UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Dipartimento di Informatica
Corso di laurea in Informatica

Analisi e sperimentazione di sistemi distribuiti basati sul protocollo BitTorrent

RELATORE: Dott. Andrea Trentini

CORRELATORE: Prof. Mattia Monga

TESI DI LAUREA DI:

Davide Bruschi

691122

Anno Accademico 2012/2013

Indice

1 Introduzione

1.1 Contesto

In molte situazioni, i classici sistemi di comunicazione client-server, presentano significative limitazioni dovute tipicamente all'uso di banda (in casi di distribuzione di files) e alla centralizzazione dei servizi[1]. Il protocollo BitTorrent possiede caratteristiche particolarmente interessanti che permettono di superare queste problematiche permettendo di realizzare sistemi più efficienti e non centralizzati. Di seguito sono riportati alcuni esempi di scenari d'interesse.

1.1.1 Boot di rete distribuito

In molti ambienti (laboratori, università, ecc...), è necessario avere a disposizione un gran numero di computer con la stessa configurazione hardware e software. Solitamente questi pc sono connessi in LAN e vengono avviati trasferendo a ciascuno di essi il sistema operativo da caricare in RAM, opportunamente configurato con il software necessario. Per fare ciò viene effettuato un boot di rete tramite PXE, a cui segue un trasferimento, dal server verso ciascun client, del kernel e dell'immagine del file system da caricare in RAM (initrd o initramfs).

1.1.2 Streaming di video

La sempre più alta diffusione di dispositivi con display ad alta definizione, ha portato naturalmente alla relativa evoluzione dei contenuti video disponibili in rete che tendono ad essere sempre più di frequente in alta definizione (video su youtube, trailer cinematografici, piattaforme di streaming di film online, ecc...). Tali contenuti video in HD sono ovviamente più pesanti e richiedono una banda sempre maggiore per garantire una visione fluida e senza interruzioni. L'uso di sistemi di streaming peer-to-peer può ridurre significativamente il carico sul server portando a significativi miglioramenti nell'esperienza d'uso da parte dell'utente e a riduzione di costi da parte di chi offre i servizi.

[1] vedremo i dettagli in seguito

Ad esempio su Youtube, un minuto di video in HD720p pesa mediamente sui 10MB ne consegue che video di solo mezz'ora possono pesare anche più di 300MB (senza essere in FullHD!). Se estendiamo questa considerazione a tutti i vari servizi di video in streaming a pagamento e non, possiamo cogliere la portata del problema. Inoltre anche i video in 3D creano gli stessi problemi (anzi anche maggiori visto che di solito i film in 3D sono anche in HD). Ipotizziamo quindi di voler vedere un film della durata standard di 120 minuti, questo peserebbe come minimo 1,2GB (dipende dalla qualità delle immagini). Con una ADSL da 5Mbps (che corrisponde alla banda media delle ADSL in Italia[24]) servirebbero 32min per il download completo del film nelle condizioni migliori. La problematica è di particolare interesse data l'enorme diffusione di tv HD e 3D, SmartTv ecc..., tanto che la stessa comunità europea ha finanziato progetti che mirano ad ottimizzare la trasmissione di video senza dover necessariamente incrementare le potenzialità della rete fisica[8]. Questo perché ovviamente servizi con un grande bacino di utenti devono avere infrastrutture tali da permettere un upload adeguato e ciò comporta costi in termini di hardware, di aggiornamento delle infrastrutture per via dell'avanzare delle tecnologie disponibili (già si stanno diffondendo i primi televisori 4K) e di manutenzione.

1.1.3 File system decentralizzati

Il problema della censura online sta diventando, di recente, sempre più d'interesse. Questo perché spesso viene usata da governi e istituzioni per influenzare la diffusione di informazioni, nascondendo ciò che non viene ritenuto conveniente. Una valida soluzione potrebbe essere rappresentata dai file system distribuiti p2p dove la diffusione di specifici contenuti non può essere limitata dalla semplice chiusura o dall'oscuramento di un server o sito, dato che una copia dei files è presente su ciascun client connesso al suddetto file system. Esistono già implementazioni simili come, ad esempio, BitTorrent Sync[1] che si propone come alternativa a servizi centralizzati come ad esempio il famoso Dropbox[2]. Pur essendo ancora un servizio in fase beta e pur avendo delle problematiche ovvie (prima fra tutte la necessità o addirittura l'obbligo, per chi vuole usarlo, di avere svariati pc su cui mantenere i files sincronizzati), rende chiaro come sia possibile mantenere i propri files strettamente privati, dato

che non vengono memorizzati su server di terze parti al di fuori del nostro controllo. Il software usa inoltre un sistema di cifratura dei dati che vengono trasferiti tramite internet in modo tale da proteggere l'utente da possibili intercettazioni del traffico. In sostanza il funzionamento è il seguente:

- alla prima installazione, viene chiesto di specificare una cartella da mantenere sincronizzata
- una volta indicata la cartella, il programma genera una chiave che dovrà essere inserita su tutti i pc su cui vogliamo mantenere sincronizzati i dati; tale chiave verrà usata anche per la cifratura dei dati durante il trasferimento
- su ogni successivo dispositivo, basterà installare il software e indicare la chiave sopracitata per poter accedere ai dati della cartella e sincronizzarli con quelli locali

Questo richiede ovviamente che i pc siano accesi e connessi per garantire una sincronizzazione efficacie. A differenza di Dropbox e simili, qualora si volesse condividere la cartella, sarà necessario far installare il programma anche alla persona in questione e bisognerà fornirgli la chiave per permettere la sincronizzazione. E' ovviamente possibile creare più di una cartella quindi nel caso si può creare una nuova cartella e condividere solo quella. Così facendo dovremo condividere solo la chiave di quella cartella e non la chiave delle altre che resteranno personali. A differenza dei servizi stile Dropbox, paga tuttavia una minor semplicità d'uso e la necessità di avere, oltre alle opportune conoscenze per configurare il tutto, anche una quantità di hardware maggiore perché, non avendo un server esterno, starà a noi fornire lo spazio e le macchine su cui replicare e mantenere sincronizzati i dati; è chiaramente possibile usare anche pc di amici al posto di comprare hardware apposito, tuttavia questo comporta che l'amico possa accedere ai nostri files e richiede quindi un certo livello di fiducia. Pensando al problema della censura, segnalato sopra, BitTorrent Sync mostra le caratteristiche necessarie a realizzare un archivio dati non censurabile, anche se al momento è principalmente orientato verso un uso personale, piuttosto che condiviso fra molte persone. Per garantire poi l'impossibilità di censurare i contenuti, bisognerebbe prevedere un meccanismo che permetta solo di aggiungere file

e non di modificarli o addirittura rimuoverli. Essendo però il software ancora in fase beta, non è escluso che in future versioni vengano aggiunte tali funzionalità.

1.2 Obiettivi

Tra gli scenari elencati in precedenza, è stato preso in considerazione il boot distribuito di macchine diskless in rete locale. L'intenzione è quella di studiare il problema, realizzare (se effettivamente possibile) una nuova procedura, basata su BitTorrent, che realizzi il suddetto boot di rete e, infine, verificare l'effettivo livello di miglioramento raggiunto. In rete è stato trovato un lavoro (Moobi)[17] molto simile di cui però non è stata trovata alcuna implementazione e che presentava una serie di problemi che ne avrebbero reso problematica la distribuzione (come ad esempio uno scarso supporto HW). Col lavoro presentato di seguito, si è cercato di realizzare uno strumento pronto all'uso e con il massimo livello di compatibilità HW e SW possibile. L'idea è di usare BitTorrent, per trasferire dal server ai client, il kernel e l'immagine del filesystem da caricare in RAM, riducendo significativamente l'impatto sul server e velocizzando in generale l'intera procedura rispetto al tempo impiegato con il classico boot tramite tftp. La scelta di questo specifico scenario è motivata in primis dall'esigenza di migliorare la procedura di boot utilizzata durante gli esami, nel laboratorio SiLab, presso il Dipartimento di Informatica dell'Università degli studi di Milano ed in secondo luogo dal fatto che si tratta di una problematica molto comune nel mondo IT e quindi di particolare interesse.

1.3 Soluzione tipica

Il sistema più comunemente utilizzato, consiste nell'eseguire un boot tramite PXE. In questa fase, le macchine, prima di tutto ottengono l'indirizzo IP tramite DHCP; subito dopo iniziano il trasferimento del kernel e dell'immagine del file system del sistema operativo da caricare in RAM. Durante tale trasferimento, il server distribuisce i files necessari a ciascun client in rete con ovvie problematiche dovute alle limitazioni di banda della rete stessa. Al termine del download, viene caricato il ker-

nel appena scaricato e, dopo la dovuta decompressione, viene montata l'immagine del filesystem direttamente in RAM terminando così l'avvio delle macchine.

1.3.1 Il protocollo PXE (Introduzione)

Il protocollo PXE[9] (Preboot Execution Environment) è un protocollo sviluppato da Intel per garantire una serie di servizi di pre-boot uniforme, affidabile ed indipendente dal fornitore del software e dell'hardware sia dei client che del server permettendo così agli amministratori IT di personalizzare i tipi di macchine nelle loro reti senza il rischio di problematiche per quanto riguarda le configurazioni necessarie.

Il protocollo PXE consiste in pratica in una combinazione di altri due protocolli. Il primo è un'estensione del protocollo DHCP, il secondo è il protocollo TFTP.

1.3.2 Il protocollo DHCP esteso

Nello specifico si tratta di una combinazione di un'estensione del protocollo DHCP standard[25] (dovuta all'utilizzo di nuovi tag di opzione DHCP) e di una definizione di semplici transazioni di pacchetto che sfruttano il formato dei pacchetti DHCP per trasferire informazioni addizionali tra client e server.[2] Queste aggiunte sono giustificate dalla necessità di lasciare invariati i servizi DHCP esistenti.

1.3.3 Il protocollo TFTP

Il protocollo TFTP[18] (Trivial File Transfer Protocol), è un protocollo di trasferimento file, molto semplice con le funzionalità base del FTP[14]. Tra le sue caratteristiche è opportuno segnalare le seguenti:

- Usa l'UDP (porta 69) come protocollo di trasporto (a differenza del FTP che usa il TCP)
- Non supporta funzioni di autenticazione o cifratura

Il fatto di essere basato su UDP comporta la mancanza di quelle funzionalità proprie di TCP per garantire funzionalità avanzate sul trasferimento dei pacchetti.

[2]Dettagli sulle modifiche introdotte sono disponibili sulle specifiche intel del protocollo PXE[9]

La mancanza di funzioni di autenticazione o cifratura rende il suo utilizzo insicuro su internet e ne limita quindi l'utilizzo solo in reti LAN. Per quanto riguarda quest'ultima problematica, BitTorrent non rappresenta una soluzione completa anche se alcune sue caratteristiche rendono ragionevole ipotizzare procedure che ne permettano tale uso. Nello specifico, dato che nel file .torrent viene memorizzato l'hash del file da scaricare, si potrebbe garantirne la consistenza una volta terminato il download. La parte di autenticazione e cifratura andrebbero tuttavia implementate a parte e sarebbe necessario inviare il file torrent iniziale in modo sicuro per garantire che poi l'hash da verificare sia effettivamente quello giusto. Infine, nella procedura proposta in questa tesi, resterebbe il problema di dover trasferire in ogni caso un sistema minimale tramite tftp minando alla base tutto quello che potrebbe essere fatto tramite BitTorrent. Per questo motivo, la procedura proposta, è da considerarsi sicura solo in reti locali.

