## 2.1 - Informazioni generali

Consegna: Venerdì 18

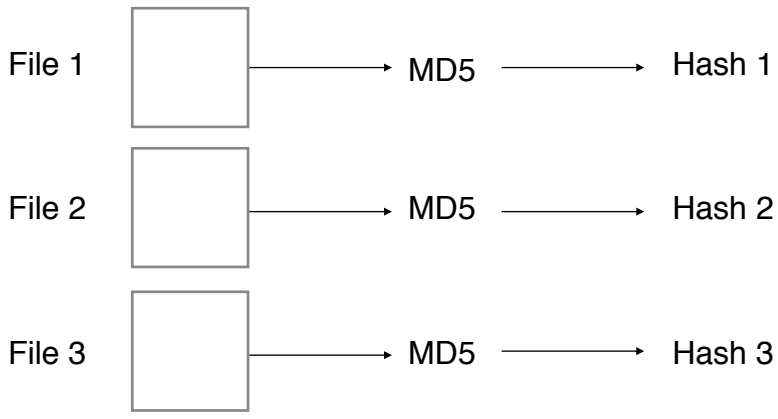
Materiale condiviso: nozioni teoriche (slide), manuale Python, manuale TCPdump (per monitorare i pacchetti che girano in rete, se qualcosa non va, guardare il traffico, mettere in modalità fullascii (?) e vedere se il formato del pacchetto che buttiamo fuori corrisponde).

Scriveremo l'indirizzo IP in modo particolare (lungo) in ASCII in modo da vederlo subito, con un protocollo vero non lo scriveremmo così, meglio esplicitarlo il più possibile (meglio lento ma documentato che veloce che non si capisce).

## 2.2 - Napster

Prima esperienza: directory centralizzata, deve esserci una directory da qualche parte che contiene le informazioni sui file, se io sono directory e gli altri i peer, ognuno di essi deve dirmi che file ha e ogni volta che gli serve un file viene a chiedere dove si trova. Quello che guida in questa esperienza è il titolo del file, non il contenuto. La directory manda indietro un identificativo dicendo quali sono tutti i peer che hanno il file cercato. Come faccio a capire se due soggetti diversi hanno cercato due file diversi ma con lo stesso nome? Per ogni file va fatta una signature, appunto del file, e verrà usato un md5.

MD5: se si ha un file molto lungo (insieme di stringhe molto lungo) quello che si può fare è costruire una funzione, funzione di hash, che passa da qualcosa di molto lungo a qualcosa di molto corto (da X byte a 16 byte), li trasforma in modo intelligente ovvero per una variazione di un bit devo riuscire ad avere il massimo numero di variazioni nel prodotto finale, deve essere un meccanismo ad altissima capacità di mescolazione.




Dimensione file >> dimensione hash

Per motivi di sicurezza quando metto un file ne calcolo l'hash e me lo tengo da qualche parte (essendo piccolo posso tenermelo in un dispositivo a parte). Per capire se il file è stato modificato calcolo il nuovo hash e lo confronto con quello salvato su dispositivo, se sono diversi è stato cambiato. MD5 è una delle implementazioni più famose di hash. Ci serve per identificare due file diversi per contenuto ma non per nome. Va fatto perché la directory non tiene il file ma l'MD5. Quando un peer vorrà "registrare" il suo file sulla directory le dirà: "questo file ha questo nome e questo MD5". Quando cerchiamo qualcosa nella directory questa ritorna tutti i nomi di file e tutti gli MD5. La directory ritorna gli indirizzi IP e la porta su cui ascoltano i peer che hanno fisicamente il file, se ci sono più soggetti col file dobbiamo decidere noi a quale chiedere, ne scegliamo uno e scarichiamo da quello.

Quest'anno qualunque formato di indirizzo dovrà essere sia v6 che v4 e la porta su cui ascoltare dovrà essere la stessa per entrambi.

