso non bisogna dare un indirizzo della rete, ma bisogna usare l’indirizzo “localhost”: in questo caso un sacco dei sistemi di trasmissione vengono bypassate e anche se questa è un’opzione corretta per fare debugging, non va bene per fare test come su una rete vera, perché i round trip time saranno troppo piccoli.

Se volessimo testare la rete per conto nostro senza usare l’indirizzo della rete universitaria, potremmo usare due macchine poste vicine in modo che siano connesse allo stesso access point; in tal caso l’indirizzo da usare sarà un indirizzo privato. A tal proposito si può usare il comando ifconfig.

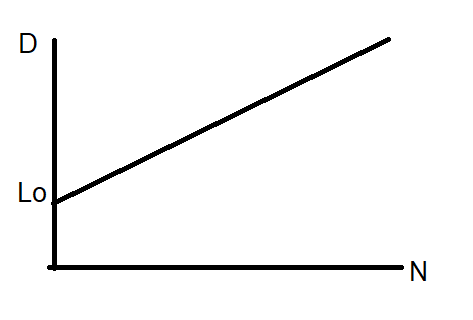
Ci sono due modi diversi di vedere la velocità di una rete. Un modo di intendere la velocità della rete viene espresso sotto il nome di banda di comunicazione: consiste nel numero di bit o di byte trasmessi sulla rete per unità di tempo.   
Se faccio degli esperimenti del genere di questo laboratorio è abbastanza ovvio pensare che la banda possa migliorare aumentando il numero di byte che vengono trasmessi (perché ammortizza il numero di byte che vengono usati in più per i vari header, poiché ricopre una percentuale maggiore del datagramma). Per la banda si prende sempre quindi il valore massimo che si riesce a misurare (tale massimo si otterrà con messaggi grandi).  
L’altro modo per vedere la velocità della rete è la latenza. Tale latenza è il tempo necessario per recapitare un messaggio a destinazione. In tal caso la minima latenza (e quindi la maggior velocità in quel senso) si otterrà con messaggi più piccoli. La latenza è difficile da misurare remotamente: se server e client sono sullo stesso host, essi useranno lo stesso orologio, dunque, anche se sbagliato il valore della latenza sarà preciso. Non si può avere la certezza che due macchine abbiano lo stesso fuso orario /l’orologio alla stessa ora, dunque, il modo in cui si misura la latenza è calcolando il tempo fino all’arrivo del messaggio di risposta e dividendolo per due (la latenza è quindi metà del RTT -> non è una misura precisissima, perché l’andata potrebbe prendere più hop del ritorno o viceversa, ma è il meglio che si può fare).

Potremmo avere delle situazioni in cui a livello di rete locale la connessione ha una banda piccola ma con una latenza bassa e una connessione con una rete molto lontana in cui la banda è larga ma la latenza è grande a sua volta.   
Se vogliamo conoscere la velocità della rete dobbiamo calcolare sia banda che latenza.  
Il numero di byte per unità di tempo è detta Throughput: la banda è il valore massimo che può assumere il Throughput.

La formula per comunicare il ritardo per l’arrivo di un messaggio usando il modello banda-latenza è:

D = Lo + N / B, dove B è la banda, Lo è la Latenza e N è il numero di byte che dobbiamo inviare.

Questa formula ci dice che quando il messaggio è molto piccolo, D tende a 0, idealmente se mando zero byte il tempo impiegato dal messaggio è puramente il valore della latenza Lo. Quando la banda diventa molto piccola, Lo diventa trascurabile perché N / B tende ad infinito e dunque tende ad infinito anche il tempo impiegato dal messaggio per arrivare. Quando il messaggio è piccolo dipende tutto dalla Latenza, quando il messaggio cresce dipende tutto dalla banda di comunicazione.



La crescita è lineare e il coefficiente angolare della retta è 1/B. I ritardi, quindi, crescono linearmente (in linea teorica, poi nella pratica ci sono un po’ di discostamenti) con il crescere del numero di byte da inviare.

Quali sono i trucchi per scoprire la latenza e la banda della nostra rete di comunicazione? Bisogna effettuare due esperimenti di ping pong, con due valori molto diversi di N (byte inviati). Si prende il ritardo (che è mezzo round trip time) e si calcola la latenza e la banda empiricamente, invertendo la formula e facendo un sistema lineare a due incognite:

{ Dmin = Lo + Nmin / B

{ Dmax = Lo + Nmax / B

Settimana prossima ci sarà da completare l’udp\_ping.c.

UDP è meno sicuro del TCP per quanto riguarda l’arrivo dei messaggi.  
UDP usa semplicemente il protocollo IP senza aggiungere niente, TCP implementa il controllo di flusso/di congestione ecc.

A proposito del protocollo di congestione, cosa succede quando uno o più router è sovraccarico? Le connessioni TCP riducono il traffico in uscita per permettere al router di liberarsi, ma le connessioni UDP non hanno alcun controllo di congestione, di conseguenza i datagrammi UDP in quel caso si limitano a prendersi tutto lo spazio disponibile (togliendo spazio al traffico TCP) -> dunque usare il protocollo TCP è efficiente se non c’è troppo traffico UDP nei router necessari per l’instradamento.

Per disegnare i grafici statistici è comoda la libreria Gnuplot (o Gniplot?)

---la bind lato server prende l’indirizzo della stessa macchina e la porta, che vengono specificati dal getaddrinfo---

La consegna è individuale: le persone nello stesso gruppo devono tutte consegnare la stessa cosa. Bisognerà consegnare anche una relazione.

Vediamo adesso un argomento relativo al protocollo TCP non coperto in precedenza, ossia la chiusura delle connessioni. La chiusura delle connessioni può essere effettuata in modi diversi (più o meno rapidi): tipicamente vengono usati gli altri flag del protocollo TCP (avevamo già visto il syn e l’ack). Per la chiusura delle connessioni si guardano il flag “fin” e il flag “ack”. Il flag fin indica la chiusura della connessione e può avvenire da parte di entrambi i partner (sia client che server possono dare inizio alla chiusura della connessione).  
Se un host ha mandato tutto, può inviare l’ultimo messaggio col flag fin a 1. Il destinatario che riceve il messaggio col flag fin risponde con un acknowledgement che ha attivi i flag fin e ack (se vuole chiudere la connessione a sua volta, se ha bisogno di inviare altre cose risponde con un ack e basta -> in quel caso il socket non si chiude, ma la connessione continua solo in un verso, perché tanto l’host di partenza non ha più nulla da inviare). Quando riceve la risposta fin/ack, per l’host di partenza la connessione è chiusa, però manda un ultimo messaggio “ack” che conferma al destinatario la chiusura della connessione.   
Questa è la versione lunga e assomiglia a un three way handshake ma al contrario. Ci sono modi più rapidi per interrompere la connessione. Il modo più veloce per staccare la connessione è usare il flag “rst”: se un host vuole chiudere la connessione di colpo, basta che mandi un datagramma con quel flag e la connessione viene chiusa (la differenza rispetto a fin è che mentre col fin ci si assicura che tutti i byte in giro arrivino a destinazione, col rst no).  
Rst può anche essere la risposta alla richiesta Syn. Se un server non vuole rispondere alla connessione, anziché rispondere con syn/ack risponde con rst (durante la three way handshake).

La comunicazione può anche essere chiusa quando non ci sono messaggi in corso per un po’ di tempo (ciò permette di risolvere la situazione in cui c’è un problema hardware, magari si stacca il filo, oppure se l’ultimo ack di chiusura non viene recapitato), tramite un timeout più lungo rispetto a quello di ritrasmissione.