1.3.4 Il protocollo PXE (Descrizione)

Di seguito viene riportato il protocollo PXE[3] passo-passo

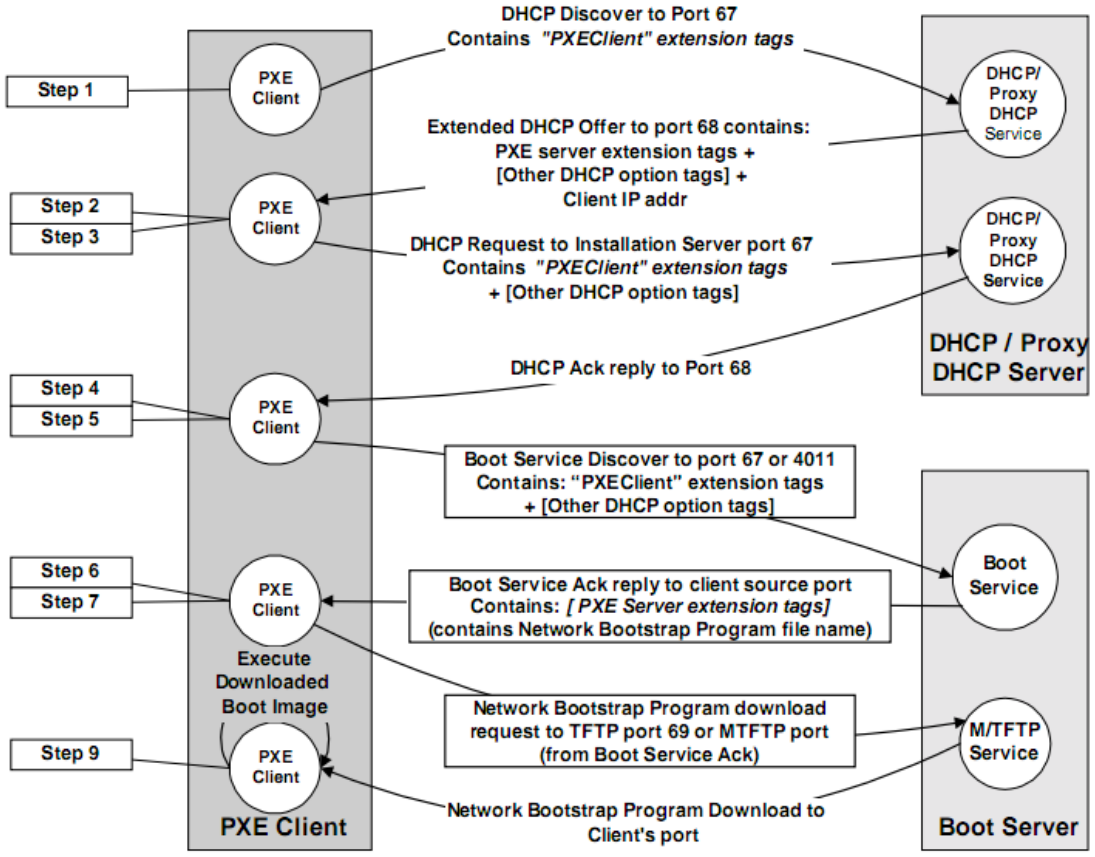

Figura 1: Il protocollo PXE passo-passo

[3]Per semplicità alcuni dettagli verranno omessi (nel caso si rimanda alle specifiche ufficiali Intel v2.1 da cui è tratta questa descrizione)

1. Il client invia in broadcast una DHCPDISCOVER sulla porta standard DHCP (67)

2. Il server DHCP risponde con una DHCPOFFER sulla porta standard di risposta (68)

3. Dalla DHCPOFFER ricevuta, il client ottiene l'IP e l'indirizzo del boot server da cui scaricare il sistema operativo (più alcune informazioni opzionali)

4. Viene completata opportunamente l'assegnazione dell'ip al client secondo lo standard del protocollo DHCP

5. Il client invia una discovery del boot server con un messaggio analogo al punto 1 ma codificato come una DHCPREQUEST contenente informazioni differenti

6. Il boot server invia al client un pacchetto DHCPACK contenente il nome del file di boot più altre informazioni di servizio

7. Il client scarica il file indicato (o i files se ce ne sono più d'uno) tramite protocollo TFTP (o MTFTP)

8. Il client determina se è necessario un test di autenticità sul file scaricato ed eventualmente richiede le informazioni al server

9. Dopo aver effettuato il test di autenticità (se necessario) il client avvia l'esecuzione del file di boot scaricato in precedenza

1.4 Alternative esistenti

Attualmente esistono due alternative principali a quanto descritto sopra.

La prima è un chiaro tentativo di migliorare le prestazioni. Per fare ciò utilizza, per il trasferimento dei files di boot, il protocollo MTFTP (Multicast TFTP). Pur offrendo prestazioni migliori rispetto al TFTP classico, introduce delle problematiche e non risulta ottimale come vedremo in seguito.

La seconda, invece, si propone più come un'alternativa in termini di funzionalità piuttosto che di prestazioni (vedremo in seguito perché).

1.4.1 MTFTP

Il protocollo MTFTP[13][4] (Multicast TFTP) è un estensione del protocollo TFTP visto in precedenza. In sostanza utilizza una comunicazione in multicast[12] verso i vari client, con l'obiettivo di far inviare una sola copia dei file da parte del server, lasciando agli apparati di rete, il compito di duplicare i pacchetti verso i vari client. In teoria così facendo, vengono ridotti significativamente i tempi dato che il server invierà una sola copia dei files che i client riceveranno, in linea di massima, contemporaneamente.

Pur essendo una tecnica valida in teoria, il multicast porta delle problematiche[19][10]:

- Inoltro non garantito dei pacchetti (per meccanismo best-effort che in UDP implica arrivo fuori ordine e possibile perdita dei pacchetti)
- La non presenza di congestion-avoidance (sempre a causa dell'uso dello UDP)
- Duplicazione dei pacchetti
- E' necessario che gli apparati di rete e i client supportino il trasferimento tramite multicast; se così non fosse aggiornare i sistemi comporterebbe dei costi aggiuntivi (in realtà la maggioranza dell'HW attuale supporta questo tipo di comunicazione[32] tuttavia potrebbe essere necessario un costo in termini di tempi di configurazione)

Inoltre data la mancanza di uno standard IETF per il MTFTP (viene definito ancora sperimentale nell'rfc stesso del protocollo[13]), l'implementazione non è detto che sia sempre identica a fronte di hardware diversi. Questo rende l'uso del protocollo stesso un po' delicato.

[4] Dettagli sul funzionamento sono disponibili sulle specifiche intel del protocollo PXE[9]

1.4.2 TFTP + NFS

Una seconda alternativa al TFTP come descritto sopra, consiste nell'utilizzare il TFTP per scaricare solo il kernel del sistema in questione per poi montare il file system tramite NFS[22] (Network File System).

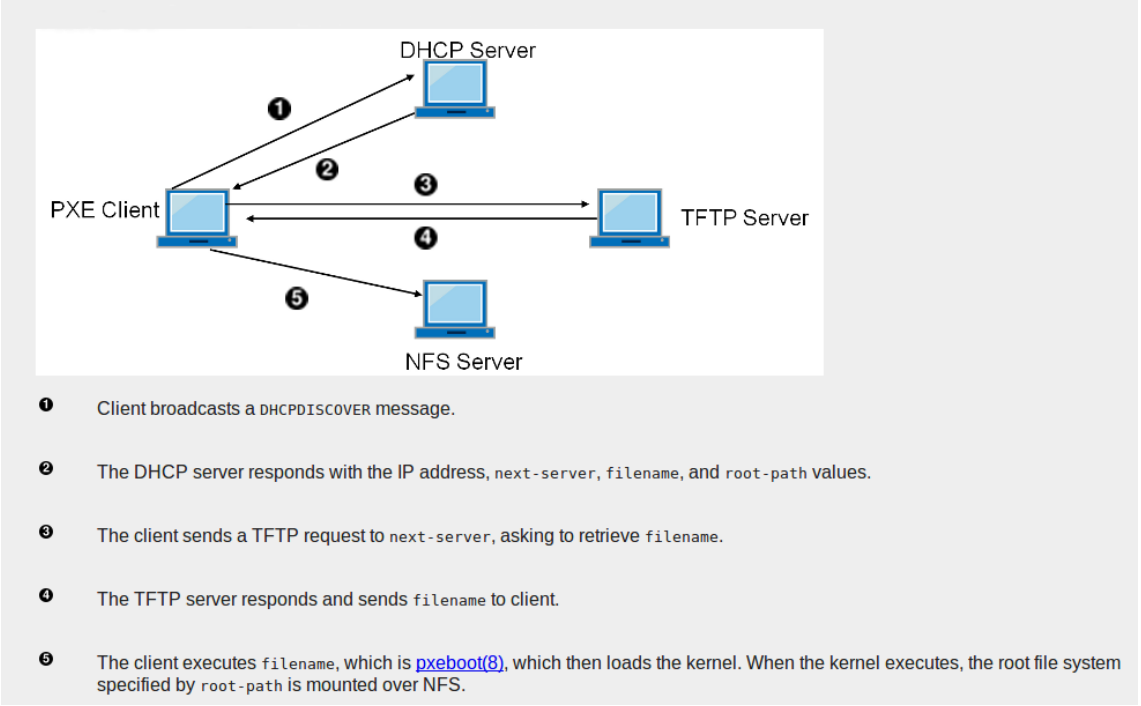

Figura 2: Boot PXE con NFS[21]

Esiste anche la possibilità di trasferire un file system minimale con TFTP e poi di montare solo la home directory tramite NFS ma questo dipende dalla scelta dell'amministratore di rete. Ad ogni modo questo permette di trasmettere solo una minima parte di dati tramite TFTP e di trasmettere il resto tramite NFS (ormai basato su TCP nelle ultime versioni). I principali vantaggi di questo sistema sono una richiesta minima di RAM per i dati (dato che il file system è remoto) e una centralizzazione del file system che semplifica la configurazione del sistema da parte degli amministratori. In caso sia necessario aggiornare del software, basta fare l'update sul server e automaticamente tutti i client troveranno il sistema aggiornato. Come detto in precedenza, questo comporta una differenza più dal punto di vista delle funzionalità, mentre l'interesse di questo lavoro è quello di migliorare le prestazioni del sistema classico (TFTP). Di contro bisogna considerare i tempi di accesso,

lettura e scrittura ottenibili con NFS e confrontarli con quelli ottenibili con dischi locali e, addirittura, (come nel nostro caso) RamDisk. Pur non potendo fornire tempistiche per un confronto preciso, è possibile fornire l'ordine di grandezza di tali misure. Per farlo sono state misurate le prestazioni con l'esecuzione del comando *dd* durante la creazione di files su disco remoto (ricordiamo che la rete disponibile in SiLab è a 100Mbps), disco locale e ramdisk:

	2MB	10MB	100MB	500MB	1GB
RamDisk	0,005s	0,019s	0,178s	0,895s	1,777s
HardDisk	0,011s	0,048s	1,434s	11,855s	23,961s
NFS	0,197s	1,059s	9,196s	48,455s	1m36,461s
RamDisk	607MB/s	568MB/s	578MB/s	573MB/s	577MB/s
HardDisk	231MB/s	221MB/s	71,5MB/s	43,2MB/s	42,7MB/s
NFS	10,7MB/s	9,7MB/s	11,1MB/s	10,6MB/s	10,6MB/s

Come possiamo vedere l'uso di NFS comporta un enorme calo, in termini di prestazioni di I/O, rispetto all'uso di un RamDisk che invece viene usato durante un boot tramite TFTP.

Per questo motivo è stata sviluppata la soluzione proposta di seguito.

1.5 Soluzione proposta

Volendo rivedere in punti la procedura classica, possiamo identificare i seguenti passaggi:

1. Il client riceve la configurazione IP tramite DHCP
2. Il client ottiene l'indirizzo del server TFTP e inizia il download dei file di boot (kernel + immagine del filesystem)
3. Il client termina il download e avvia i file di boot iniziando dal kernel
4. Il kernel, decomprime l'immagine in RAM
5. Completata la decompressione il sistema monta il filesystem e termina il boot

La soluzione proposta, consiste nell'aggiungere uno strato fra il download tramite TFTP (2) e l'avvio del sistema finale (3).