Formato dell'indirizzo: 55 byte, ipv4: %03d.%03d.%03d.%03d (4 decimali, 15 byte). Pipe di separazione( | ), %04x (formato classico della printf per scrivere), ce ne sono 8 (ipv6) -> %04x:%04x:%04x:%04x:%04x:%04x:%04x:%04x (8 esadecimali, 39 byte);

L'indirizzo complessivo sarà ipv4 |(pipe 1 byte?) ipv6.
Noi li rappresentiamo così in modo che siano chiari. Le funzioni per convertirli in questo modo possiamo rifarle o usarle già fatte. Di solito gli errori sono sugli indirizzi, MSB e LSB (little o big endian, in realtà in tutti i linguaggi ci sono funzioni che convertono dal formato network a quello SO, vanno usati! Se non usiamo un indirizzo con swap nei vari byte. Col tcpdump lo si vede).

### 2.2.1 - Funzioni

**Login**: un peer si deve classificare alla directory perché se poi vuole trasmettergli delle cose essa deve capire che è lui. Partiamo da un certo IP (quello di 55 byte, potrebbe essere o il v4 o il v6, dobbiamo usare una funzione random per decidere quale va, può cambiare ogni volta, alla dimostrazione va dimostrato col tcpdump che una volta va col v4 e una col v6), porta random. Il peer si interconnette alla directory, che quindi avrà un IP directory su una certa porta a 5B (3000, la prendiamo > 1024 perché se no si hanno problemi per alcuni SO). A questo punto dal peer verso la directory noi facciamo la

funzione di logi (4 byte), metto poi un . per dire che è concatenato, l'indirizzo IP di 55 byte e poi la porta (16 byte, numero che va da 0 a 65536). Ritorna un ack (4 byte) e un sessionID (oggetto che dice: "ti memorizzerò con questo identificativo" 16 byte messi a caso, random di numeri e lettere maiuscole).

**UP di un file:** da peer a directory, 4 byte per il nome della funzione.SessionID(16 byte).FileMD5(16 byte).nome del file(100 byte, usare maiuscole e/o minuscole e/o numeri, no lettere accentate o cose strane, il . va bene); Da directory a peer rispondiamo con un numero di 3 byte che rappresenta il numero di file con lo stesso MD5 presenti nella directory (così scopriamo se quel file l'abbiamo messo solo noi o l'hanno messo anche altri), se non si riceve l'ack ahia.

**Rimozione:** per il format vedere spec, qui il soggetto è il peer, sta dicendo che ha rimosso il file alla directory, a questo punto la directory deve mandare l'ack con il numero di file con lo stesso md5 dopo la rimozione del nostro. Se rimuoviamo qualcosa che non c'era il ritorno sarà 999, ci serve per capire cosa sta succedendo.

(La differenza tra directory e peer è che la directory dovrà salvarsi in memoria i dati, mysql, file di testo, ecc ecc. Scelta implementativa)
**Ricerca:** vedere spec, meglio farla il più semplice possibile basandosi sul nome del file. Per quanto riguarda la risposta è abbastanza complesso, numero di oggetti con quel nome ma con md5 differenti (3B), per ognuno di questi { itero per ogni cosa che metto nella graffa: per ognuno scrivo l'md5, il nome del file e il numero di copie che ho di quello, per ognuno di questi devo dire chi ce l'ha utilizzando un altro indice { (j), IP e porta di chi ce l'ha. Nello scrivere i nomi di file meglio stare dentro i 20B.

**Logout:** momento in cui un peer si spegne, è molto importante farlo, così gli altri smettono di cercarmi. Quando uno esce lo deve comunicare. Se no danni gravi. Restituisce i file tolti dalla directory in risposta all'uscita del peer (quelli che aveva il peer).

(Va creato un meccanismo semplice, non per forza bello, per far vedere cosa è cambiato ecc).
**Download:** funziona da un peer verso un altro peer, niente sessionID. Il file viene diviso in tanti pezzettini (chunk), quindi ritorna il numero di questi chunk (6B), poi itera tornando il chunk i coi suoi dati. I chunk possono essere di lunghezze diverse. Cosa trasferiamo?

Meglio immagini jpeg così capiamo subito se vanno bene oppure no. Inoltre il peer parla con la directory e gli dice che ha fatto il download di un certo file, così la directory ne tiene traccia. Risponde col numero di download di quel file. La registrazione del file è a carico del peer, non viene fatta qui come effetto collaterale.

Assegnazione IP: v4 numero gruppo.numero utente./16
fc00/7 ipv6 puntati -> fc00::numerogruppo::numeroutente/64
Bisogna collegarsi all'access point e farsi assegnare l'ip.