Più precisamente, al punto (2), viene scaricato un sistema minimale (circa 13MB) temporaneo. Essendo una quantità di dati minima, l'impatto in termini di prestazioni è trascurabile. Terminato il download, viene avviato il sistema temporaneo. Questo, appena avviato, inizia il download del sistema finale (kernel + immagine del filesystem) usando il protocollo BitTorrent. A questo punto il sistema, esegue un kexec che scarica dalla memoria il sistema temporaneo e avvia quello definitivo (con le opportune opzioni). A questo punto ci ricolleghiamo al punto (3) avviando il kernel del sistema definitivo. Ora, il processo continua con i punti (4) e (5) terminando così il boot del sistema.

Così facendo si ottimizza l'uso delle risorse di rete, si evitano i colli di bottiglia dovuti alla limitazione di banda a disposizione del server e si riducono sensibilmente i tempi di download. Tutto questo appoggiandosi ad un protocollo robusto e basato su TCP, protocollo, quest'ultimo, che offre migliori garanzie per quanto riguarda il trasferimento dei pacchetti tra i vari client al contrario di quanto è possibile fare con UDP.

1.5.1 Il protocollo BitTorrent

BitTorrent[7][34] è un protocollo di tipo peer-to-peer (P2P) utilizzato per lo scambio di file in rete. Tale protocollo ha come obiettivo quello di distribuire il file verso il maggior numero possibile di utenti che possono quindi scaricarlo e a loro volta diffonderlo (già durante il download del file stesso).

Per funzionare il protocollo ha bisogno delle seguenti entità:

.torrent: è un metafile che contiene le informazioni del file da condividere, tipicamente: nome del file, dimensione, hash del file, numero di parti in cui è diviso il file, indirizzo del tracker.

Tracker: e' un server che ha il compito di coordinare l'attività dei vari peer; viene contattato a intervalli regolari, sia dai client che condividono file, sia da quelli che li scaricano e provvede a fornire a ognuno di loro la lista degli altri client

connessi che possiedono lo stesso file, permettendo così di connettersi reciprocamente. In pratica tiene traccia dello stato di download/upload di tutti i client in modo da permettere la comunicazione fra essi.

Seeders: sono i client che hanno già il file completo e lo stanno solo condividendo; inizialmente ce n'è solo uno che è la fonte iniziale, man mano che altri terminano il download lo diventano a loro volta.

Peers: sono i client che scaricano e condividono il file parziale ottenuto fino a quel momento.

Leechers: il rapporto fra quantità di dati trasmessi e scaricati, è detto seed-ratio. Se un client ha un seed-ratio basso viene definito Leecher (sanguisuga). Il tracker potrebbe essere configurato per privilegiare i client con seed-ratio alti rispetto a quelli con seed-ratio bassi in modo da favorire la diffusione del file.

Swarm: indica il numero complessivo di client che stanno condividendo il file (totale e/o parziale); indica quindi il numero totale di fonti per quel file; non il numero di fonti connesse al proprio client.

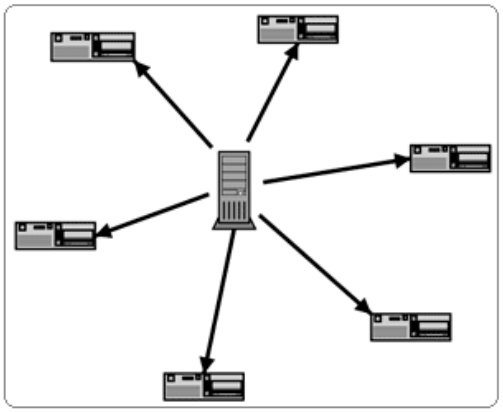

Figura 3: Lo scambio di file con un classico protocollo client-server

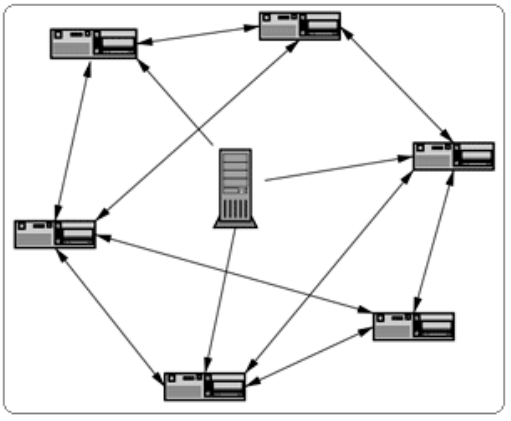

Figura 4: Lo scambio di file con BitTorrent

Dettagli del protocollo

Comunicazione col Tracker Un client che intende scaricare un torrent, una volta ottenuto il file di meta-informazioni (.torrent), deve contattare il tracker tramite il protocollo HTTP con una richiesta GET. Esso fornirà al tracker i dati richiesti tramite la Query String, e riceverà in risposta un dizionario Bencode contenente informazioni sul tracker e gli indirizzi IP dei client connessi. Questi sono i parametri che devono essere inviati al tracker:

- **info_hash**: hash SHA1 del dizionario info codificato in Bencode, in formato stringa codificata secondo le convenzioni URL
- **peer_id**: una stringa di 20 caratteri che permette di identificare in maniera quasi-univoca l'utente sul tracker
- **port**: numero di porta sulla quale il client è in ascolto (le porte tipiche sono nel range 6881-6900)
- **uploaded**: bytes inviati agli altri client dall'inizio della sessione, codificati in ASCII base-10
- **downloaded**: bytes scaricati dagli altri client dall'inizio della sessione, codificati in ASCII base-10
- **left**: bytes rimanenti al completamento del file, codificati in ASCII base-10 (il valore 0 indica un seeder)
- **compact**: indica al tracker di utilizzare il Compact Announce, per l'elenco dei peer (si veda l'elemento peers della risposta)
- **no_peer_id**: indica che il tracker può omettere il campo peer_id nel dizionario dei peers (questa opzione è ignorata se viene specificata l'opzione compact)
- **event**: può assumere i tre seguenti valori: **started**, **stopped**, **completed**. Il primo viene inviato a inizio sessione e indica al tracker che si sta iniziando una nuova sessione. Il secondo chiude la connessione con il tracker e chiede di essere

rimosso dalla lista dei peer. L'ultimo comunica al tracker il completamento del download e il passaggio allo status di seeder

- **ip**: (opzionale) indirizzo IP da comunicare agli altri peer (solitamente utilizzato se si è dietro NAT/router)
- **numwant**: (opzionale) numero di fonti massimo che il tracker deve comunicare
- **key**: (opzionale) stringa randomizzata per una migliore identificazione univoca del client
- **trackerid**: (opzionale) se il tracker ha comunicato in precedenza un tracker id, esso va inviato qui

A questo punto il Tracker risponde con una stringa (che rappresenta un dizionario) con i seguenti elementi:

- **failure reason(stringa)**: se presente, rappresenta la condizione di errore in formato human-readable; in presenza di failure reason non devono essere presenti ulteriori elementi
- **warning message(stringa)**: rappresenta una condizione di allerta in formato human-readable. L'elaborazione della risposta di Announce non viene interrotta e all'utente viene mostrato un messaggio descrittivo
- **min interval(intero)**: tempo minimo in secondi che deve trascorrere tra due richieste Announce, pena il rifiuto da parte del server
- **interval(intero)**: tempo in secondi da raccomandare al client per gli intervalli tra gli aggiornamenti Announce, al fine di non sovraccaricare il server (deve essere compreso tra il valore di min interval e il time-out per la disconnessione forzata dei peer morti)
- **tracker id(stringa)**: una stringa che identifichi univocamente il tracker
- **complete(intero)**: seeder attualmente connessi (utile se il numero di peer connessi supera di molto il limite di risposta)

- **incomplete(intero)**: leecher attualmente connessi (utile se il numero di peer connessi supera di molto il limite di risposta)

- **peers**: Se il client utilizza il *Compact Announce* (1) altrimenti (2):

    1. **peers(stringa):** ogni peer occupa 6 byte in questa stringa; i primi 4 rappresentano l'indirizzo IP in formato numerico, e gli altri 2 la porta

    2. **peers(lista):** lista di dizionari, ciascuno dei quali contiene informazioni sul peer

        - **peer id(stringa)**: il valore arbitrario che il peer ha fornito in fase di connessione
        - **ip(stringa)**: l'indirizzo IP in formato IPv4, IPv6 o DNS
        - **port(intero)**: numero di porta usato dal peer

Per convenzione il numero massimo di peer forniti durante un'interrogazione di Announce non dovrebbe superare i 50.

Comunicazione tra client Terminata la comunicazione col Tracker, il client inizia a comunicare coi peers presenti nella lista appena ottenuta. Tutte le comunicazioni tra peers avvengono su TCP. Le comunicazioni sono simmetriche ovvero i messaggi in entrambe le direzioni sono identici e i dati possono viaggiare in entrambe le direzioni.

Lo stato dei peers (per ogni connessione) è specificato nei messaggi e le condizioni in cui un peer può trovarsi sono le seguenti (per semplicità useremo D per identificare il peer in download e U per quello in upload ricordando tuttavia che qualunque client può essere sia D che U in base alla situazione...):

- **choked**: se D è stato messo choked da U, D saprà che non riceverà nessun dato da U finché questo non lo metterà not choked

- **not choked**: al contrario di choked funge da via libera

- **interested**: serve ad esprimere l'interesse da parte di D verso una parte del file di U

- **not interested**: serve a segnalare ad U la mancanza di interesse per quella parte di file da parte di D

Un trasferimento di dati avviene ogniqualvolta D è *interested* e non è *choked* da parte di U. Lo stato di interesse o non interesse da parte di D nei confronti di U deve essere mantenuto aggiornato per tutto il tempo, anche se D è stato settato choked da parte di U. Questo perché così ogni U sarà in grado di sapere quali D saranno pronti a scaricare immediatamente una volta tolti dallo stato di choked. Le connessioni dei D partono sempre come choked e not interested. Una volta che D ottiene una parte e ne verifica la correttezza tramite hash, deve comunicare a tutti i propri peers che ne è venuto in possesso.

L'uso del choking è giustificato da vari fattori. In particolare il controllo della congestione di TCP risulta povero nel momento in cui vengono stabilite molte connessioni in contemporanea. L'uso del choking permette a ciascun peer di usare un algoritmo stile tit-for-tat[3], per garantire un download-rate consistente.

Il protocollo di comunicazione fra peers, consiste in un handshake iniziale seguito da uno scambio di messaggi di lunghezza prefissata.

Handshake L'handshake è un messaggio richiesto e deve essere il primo messaggio trasmesso da D. E' così composto:
$< pstrlen >< pstr >< reserved >< info\_hash >< peer\_id >$

- **pstrlen**: lunghezza della stringa pstr
- **pstr**: stringa identificativa del protocollo
- **reserved**: 8 byte riservati. Tutte le correnti implementazioni usano 8 zeri. Ciascun bit di questi byte, può essere usato per modificare il comportamento del protocollo
- **info_hash**: 20-byte SHA1 identificativi del file come nel file .torrent
- **peer_id**: stringa di 20-byte usata per identificare univocamente il client

Nella versione 1.0 del protocollo: pstrlen = 19 ; pstr = "BitTorrent protocol".

Messaggi Tutti i messaggi rimanenti sono della forma:
$< lengthprefix >< messageID >< payload >$
La voce *length prefix* corrisponde ad un valore di 4 byte big-endian. Il campo *message ID* è un byte decimale. Il *payload* dipende dal tipo di messaggio:

- **keep-alive**: $< len = 0000 >$ di lunghezza zero, tipicamente inviato ogni due minuti
- **choke**: $< len = 0001 >< id = 0 >$ nessun payload
- **unchoke**: $< len = 0001 >< id = 1 >$ nessun payload
- **interested**: $< len = 0001 >< id = 2 >$ nessun payload
- **not interested**: $< len = 0001 >< id = 3 >$ nessun payload
- **have**: $< len = 0005 >< id = 4 >< pieceindex >$ il payload è un singolo numero, corrispondente all'indice della parte di file appena scaricata e verificata tramite hash
- **bitfield**: $< len = 0001 + X >< id = 5 >< bitfield >$ di lunghezza variabile (X) può essere inviato solo come primo messaggio subito dopo l'handshake; il payload corrisponde ad una maschera in cui i bit a 1 corrispondono agli indici delle parti già in possesso del client. Se il client non ha nulla, può omettere questo messaggio.
- **request**: $< len = 0013 >< id = 6 >< index >< begin >< length >$ usato per richiedere un blocco
  - **index**: indice del blocco richiesto
  - **begin**: offset, all'interno del blocco, da cui iniziare il trasferimento
  - **length**: quantità di dati da trasferire partendo dall'offset indicato
- **piece**: $< len = 0009 + X >< id = 7 >< index >< begin >< block >$ di lunghezza variabile (X), simile a request; **index** e **begin** sono analoghi ma al posto di specificare *length*, si usa **block** che corrisponde all'indice di una

sottoparte del blocco specifica (in pratica usa un indice invece di specificare a mano offset e quantità di dati da copiare)

- **cancel**: $< len = 0013 >< id = 8 >< index >< begin >< length >$ analogo a *request* serve ad annullare la request di un blocco; usa lo stesso payload

Esistono inoltre delle estensioni del protocollo come ad esempio la funzione DHT[4] (Distribuited Hash Table) pensata per eliminare la necessità di un tracker o più precisamente per poter mantenere funzionante la rete anche se il tracker dovesse andare offline. In questa versione c'è un messaggio aggiuntivo che è il seguente:

- **port**: $< len = 0003 >< id = 9 >< listen - port >$ è la porta su cui, il nodo DHT di questo peer, è in ascolto

Con questo, è stato descritto il funzionamento generale del protocollo. Per ulteriori dettagli sulle estensioni non ufficiali, le scelte algoritmiche per il choking, ecc... si rimanda al wiki delle specifiche, non ufficiale, di BitTorrent[34] e all'articolo "*Incentives Build Robustness in BitTorrent*"[6] di Bram Cohen.

2 Progetto

Solitamente il boot di rete viene effettuato per vari scopi:

- Installazione via rete di un sistema operativo
- Necessità di caricare su un pc un sistema operativo diverso da quello normalmente usato, ad esempio per riparare una installazione mal funzionante o per utilizzarlo per funzioni particolari
- Restore completo di un pc da un backup
- Network computing, ovvero utilizzo primario di un sistema operativo caricato dalla rete

Nel caso preso in considerazione, è stata considerata la situazione presente nel laboratorio SiLab del Dipartimento di Informatica dell'Università degli Studi di Milano. Presso il laboratorio, sono presenti circa 130 PC divisi in 3 aule/laboratori. Il problema tipico è che durante i periodi d'esame (ma anche in altre circostanze), occorre rendere disponibili tutte le macchine, con configurazioni differenti da quelle presenti abitualmente in locale, ed in tempi brevi per via del frequente alternarsi di esami nelle diverse aule. Per gli esami, su ogni macchina viene caricata una distribuzione linux direttamente in ram che verrà usata al posto di quelle su disco per svolgere l'esame.Questo permette, per ogni esame, di settare correttamente il sistema in base al tipo d'esame (per esempio la lista di siti in cui è possibile navigare, il layout dell'editor per programmare, l'apertura automatica del browser sulla pagina della documentazione, ecc...).

Per rendere tutto ciò possibile, il personale del laboratorio ha creato un'immagine di Gentoo contenente tutto e solo il software strettamente necessario allo svolgimento degli esami e ad ogni esame, setta la configurazione adeguata tramite script. Questo permette di non dover modificare l'immagine (se non in casi particolari) e di effettuare i settaggi necessari per lo specifico esame direttamente all'avvio del sistema. Tale configurazione, permette di fornire a ciascuno studente un sistema identico agli altri e permette altresì di controllare la libertà d'azione dello studente, per esempio:

limitando la possibilità di navigazione solo a determinati siti predefiniti; settando il linguaggio di programmazione predefinito nell'editor di testo; ecc...

Questo comporta una dimensione dell'immagine abbastanza elevata (circa 400MB compressa, vedremo i dettagli nel capitolo seguente) nonostante si sia limitato il più possibile la quantità di software e addirittura di driver, per mantenere l'immagine il più piccola possibile. Dato che questa viene trasferita ai client con il classico metodo basato su TFTP visto nel paragrafo 1.3, i tempi di trasferimento sono piuttosto elevati e, data la possibilità di dover fare più esami in sequenza e su più aule, diventano un fattore critico (o quantomeno particolarmente delicato) che richiede una soluzione più efficiente.

3 Implementazione

Dopo aver introdotto la problematica nel capitolo precedente, passiamo ora a descrivere la soluzione sviluppata per far fronte al problema. Verrà descritto prima in maniera più dettagliata quanto utilizzato fin'ora; successivamente verranno analizzati i punti critici (come vedremo saranno due) e infine si descriverà nel dettaglio come è stato possibile risolvere i problemi rilevati e si descriverà la nuova procedura ottenuta.

3.1 La procedura attuale

Dunque, prima di tutto consideriamo l'hardware a disposizione. Le macchine a disposizione nel SiLab per gli studenti, sono Pentium4 3.0GHz con 2GB di RAM ciascuna. Tale hardware limita obbligatoriamente la dimensione dell'immagine non compressa che deve essere il più possibile inferiore ai 2GB per poter lasciare della memoria libera per le applicazioni che gireranno. Nel caso del SiLab l'immagine non compressa è di 1,2GB (più circa 3MB di kernel che ovviamente è trascurabile). Tale immagine, viene compressa con BZIP2 usando il massimo livello di compressione fino ad ottenere un file di 384,1MB.

Ciascuna macchina dispone di una scheda di rete gigabit ma dato che il server non dispone di interfaccia gigabit, la rete funziona a 100Mbps. In realtà la situazione è leggermente più articolata perché il server è una macchina virtuale che si trova su una macchina fisica che è collegata alla rete LAN tramite 4 cavi da 100Mbps. Questo fa si che il server veda un'unica interfaccia da 400Mbps (grazie al protocollo LACP[5]) ma che ciascun client possa comunicare col lui, nella migliore delle ipotesi, solo a 100Mbps, perché fisicamente passerebbe da uno solo dei 4 cavi a disposizione.

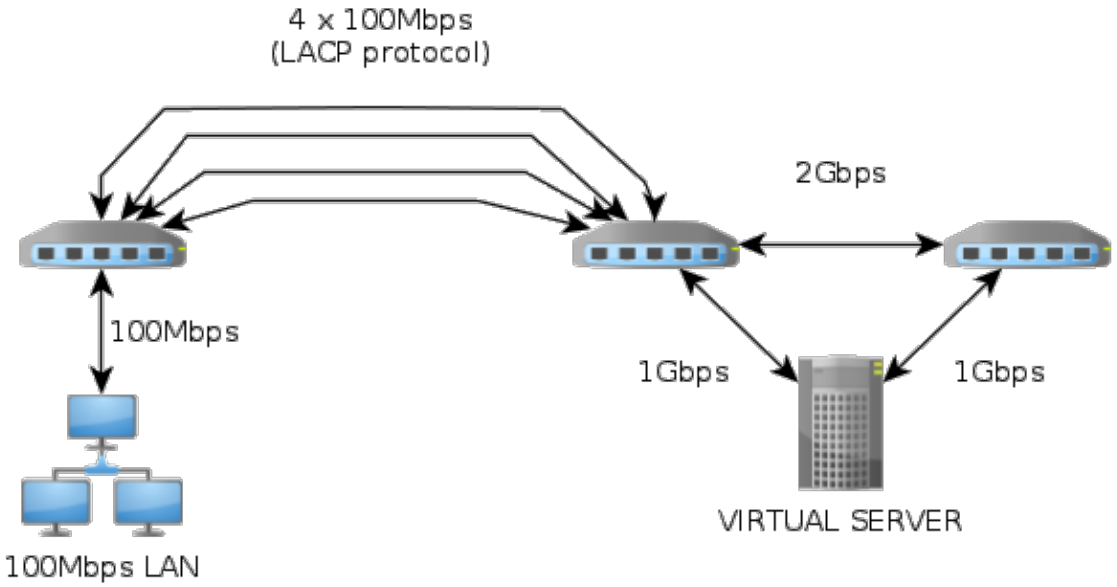

Figura 5: Lo schema della rete presente in SiLab

A questo punto la procedura è la seguente:

1. Dal server viene lanciato uno script che sveglia le macchine necessarie tramite WakeOnLAN[16]

2. Le macchine si avviano ed eseguono il boot usando esattamente la procedura descritta nel paragrafo 1.3

    - Ottengono l'ip tramite DHCP
    - Scaricano dal server i files di boot (kernel + initrd) usando il protocollo TFTP
    - Caricano il kernel
    - Decomprimono l'immagine del file system (initrd)
    - Montano la suddetta immagine e terminano il boot

3. All'avvio del sistema, caricano le configurazioni prestabilite passate dallo script su server

Figura 6: La procedura usata attualmente in SiLab

3.2 I punti critici

Come si può intuire, i punti critici della procedura appena descritta, sono due:

1. Il trasferimento dell'immagine dal server verso i vari client
2. La decompressione dell'immagine una volta terminato il download

Dei due analizzeremo prima di tutto il secondo perché di più immediata risoluzione e poi passeremo a descrivere la soluzione trovata per il punto 1 che poi è il vero obiettivo di questa tesi. Il resto delle operazioni comportano un tempo pressoché costante (minimamente influenzato dall'hardware a disposizione) quindi non rappresentano un problema né tanto meno possibili margini di miglioramento.

3.3 Le soluzioni e la nuova procedura

3.3.1 Algoritmo di compressione per l'initrd

Dunque, il problema di scegliere l'algoritmo di compressione sembra poco rilevante, tuttavia risulta un problema chiave dato che occupa una parte significativa di tempo nel processo di boot. I tre algoritmi principali di compressione che possiamo scegliere sono i seguenti:

- GZIP[11]
- BZIP2[23]

- LZMA[20][5]

Notiamo che dei tre indicati, nella procedura utilizzata in SiLab, viene usato il BZIP2. Quello che a noi interessa valutare, per ciascun algoritmo, è l'impatto di due parametri:

- la dimensione del file compresso
- il tempo necessario alla decompressione del file

Non ci interesserà invece la velocità di compressione dato che l'immagine viene creata una volta e poi modificata molto raramente.

Facendo alcuni test, quello che possiamo verificare è che l'immagine originale della dimensione di 1,2GB viene ridotta come segue:

GZIP 416,7MB

BZIP2 384,1MB

LZMA 307,8MB

Si noti che per quanto riguarda il BZIP2 e l'LZMA, è stato usato il massimo livello di compressione, mentre per il GZIP è stato usato il livello standard dato che la differenza fra lo standard e il massimo è solo di un paio di MB.

Ora invece consideriamo i tempi di decompressione dell'immagine in RAM. Per il test è stata usata una macchina con la seguente configurazione HW: Pentium D 3.4GHz, 2GB DDR667:

GZIP 12s

BZIP2 1m42s

[5]Esiste anche LZMA2, tuttavia non è stato verificato il riconoscimento del formato da parte del kernel e nei test di decompressione offre prestazioni migliori solo di pochi MB in compressione e di pochi secondi in decompressione quindi in seguito si farà riferimento solo all'algoritmo LZMA per praticità. Si lascia all'utente finale la possibilità di usare LZMA2 o qualsiasi altra evoluzione che seguirà a questo lavoro di tesi.

LZMA 1m12s

Quello che appare evidente fin da subito (anche da test esterni[15]) è che l'algoritmo scelto e usato attualmente in SiLab è, dei tre, il peggiore. Questo perché, è quello che ci mette più tempo a decomprimere i files e non è quello che offre il miglior rapporto di compressione. Per quanto riguarda gli altri due, la scelta dipende dall'HW disponibile nel proprio contesto. Il motivo è semplice: i due algoritmi sono i migliori (l'uno rispetto all'altro) solo in una delle due caratteristiche quindi bisogna valutare l'impatto, sui tempi finali, della decompressione del file. Se le macchine in questione sono equipaggiate con HW recente, il tempo di decompressione viene ridotto significativamente e quindi è preferibile il formato LZMA dato che fornisce il miglior livello di compressione possibile. Se l'HW non è recente (come nel caso del SiLab) il tempo di decompressione in RAM risulta significativo e quindi è preferibile scegliere GZIP dato che è straordinariamente rapido nella decompressione anche su HW obsoleto. Il fatto di dover trasferire un'immagine più grossa di circa 100MB viene mitigato dal protocollo BitTorrent e quindi non risulta un problema particolarmente rilevante.

In conclusione l'algoritmo proposto e utilizzato in questo lavoro è stato il GZIP; tuttavia è importante ricordare che se l'HW è recente, potrebbe essere preferibile l'LZMA. L'uso dell'uno o dell'altro è del tutto trasparente al processo dato che il kernel riconosce automaticamente il formato di compressione senza bisogno di specifiche aggiuntive, da parte di chi configura il tutto.

3.3.2 Il trasferimento dell'immagine e la nuova procedura

Ora che abbiamo analizzato il modo migliore di comprimere il file system, verrà descritta la nuova procedura che sfrutta BitTorrent per ottimizzare il trasferimento dell'immagine verso i client.

Come detto in precedenza, la nuova procedura sviluppata, inserisce uno strato fra il download dei file di boot dal TFTP server e il caricamento del sistema operativo finale. Per fare ciò sono state usate 3 componenti principali:

1. Una distribuzione linux minimale

2. Un client torrent a riga di comando

3. Il pacchetto kexec-tools

La distribuzione minimale è stata usata come intermezzo per effettuare il trasferimento del sistema operativo finale usando BitTorrent.

SliTaz: Tra le moltissime distribuzioni minimali disponibili in rete, si è deciso di utilizzare SliTaz[33]. Il motivo di questa scelta è dovuto a vari fattori:

Compatibilità: nella sua versione rolling usa il kernel 3.2[6] che garantisce un ampio supporto all'HW disponibile in commercio (nel nostro caso ci interessa nello specifico il riconoscimento della scheda di rete e poco altro)

Dimensione: il filesystem dell'immagine live è diviso in 4 parti (rootfs1.gz, rootfs2.gz, rootfs3.gz, rootfs4.gz) e in base al tipo di sistema che si vuole avviare è possibile caricarne solo alcuni; in particolare, per i nostri scopi, è sufficiente un sistema a riga di comando (non c'è alcuna necessità di ambiente grafico) e quindi possiamo limitarci ad utilizzare il file rootfs4.gz che occupa meno di 8MB (mentre le altre distribuzioni minimali hanno filesystem unici di circa 50MB) e ovviamente il kernel che occupa 3MB.

Supporto: SliTaz è una distribuzione molto apprezzata e supportata e, sui repository ufficiali, sono disponibili pacchetti precompilati di moltissimi software e utilità; questo, congiuntamente al fatto che viene fornita con un kernel piuttosto recente, garantisce una maggior longevità rispetto ad altre mini-distro che pur essendo molto diffuse e conosciute, sono molto meno aggiornate.

Semplicità: SliTaz possiede anche caratteristiche che ne rendono molto facile la personalizzazione in base alle proprie esigenze (ad esempio la gestione dei runlevel è stata semplificata rendendo molto più semplice aggiungere applicazioni e script personali in avvio automatico)

[6] Al momento della stesura di questa tesi.

La prima cosa da fare è stata ottenere una versione funzionante sotto forma di kernel + initrd e quindi è stata scaricata la ISO della versione rolling direttamente dalla homepage di SliTaz. A questo punto è sufficiente montarla e copiare in una cartella i files che ci interessano che nello specifico sono 3:

rootfs4.gz la porzione di file system che fornisce il solo sistema a riga di comando[7]

bzImage è il kernel di SliTaz

/boot/isolinux/isolinux.cfg è il file da cui leggeremo i parametri da fornire in append nel server TFTP per il corretto avvio di SliTaz

Ora è possibile smontare l'immagine e configurare il server TFTP per renderla disponibile come file di boot ai client. Per fare ciò è sufficiente copiare i due file bzImage e rootfs4.gz e modificare il file di configurazione del TFTP (syslinux.cfg) come segue:

```
default base

label base
        kernel ./bzImage
        append initrd=./rootfs4_rpc.lzma lang=it_IT kmap=it rw root=/dev/null
               ip=dhcp vga=normal autologin
```

A questo punto si può provare a fare il boot di un client per verificare il corretto caricamento della SliTaz (si otterrà una login root root e quindi una shell). Verificato il corretto avvio e riconoscimento della rete, vediamo come personalizzare l'immagine. Si ricorda che l'obiettivo è ottenere una procedura che dal momento del wake up delle macchine al momento in cui sono tutte avviate col sistema finale, non richieda alcun intervento da parte dell'utente.

Per ottenere ciò sarà necessario installare all'interno dell'immagine i pacchetti desiderati (e le relative dipendenze) e aggiungere una serie di script all'avvio che eseguano in automatico il download del sistema finale e che poi eseguano il kexec terminando il boot correttamente.

I pacchetti necessari sono disponibili presso il repository ufficiale e per installarli basta fare quanto segue.

[7]Si noti che anche se il file ha estensione .gz, si tratta in realtà di un archivio lzma

Bisogna decomprimere l'immagine del filesystem in una cartella, per fare ciò si consideri che il file rootfs4.gz, pur avendo estensione .gz, è in realtà un archivio LZMA quindi basterà rinominare il file in rootfs4.lzma e decomprimerlo usando il comando unlzma da riga di comando. A questo punto otteniamo un archivio CPIO, decompresso il quale, si otterrà il filesystem desiderato. Per far ciò è sufficiente usare i seguenti comandi:

```
# mkdir tmp
# cd tmp/
# cpio -id < ../initrd
```

A questo punto abbiamo il root filesystem e possiamo eseguire un chroot[31] al suo interno (su alcune distribuzioni non è stato possibile, tuttavia con Ubuntu e Gentoo non si sono rilevati problemi). Una volta eseguito il chroot, possiamo fare tutte le modifiche di cui abbiamo bisogno.

Dobbiamo installare i pacchetti necessari direttamente tramite il gestore di pacchetti integrato:

```
tazpkg recharge
tazpkg get-install aria2
tazpkg get-install kexec-tools
```

Dobbiamo scrivere lo script che si occupa di scaricare il sistema finale e di avviarlo.

```
#!/bin/ash

#Download dei file tramite torrent

#aria2c --enable-rpc --rpc-listen-all=true --disable-ipv6=true --seed-time=0 -j2
/torrent/2.6.39-gentoo-r3.torrent /torrent/diskless.img.gz.torrent

aria2c --enable-dht=false --disable-ipv6=true --seed-time=0
--file-allocation=none -j5 /torrent/SILab.torrent



#Preparazione ed esecuzione kexec

kexec -l SILab/2.6.39-gentoo-r3 --append="ramdisk_size=1228800
root=/dev/ram rw ip=dhcp udev nodevfs" --initrd=SILab/diskless.img.gz

kexec -e
```

Bisogna mettere tale script in avvio automatico[8] e per farlo bisogna considerare che SliTaz non usa i runlevel, ma il sistema viene inizializzato da uno script primario che a sua volta chiama alcuni script accessori e ogni altro script indicato dall'utente nel suo file di configurazione. Nello specifico bisogna aggiungere lo script nella cartella /etc e poi va inserito nella sezione scripts del file /etc/rcS.conf. A questo punto verrà eseguito correttamente all'avvio.

Poi bisogna inserire il file .torrent precedentemente creato dal server in modo che il download possa avvenire correttamente[9].

A questo punto è possibile ricreare il file CPIO contenente il filesystem modificato e ricomprimerlo con lzma usando i seguenti comandi:

```
# find . | cpio --create --format='newc' > ../newinitrd
# cd ../
# lzma newinitrd
```

Si noti che potrebbero sorgere problemi, a causa di permessi mancanti su alcuni files. Per risolvere, è sufficiente assegnare i permessi necessari tramite chmod dopo essere usciti da chroot. In caso di dubbi si può assegnare tutti i diritti senza troppe preoccupazioni, visto che il sistema verrà eseguito in ambiente controllato e solo per il tempo necessario al download, dopodiché verrà scaricato dalla RAM. Per farlo basterà usare: *chmod -R 777 ./slitazdir/**.

Ora abbiamo il filesystem correttamente configurato per completare tutte le operazioni in automatico come desiderato. Naturalmente le configurazioni e gli script mostrati in questa tesi sono indicativi; potrebbe essere necessario aggiungere e/o modificare alcuni comandi per adattarli alla specifica situazione (non tutti i laboratori hanno le stesse configurazioni) tuttavia come linea generale quanto indicato è corretto.

Configurata correttamente l'immagine di SliTaz è necessario configurare il server in modo tale che possa creare il torrent della cartella contenente i file del sistema

[8] come indicato nella documentazione[27]

[9] Per il momento il .torrent è inserito a mano, tuttavia sarebbe meglio fare in modo che tale file venga scaricato dal server con una semplice wget in modo che future modifiche del file principale non rendano necessario mettere mano anche all'immagine di SliTaz (al momento questa modifica, rientra fra le ottimizzazioni che sono ancora in corso e che faranno parte del tool finale)

finale e che li metta in condivisione (seeding) per tutti i client.

Questo richiede l'installazione sul server di due pacchetti (si suppone che i server DHCP e TFTP siano già configurati):

- Il pacchetto aria2
- Il pacchetto ufficiale BitTorrent[10]

Aria2 verrà usato per la condivisione del file, mentre il pacchetto di bittorrent per l'attivazione del Tracker e la creazione del file .torrent (aria2 è solo un client; non ha funzionalità di Tracker o di creazione dei .torrent). A questo punto sarà sufficiente copiare in una cartella i file del sistema finale (nel caso del SiLab sono stati messi in una cartella ed è stato creato direttamente il .torrent della cartella) e lo script che esegue le operazioni di avvio tracker, creazione e pubblicazione del .torrent, condivisione e chiusura:

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Questo script usa i pacchetti bittorrent e aria2 disponibili nei repository +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#Attivo il Tracker in background sulla porta 8000 (ovviamente posso specificare
# quella che voglio)

echo ">>> Avvio il Tracker http://indirizzodelserver:8000/announce"
/usr/bin/bittorrent-tracker --port 8000 --dfile dfile > log_tracker.txt &

#Creo i torrent e li associo al Tracker (va fatto ogni volta perche'
#se il file cambia, il nome del file e' lo stesso
#ma l'hash nel metafile e' diverso)
#(ip e porta sono modificabili a piacere in base alle proprie impostazioni di rete)

echo ">>> Creo i torrent aggiornati e li pubblico sul Tracker"

/usr/bin/maketorrent-console http://indirizzodelserver:8000/announce SILab

#Avvio il client passandogli tutti i torrent nella cartella corrente
```

[10]Nel nostro caso, sul server era presente Gentoo ed è stato notato che i comandi disponibili a seguito dell'installazione del pacchetto erano differenti da quelli disponibili a seguito dell'installazione su sistemi Debian-like. Per verificare i comandi effettivamente disponibili si usi il comando *dpkg -L bittorrent* Per ciascun comando è poi disponibile un piccolo manuale consultabile tramite *man nomecomando*.

```
echo ">>> Avvio la condivisione dei torrent"

#imposto il seed-ratio a 999.0 in modo che sia un valore talmente alto da non
#essere raggiunto;
#in questo modo aria2 resta aperto in seeding finche' non viene chiuso a mano

aria2c --enable-dht=false --disable-ipv6=true --bt-seed-unverified=true
--seed-ratio=999.0 --listen-port=6882 SILab.torrent

#aria2c --enable-rpc --rpc-listen-all --disable-ipv6=true --bt-seed-unverified=true
--seed-ratio=999.0 *.torrent


#Killo il tracker alla fine del lavoro (in realta' puo' convenire farlo a mano)

echo ">>> Termino il tracker"
kill $(pidof -x bittorrent-tracker)
```

Avviando lo script verrà attivato il tracker, verrà generato il file .torrent (che verrà automaticamente pubblicato sul tracker) e verrà messo in condivisione tramite aria2. Una volta che non è più necessario avere la condivisione attiva, basta terminare aria2 per permettere la prosecuzione dello script che non farà altro che terminare il processo del tracker prima di chiudersi.

Ovviamente lo script al momento è pensato per essere lanciato a mano e terminato a mano. In base all'esigenza è possibile cambiare le configurazioni per esempio impostando il tracker come servizio all'avvio in modo che sia sempre attivo, oppure si può configurare tutto con cron (le possibilità di personalizzazione sono moltissime). In questo lavoro viene presentata una versione base funzionante; starà poi agli amministratori di rete personalizzare la configurazione finale dei comandi in base alle proprie esigenze.

Aria2: Aria2[26] è un client torrent utilizzabile da riga di comando; è stato scelto questo perché si tratta di un client veramente completo e altamente personalizzabile (controllo remoto, gestione eventi, ...) nonché veloce e pratico.

Tra le sue caratteristiche, quelle che sono risultate più importanti sono:

1. la possibilità di chiudere automaticamente il programma al verificarsi di una

particolare situazione, ovvero nello specifico, al termine del download o al raggiungimento di un determinato seed-ratio o seed-time[29]

2. la possibilità di controllare, o meno, ogni parametro dei client usando la comunicazione RPC[30]

Negli script mostrati sopra, si può osservare che, nel caso di aria, vi è sempre anche una versione del comando dove viene usato l'rpc (ovviamente commentata se è usata l'altra). L'uso dell'rpc permette di controllare tutti i client e di monitorare/loggare lo stato dei download.

L'uso o meno di tale modalità all'atto pratico dipende dalle scelte implementative e dalle esigenze specifiche di ciascuna situazione. Quando viene attivato l'rpc, aria2 si avvia, scarica i file indicati e poi invece di chiudersi automaticamente, resta in ascolto sulla porta rpc (default 6800). Così facendo, è possibile terminare il programma da remoto solo quando tutti i client hanno terminato il download. In questo modo si massimizza la quantità di fonti perché tutti i pc (meno uno ovviamente) resteranno in seeding finché anche l'ultimo non avrà terminato il download. Solo a questo punto il server comanderà la chiusura di aria2 sui vari client permettendo la prosecuzione del boot. Così facendo, è stato verificato sperimentalmente che viene ridotta al minimo la quantità di dati che il server dovrà inviare (più fonti = meno richieste al server) ma non c'è un significativo aumento di velocità. Di contro invece richiede che il server controlli periodicamente lo stato di tutti i client complicando leggermente la gestione dell'intera fase di boot.

Un esempio di come è possibile controllare lo stato e terminare aria2 al momento opportuno è rappresentato dallo script seguente che è stato usato per rilevare i tempi e le statistiche durante i test in SiLab:

```
#!/bin/bash

DELAY=1
LOGFILE=log_tracker.txt
MONLOGFILE=$(date +%Y%m%d%H%M%S)_monscript.log
SEEDINGLIST=seeding.list
NONSEEDINGLIST=nonseeding.list
TRACKER=159.149.137.252

#porta di default rpc 6800

date
/root/.scripts/tuttoprogrammazione
date

################################################
while
 sleep $DELAY
do


 ###################################################################
 # crea log completo (tutto lo stato della rete, in accrescimento)
 cut -f 1 -d" " $LOGFILE |sort|uniq|while
        read NODE
 do
        echo -n $(date +%Y%m%d";"%H%M%S)";"$NODE";"
        #./aria2mon --server=$NODE |grep GID | tr "\n" "-"
         ./aria2mon --server=$NODE |grep GID
        #echo
        #./aria2rpc shutdown --server $NODE
 done | tee -a $MONLOGFILE


 ###################################################################
 # crea file macchine in SEEDING (possono bootare)
 grep SEEDING $MONLOGFILE |cut -f3 -d";"|sort|uniq > $SEEDINGLIST
 # pero' attenzione che il monlog va in append per cui se si riparte
 #va azzerato o pensare ad altro


 ###################################################################
 # crea file macchine non in SEEDING
 echo -n > $NONSEEDINGLIST
 grep SEED $MONLOGFILE |cut -f3 -d";"|sort|uniq| while
  read seed
```

```
do
 if
  ! grep -q $seed $SEEDINGLIST
 then
  echo $seed >> $NONSEEDINGLIST
 fi
done

#################################################################
# crea file macchine NON RISPONDONO (hanno gia' bootato)
# verificare errore rpc
# verificato, manda exit code, perfetto!


#################################################################
# spengo quelle in seeding
# quando SEEDINGLIST non e' vuota e NONSEEDINGLIST e' vuota
if
 test -s $SEEDINGLIST -a ! -s $NONSEEDINGLIST
then
 grep -v $TRACKER $SEEDINGLIST|xargs -n 1 ./aria2rpc shutdown --server 2>/dev/null
 exit
fi

done
```

Per quanto riguarda la realtà del SiLab, comunque, è stato scelto di non usare l'rpc per quella che sarà la versione finale del tool. Questo per semplici ragioni di organizzazione interna dato che a livello pratico, come detto, il non utilizzo dell'rpc, semplifica la gestione del processo di boot aumentando solo di poco il carico sul server.

Kexec-tools: Kexec[28] è una system call che permette di caricare e bootare un altro kernel direttamente da quello attuale eventualmente passandogli come parametro anche un nuovo filesystem (più tutti i vari parametri di boot necessari). Il comando, scaricherà dalla memoria il vecchio sistema e avvierà quello nuovo, restituendo quindi una macchina avviata perfettamente identica a quella che avremmo ottenuto col vecchio sistema. Questo rende di fatto, quanto di nuovo descritto fin'ora, del tutto trasparente rispetto alla procedura classica.

Il kexec viene eseguito in due fasi:

1. nella prima, viene fatta una load del kernel passandogli i parametri di boot in append

2. una volta terminata la load del kernel, viene eseguito a tutti gli effetti il kexec

I comandi sono visibili nello script di boot mostrato nel paragrafo su SliTaz 3.3.2. Vengono riportati anche di seguito per praticità:

```
#Preparazione ed esecuzione kexec

kexec -l SILab/2.6.39-gentoo-r3 --append="ramdisk_size=1228800
root=/dev/ram rw ip=dhcp udev nodevfs" --initrd=SILab/diskless.img.gz

kexec -e
```

Considerazioni: A questo punto è utile riassumere la nuova procedura che può essere descritta con i seguenti passaggi:

1. Le macchine vengono svegliate dal server tramite WakeOnLAN

2. Le macchine si avviano e ottengono, tramite DHCP esteso, sia la propria configurazione IP sia l'indirizzo del server TFTP

3. Scaricano dal server, in TFTP, i files necessari all'avvio di SliTaz

4. Caricano il kernel di SliTaz

5. Il kernel decomprime l'immagine del file system (initrd) e carica SliTaz

6. Parte in automatico lo script di boot basato su BitTorrent (3.3.2)

7. Vengono scaricati kernel e initrd del sistema finale (nel nostro caso Gentoo)

8. Viene fatta la load del kernel finale e viene lanciato il kexec che lo avvia

9. Viene decompresso l'initrd finale

10. Viene montato il filesystem e avviato il sistema finale, terminando così il boot

11. All'avvio del sistema, vengono caricate le configurazioni prestabilite passate dallo script su server

Come è facile osservare, i punti dal 3 al 9 rappresentano l'evoluzione del sistema classico e risultano del tutto trasparenti agli altri passaggi.

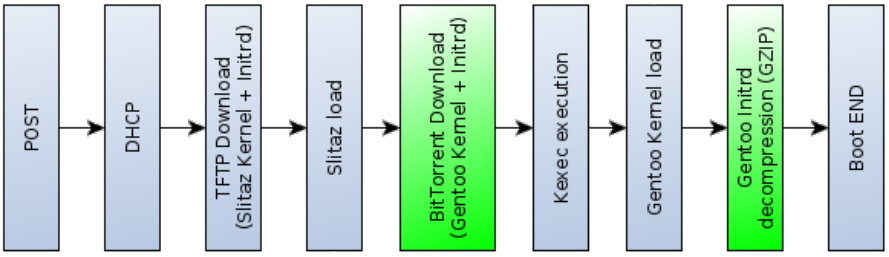

Figura 7: La nuova procedura proposta

4 Analisi prestazioni

Per quanto riguarda le prestazioni, verranno presentati i risultati degli ultimi test effettuati con la versione più recente del tool. I test in questione sono stati fatti usando la modalità RCP di aria2 in modo da poter loggare meglio i risultati. Sono da considerare attendibili anche se ovviamente differenti configurazioni potrebbero portare a risultati leggermente diversi (ad esempio come detto sopra il mancato utilizzo dell'RPC causa un lieve aumento del carico sul server) ma in linea di massima possono essere presi come riferimento attendibile. Si noti, per correttezza, che i tempi parziali registrati nel caso del boot col sistema classico sono stati registrati a mano, dato che il protocollo TFTP non offre funzionalità di log, e potrebbero quindi non essere del tutto precisi. I tempi di boot totali, invece, sono da considerarsi attendibili in quanto facilmente rilevabili anche a mano.

Nel SiLab, sono presenti 124 pc divisi su tre aule; più precisamente sono ripartiti come segue:

- Aula Omega = 26
- Aula Tau = 41
- Aula Sigma = 57

I test sono stati eseguiti confrontando i tempi di avvio e il carico sul server col metodo usato fin'ora (TFTP classico) e i tempi registrati con la nuova procedura. Inoltre il confronto è stato fatto aumentando gradualmente il numero di macchine coinvolte. Per problemi di disponibilità delle aule (i test portano via parecchio tempo e richiedono la disponibilità del personale del SiLab) il test consiste in 3 confronti:

- Con 8 pc
- Con tutti i pc dell'aula Tau (41)
- Con tutti i pc presenti in SiLab (aule Omega, Tau e Sigma) (124)

Si noti che per evidenziare i miglioramenti apportati dalla correzione di entrambe le problematiche evidenziate al punto 3.2 per i test con metodo classico è stato mantenuto il formato di compressione abituale (BZIP2) mentre per la nuova procedura è stato usato il formato GZIP.

Per ogni caso sono stati registrati:

- i tempi di boot totali
- i tempi di download
- il seed-ratio del server dopo che tutti i client hanno terminato il boot
- la velocità media di download

E' stato così possibile tracciare dei grafici che mostrano come la nuova procedura sia significativamente migliore rispetto al sistema classico; sopratutto a fronte di un alto numero di macchine. Si noti infine che i test sono stati effettuati in un giorno di vacanza in cui l'uso della rete in dipartimento era ridotto al minimo data la quasi totale assenza di personale e studenti. Questo fattore potrebbe aver portato dei vantaggi alla procedura classica, fortemente dipendente dal carico presente sul server. Di contro l'influenza sul trasferimento tramite BitTorrent è minimo dato che il coinvolgimento del server stesso nei trasferimenti è a sua volta minimo.

Nelle pagine seguenti, sono riportati i grafici che rappresentano i risultati dei test e alcune foto delle aule durante il boot con BitTorrent.

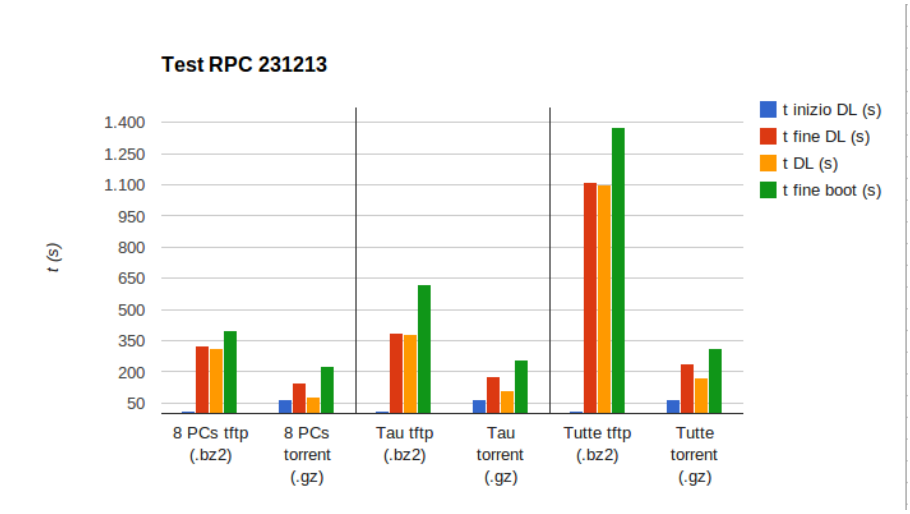

Figura 8: I risultati generali con i tempi parziali e totali.

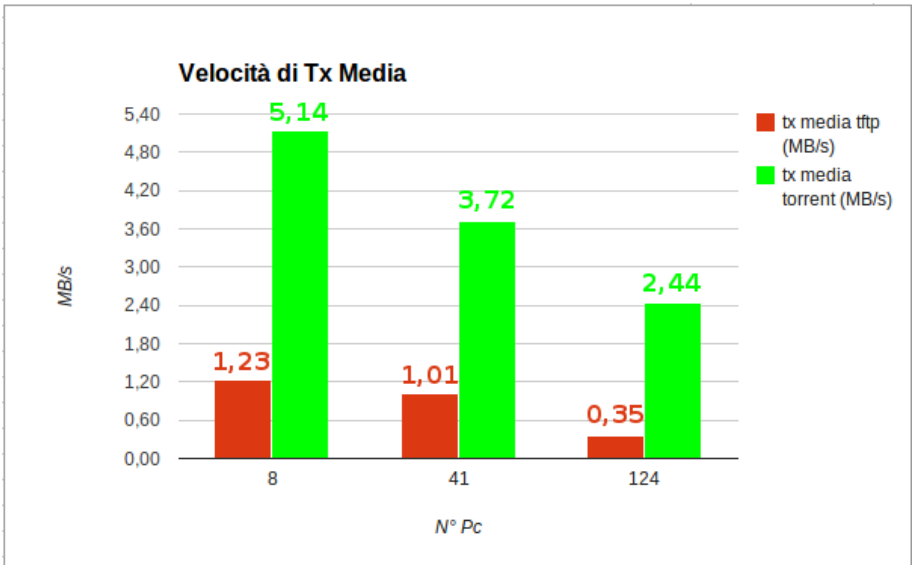

Figura 9: Velocità media di trasferimento dell'immagine a ciascun client (valori più alti = migliori prestazioni).

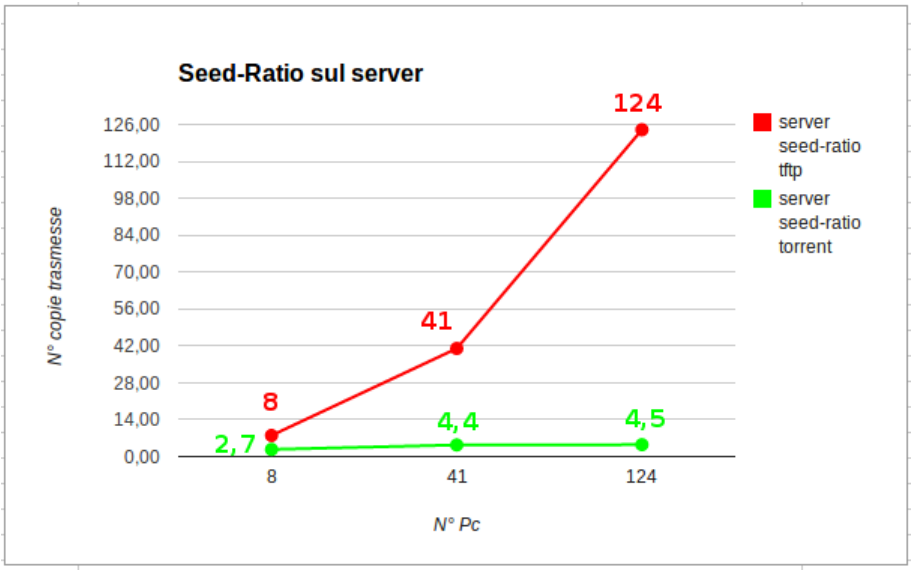

Figura 10: La quantità di dati trasmessa dal server verso i client (valori più bassi = migliori prestazioni).

Come spiegato nel paragrafo 1.5.1, il seed-ratio rappresenta la quantità di dati che un client invia rispetto al totale di dati scaricati. Dato che il server ha una copia intera dei dati, un seed-ratio di 1 corrisponderebbe all'aver inviato il 100% dei dati posseduti. Valori più alti corrispondono a maggiori quantità di richieste soddisfatte dal server. Anche se il termine seed-ratio ha poco senso nel TFTP (dato che i client non inviano nulla ma si limitano a scaricare), è stato usato lo stesso, per fare il confronto tra le quantità di dati inviati dal server. Chiaramente, in TFTP, il server invia una copia intera per ogni client quindi avrà sempre un seed-ratio massimo (e questa è una delle problematiche che vogliamo risolvere). Valori di seed-ratio bassi indicano che il server ha inviato meno dati e quindi fornisce un'idea della quantità di dati scambiati tra i vari client senza il bisogno di passare dal server. Dal grafico possiamo vedere che, anche in presenza di un significativo aumento di client, il seed-ratio del server rimane quasi inalterato al contrario del sistema classico dove il seed-ratio aumenta in maniera direttamente proporzionale.

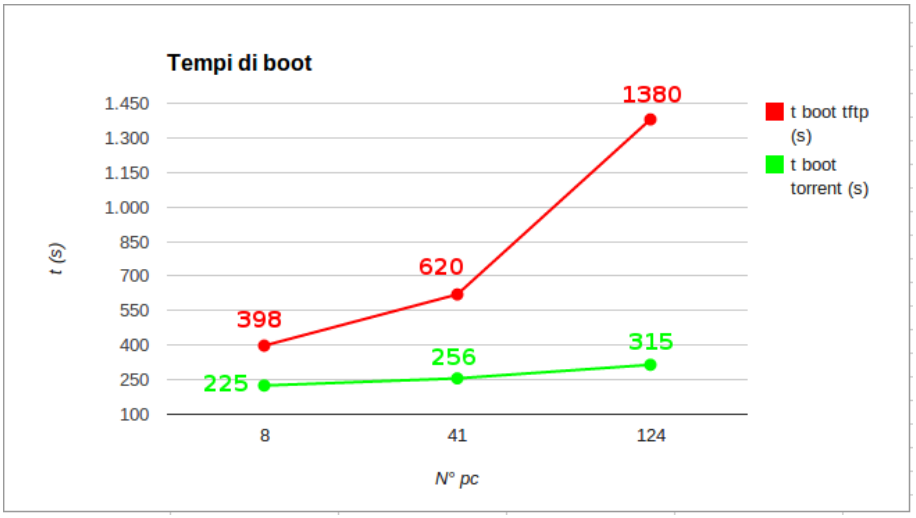

Figura 11: I tempi complessivi di boot di tutti i pc dal wake up a quando sono tutti avviati (valori più bassi = migliori prestazioni).

Figura 12: Aula Sigma durante il boot tramite BitTorrent.

Figura 13: Aula Omega durante le fasi finali del boot tramite BitTorrent.

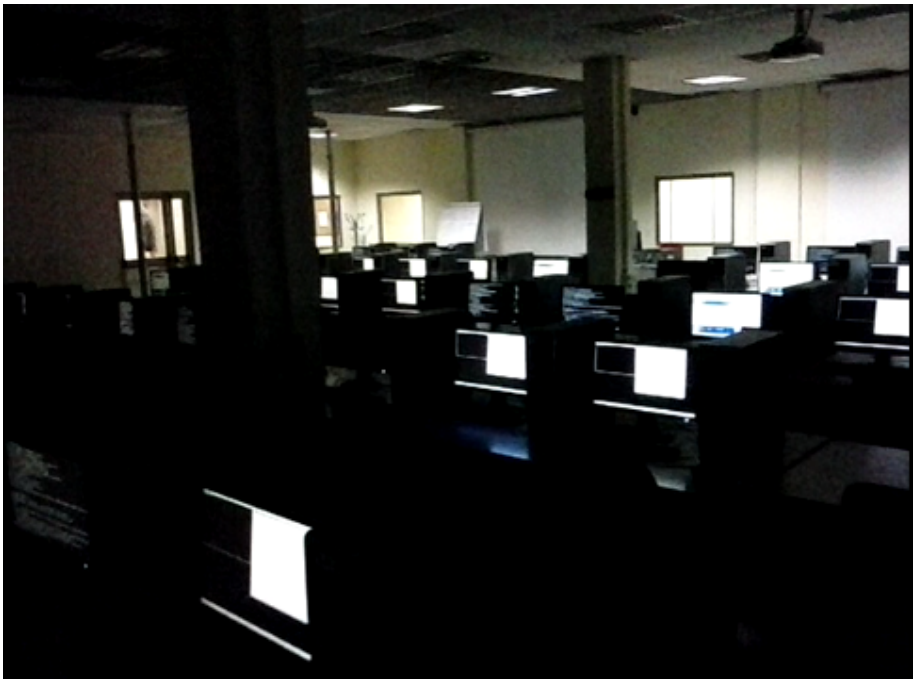

Figura 14: Aula Tau durante le fasi finali del boot tramite BitTorrent.

5 Conclusioni

Come è stato possibile vedere nei capitoli precedenti, la procedura presentata, è in grado di abbattere drasticamente sia i tempi di boot complessivi, sia la quantità di dati che il server deve trasmettere ai client, rispetto alla classica procedura basata su TFTP. In particolare viene massimizzato e ottimizzato l'uso della rete disponibile garantendo ottime prestazioni anche con reti non particolarmente veloci. I miglioramenti rilevati, sono inoltre tanto maggiori quanto maggiore è il numero di macchine coinvolte. Inoltre la presenza di un kernel aggiornato, per quanto riguarda SliTaz, garantisce un ottimo supporto hardware.

La necessità di usare comunque il protocollo TFTP per il trasferimento del sistema operativo iniziale, risulta poco rilevante, data la dimensione trascurabile del sistema stesso, mentre l'uso del protocollo BitTorrent, garantisce le funzionalità più avanzate, proprie di TCP, per l'instradamento dei pacchetti, garantendo così un'alta qualità ed affidabilità, nel trasferimento dei dati.

Per quanto riguarda le problematiche economiche, (da non sottovalutare in un contesto reale), la procedura presentata, non richiede alcuna modifica o aggiunta all'hardware esistente e richiede solo l'installazione di un paio di pacchetti software disponibili gratuitamente (e/o addirittura opensource). Questo rende il tutto completamente a costo zero e richiede una configurazione iniziale particolarmente ridotta (per esempio i server DHCP e TFTP già in uso non richiedono alcuna modifica) risultando così facilmente implementabile per chiunque.

Infine, oltre a far uso a sua volta di software utilizzabili liberamente, quanto realizzato in questo progetto, è reso disponibile come software libero in modo da poter essere utilizzabile da chiunque ne abbia bisogno senza costi di alcun genere.

Riprendendo il lavoro citato nell'introduzione (Moobi[17]), possiamo inoltre fare qualche confronto ed osservare che per quanto riguarda il guadagno di prestazioni i risultati sono simili (Moobi segnala un rapporto 1:4 rispetto al non uso di BitTorrent; molto simile a quanto riscontrato nei nostri test con un numero alto di macchine). Di contro però, l'implementazione di Moobi descritta nell'articolo risulta specifica per RedHat. Il motivo è che a differenza del metodo proposto, Moobi usa un solo sistema

(direttamente quello finale). Questo, in breve, viene diviso in due parti: una minima (che verrà scaricata insieme al kernel direttamente in TFTP) alla quale poi viene unita quella più pesante e problematica (/usr/); è proprio quest'ultima che Moobi scarica usando BitTorrent. Dato che il lavoro è stato fatto con RedHat i tools usati e il resto sono pensati per quel sistema; volendolo usare con altri sistemi è necessario adattare il tutto per quel sistema (pacchetti .deb al posto di .rpm; client torrent e tool/script opportuni) e questo chiaramente richiede una profonda conoscenza del sistema di boot usato da Moobi. Invece, col lavoro presentato in questa tesi, il sistema finale è del tutto indifferente (purché si tratti di una distribuzione linux e non di windows).

Nell'articolo, viene anche segnalata la necessità di aumentare il supporto HW del tool, mentre quanto realizzato in questo lavoro, come detto sopra, garantisce un ottimo supporto all'HW in circolazione dato l'utilizzo del kernel 3.2.x per SliTaz. Inoltre, sempre analizzando il problema dell'adattabilità a situazioni differenti, Moobi richiede un'ottima conoscenza dello strumento stesso per poter essere adattato a situazioni differenti. Questo perché il *linuxrc* usato da Moobi è particolarmente specifico per l'immagine in uso e l'immagine viene generata in modo piuttosto articolato.

Un'immagine di Moobi, è composta da varie immagini di file-system (immagini parziali che poi vengono unite), file skeleton per il file-system e il kernel. Lo skeleton di un file-system non è altro che un file di testo che specifica la proprietà e i permessi dei file di un file-system; in Moobi viene usato per /var/ e /tmp/. Le immagini di file-system principali sono / (che è il root file-system minimale) e /usr/ (la parte più grossa che verrà trasferita tramite BitTorrent). Per la creazione dell'immagine, viene usato uno script che usa dei file di configurazione per eseguire la creazione; un esempio di file di configurazione è il seguente:

```
 [partition:/]
name=root.img
size=128M
compress=1
skipmd5=1
 [partition:/usr]
name=usr.img
size=1024M
compress=0
```

```
skipmd5=1
 [partition:/var]
name=var.skel
skeleton=1
compress=0
skipmd5=1
```

A questo punto un tipico script di creazione immagine esegue le seguenti operazioni:

1. crea una cartella temporanea di lavoro da usare come cartella root per l'installazione
2. esegue un pre-script per inizializzare la cartella appena creata
3. trova ed installa i pacchetti specificati nella lista dei pacchetti necessari
4. copia ogni file stand alone da un mini-root; il linuxrc viene tipicamente inserito qui
5. esegue un post-script per gli ultimi fix sulla cartella di lavoro creata al punto (1)
6. crea i file di immagine delle partizioni partendo dal percorso più profondo risalendo in ordine fino alla radice
   (a) genera un file di immagine da /dev/zero/ della dimensione opportuna
   (b) formatta il suddetto file in ext2 o ext3
   (c) monta il suddetto file
   (d) vi copia all'interno le cartelle appropriate per quell'immagine
   (e) smonta l'immagine ed esegue le operazioni finali necessarie (come ad esempio la compressione del file d'immagine)

I file di skeleton sono generati per ogni partizione scansionando le opportune sotto-cartelle e registrando i permessi e le proprietà dei files contenuti.

A questo punto un tipico processo di boot di Moobi è il seguente[11]

[11]Si legga l'articolo[17] per una versione più dettagliata.

- avvio del pc
- dhcp setup
- tftp download di kernel e initrd (root file-system minimale)
- kernel load e mount dell'initrd
- inizializzazione del kernel e avvio del linuxrc
- riconoscimento HW da parte di linuxrc
- reinizializzazione della configurazione di rete
- richiesta del file .torrent tramite HTTP
- download dell'immagine grossa tramite BitTorrent
- al termine viene opportunamente montata l'immagine
- vengono ottenuti i file skeleton tramite HTTP e vengono configurati i permessi dei file nelle varie partizioni come indicato nei files
- viene passato il controllo a init per il termine del boot

Come è possibile notare, il processo di creazione è fortemente dipendente dal tipo di sistema usato; al contrario, il tool proposto in questo lavoro è del tutto trasparente e richiede veramente una personalizzazione minima. Non obbliga ad utilizzare un metodo specifico per creare la propria immagine di sistema finale; ognuno può crearla come meglio crede, l'importante è che poi siano disponibili il file del kernel, il filesystem compresso e i parametri necessari al boot (che verranno poi usati dal kexec all'interno di SliTaz). Ad ogni modo, tralasciando per un momento le problematiche appena descritte, (implementazione specifica per RedHat, scarso supporto HW, complessità di personalizzazione dello strumento), è interessante notare la somiglianza dei risultati ottenuti, in termini di prestazioni, in entrambi i lavori.

Al momento, la procedura proposta richiede ancora qualche perfezionamento in modo tale che, l'uso in contesti differenti dal SiLab, non richieda di mettere mano

all'immagine di SliTaz ed agli script usati, tuttavia ne sono stati verificati sia il corretto funzionamento sia il significativo incremento di prestazioni. L'intenzione finale è di rendere il tutto gestibile dall'esterno, tramite parametri opportuni, in modo che, ad esempio, il cambio dell'immagine finale non richieda di aprire l'immagine di SliTaz per aggiornare il relativo file .torrent e le opzioni usate dal kexec, ma sia sufficiente specificarle come parametri di avvio a SliTaz. Allo stesso modo si vuole rendere possibile specificare dall'esterno tutte le opzioni possibili (usare o meno l'RPC, indicare l'indirizzo del tracker, ecc...). Tutto ciò verrà realizzato quanto prima e, a seguire, sarà reso disponibile in rete tramite il sito dell'SL-Lab (`http://sl-lab.it/dokuwiki/doku.php/tesi:boottorrent`).

Riferimenti bibliografici

[1] http://www.bittorrent.com/intl/it/sync.

[2] https://www.dropbox.com/.

[3] Bertoni Alberto. Algoritmi ii (prima parte), p.40,41,42, 2011/2012. http://homes.di.unimi.it/~bertoni/Algoritmi%202%20-%20Parte%201.pdf.

[4] Arvid Norberg Andrew Loewenstern. Dht protocol, 2008. http://bittorrent.org/beps/bep_0005.html.

[5] IEEE Std 802.1AX-2008 Chapter 5.4. Ieee standard for local and metropolitan area networks — link aggregation. http://standards.ieee.org/getieee802/download/802.1AX-2008.pdf.

[6] Bram Cohen. Incentives build robustness in bittorrent, 2003. http://www.bittorrent.org/bittorrentecon.pdf.

[7] Bram Cohen. The bittorrent protocol specification., 2008. http://bittorrent.org/beps/bep_0003.html.

[8] cordis.europa.eu. Articoli di approfondimento - tv e internet: un'unione nel paradiso dell'intrattenimento., 2012. http://cordis.europa.eu/result/brief/rcn/9967_it.html.

[9] Intel corp. The preboot execution environment specification v2.1. 1999.

[10] Marco Corradi. Chiarimenti: Multicast. http://www.ce.unipr.it/~corradi/livestreamer/multicast.htm.

[11] P. Deutsch Aladdin Enterprises. rfc1952 gzip file format specification version 4.3., 1996. http://www.ietf.org/rfc/rfc1952.txt.

[12] Fred Halsall. *Networking e Internet, 5ed., p. 6, 347-350, 384.* Pearson, 2006.

[13] A. Emberson Lanworks Technologies Inc. rfc2090 tftp multicast option., 1997. http://tools.ietf.org/html/rfc2090.

[14] J. Reynolds ISI J. Postel. rfc959 file transfer protocol (ftp)., 1985. http://www.ietf.org/rfc/rfc959.txt.

[15] klausman. Gzip, bzip2 and lzma compared., 2008. http://blog.i-no.de/archives/2008/05/08/index.html#e2008-05-08T16_35_13.txt.

[16] Philip Lieberman. Wake-on-lan technology, 2010.

[17] Chris McEniry. Moobi: A thin server management system using bittorrent, 2007. http://static.usenix.org/event/lisa07/tech/full_papers/mceniry/mceniry_html/.

[18] K. Sollins MIT. rfc1350 the tftp protocol (revision 2)., 1992. http://www.ietf.org/rfc/rfc1350.txt.

[19] Marco Marletta GARR-B NOC. Multicast tutorial - il multicast nella rete dell'universita e della ricerca: la conoscenza da uno a molti. http://www2.garr.it/ws4_pdf/Multicast.pdf.

[20] Igor Pavlov. Lzma sdk (software development kit). http://7-zip.org/sdk.html.

[21] Craig Rodrigues. Pxe booting with an nfs root file system. http://www.freebsd.org/doc/handbook/network-pxe-nfs.html.

[22] D. Robinson R. Thurlow Sun Microsystems Inc. C. Beame Hummingbird Ltd. M. Eisler D. Noveck Network Appliance Inc. S. Shepler, B. Callaghan. rfc3530 network file system (nfs) version 4 protocol., 2003. http://tools.ietf.org/html/rfc3530.

[23] Julian Seward. bzip2 and libbzip2, version 1.0.5 a program and library for data compression., 2007. http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.html.

[24] sostariffe.it. Velocità adsl: analisi della velocità media delle connessioni internet in italia., 2013. http://www.sostariffe.it/news/wp-content/uploads/2013/09/Osservatorio-velocit\unhbox\voidb@x\bgroup\let\

unhbox\voidb@x\setbox\@tempboxa\hbox{a\global\mathchardef\accent@spacefactor\spacefactor}\accent18a\egroup\spacefactor\accent@spacefactor-ADSL-in-Italia.pdf.

[25] R. Droms Bucknell University. rfc2131 dynamic host configuration protocol., 1997. http://tools.ietf.org/html/rfc2131.

[26] http://aria2.sourceforge.net/.

[27] http://doc.slitaz.org/en:cookbook:bootscripts.

[28] http://linux.die.net/man/8/kexec.

[29] http://sourceforge.net/apps/trac/aria2/wiki/UsageExample.

[30] http://sourceforge.net/apps/trac/aria2/wiki/XmlrpcInterface.

[31] http://wiki.ubuntu-it.org/AmministrazioneSistema/Chroot.

[32] http://www.erg.abdn.ac.uk/~gorry/course/intro-pages/uni-b-mcast.html.

[33] http://www.slitaz.org/en/.

[34] wiki.theory.org. (unofficial) bittorrent protocol specification v1.0. https://wiki.theory.org/BitTorrentSpecification.

6 Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni possibile errore contenuto in questa tesi.

Ringrazio anzitutto il Dott. Andrea Trentini, relatore della mia tesi, per la guida e il supporto fornitomi durante il mio lavoro e il Prof. Mattia Monga, co-relatore, per l'interesse dimostratomi. Ringrazio anche il Dott. Giorgio Biacchi per il suo supporto e la sua disponibilità durante la fase di test del mio lavoro in SiLab.

Ringrazio ovviamente tutti i ragazzi dell'auletta, coi quali studio (o ho studiato) regolarmente, che oltre a fornire un valido aiuto quando ce n'è bisogno, sopportano anche la mia voce un filo più alta della norma...

Un ringraziamento particolare va inoltre al mio amico Demian per il sostegno morale e per avermi aiutato a decidere di riprendere l'uni quando lavoravo e, senza dubbio, un enorme ringraziamento va alla Prof. Anna Maria Baldi, per la sua disponibilità e il suo aiuto durante la preparazione del mio esame di analisi matematica; l'ho apprezzato moltissimo.

Infine, ovviamente, ringrazio tutte le persone, parenti e non, che per un motivo o per l'altro mi sono state vicine; non posso elencarle tutte ma, per me, è come se l'avessi fatto.

A tutti, grazie.