Implementare NTP in Java

Sandro Tosi

Sommario

In questo documento, relazione del progetto di Programmazione di Reti e Laboratorio di Reti, verrà esposta un'implementazione in Java di parte del protocollo NTP (Network Time Protocol), il quale consente la sincronizzazione dell'orologio locale tramite l'interazione di un client con un insieme di server sparsi in tutto il pianeta. Verrà prima mostrato come viene gestito il tempo e come spesso l'importanza di avere un orologio ben sincronizzato sia fondamentale; verrà poi mostrato il protocollo NTP soffermandoci in particolare sugli aspetti che sono stati implementati e che saranno trattati sotto questo punto di vista nella sezione finale.

1 Gestire il tempo

Per trattare NTP che, come suggerisce il nome stesso, si occupa del tempo, è necessario introdurre le modalità con cui il tempo viene gestito sia a livello mondiale sia dal computer.

1.1 Il tempo per l'uomo

Ormai è noto, senza orologio non possiamo vivere: la nostra vita è infatti scandita proprio dal frenetico scorrere delle lancette. Anche in passato gli orologi venivano utilizzati e quando la Gran Bretagna era al massimo del suo splendore coloniale utilizzava un piccolo osservatorio astronomico per regolare l'orologio delle navi prima che partissero per un lungo viaggio: quell'osservatorio era situato a Greenwich.

Col passare degli anni questo nome è diventato simbolo dell'ora mondiale, in quanto venne stabilito che venisse usata la sua ora per regolare gli orologi in tutto il mondo (bastava sommare il fuso orario per ottenere l'ora locale, rispetto a Greenwich) e venne chiamato il *Greenwich Mean Time (GMT)* in quanto si basava sul tempo solare medio. Con l'avanzare della tecnologia si ottennero metodi più precisi per stabilire l'ora: l'orologio atomico. Questo strumento si basa su un fatto fisico, l'eccitazione degli atomi; durante la 13-esima Conferenza Generale dei Pesi e delle Misure nel 1967, venne stabilito che:

> "Il secondo è la durata di 9 192 631 770 periodi della radiazione emessa dall'atomo di Cesio 133 nella transizione tra i due livelli iperfini (F=4, M=0) e (F=3, M=0) dello stato fondamentale ^{2}S(1/2)."

(per una spiegazione dettagliata della definizione ci si veda `http://www.science.unitn.it/~labdid/sisint/si4_fondamentali/si_fondam.html`)

Era nato il primo orologio al cesio. Attualmente il più preciso orologio atomico al mondo è il NIST F-1 che si trova nei Phisycs Laboratories del complesso di Boulder, in Colorado, sotto il controllo del NIST (National Insitute of Standards and Technology) statunitense. Il fatto che sia il più preciso, non significa che sia anche l'unico al mondo, anzi, troviamo molti orologi atomici sparsi soprattutto nelle università e negli enti nazionali, come il nostro IEN Galileo Ferraris.

Attraverso l'utilizzo di oltre 60 orologi atomici sparsi in tutta la Terra, il 1 Gennaio 1972 è entrato in vigore il nuovo standard per il segnale orario: lo *UTC, Universal Time Coordinated*. *Universal* significa che può essere utilizzato ovunque, in quanto è indipendente dai fusi orari, mentre *Coordinated* significa proprio che diverse istituzioni collaborano attraverso le loro stime del tempo corrente alla costruzione di UTC.

L'interazione di questi 60 orologi atomici rende UTC estremamente preciso, un'accuratezza di circa 10 miliardesimi di secondo al giorno, molto di più di quanto era GMT. Questo poteva essere già un valido motivo per sostituire GMT a favore di UTC, ma un'altro fatto ha contribuito a questo cambiamento: GMT era basato su eventi astronomici, spesso poco precisi; il tempo medio solare è stato infatti provato variare notevolmente. GMT non era più una misura affidabile per applicazioni che ormai diventavano molto comuni, come la gestione dei satelliti in orbita attorno alla Terra, per i quali un minimo errore può significare uscire dall'orbita terrestre e disperdersi nello spazio.

Un altro problema doveva essere affrontato: basiamo il concetto di giorno sulla rotazione della Terra attorno al proprio asse (e quello di anno sulla rotazione della Terra attorno al Sole), ma questo moto rotatorio tende a rallentare. Effetti come maree o campi gravitazionali intensi tendono a far rallentare la rotazione terrestre. Dal momento che UTC è indipendente dalla velocità di rotazione della Terra, risulta a volte necessario inserire (o togliere) quelli che sono chiamati *leap second*, dei secondi "extra" per risincronizzare UTC con la rotazione terreste. Dal 1972 ad oggi ne sono stati aggiunti 22, circa uno ogni 18 mesi.

1.2 Il tempo per il PC

Nei primi calcolatori l'orologio era formato semplicemente da un oscillatore e da un contatore; un'idea giusta non si cambia, ed ancora oggi il tempo per il PC non è molto diverso da come era una volta.

In un computer esistono due orologi: quello hardware e quello software. Il primo funziona grazie ad una batteria ed è attivo anche quando il computer viene spento; il secondo, invece, funziona soltanto a PC acceso prendendo all'avvio il tempo dall'orologio hardware. Sebbene all'avvio questi orologi siano sincronizzati, durante il funzionamento del computer possono lavorare a velocità molto differenti, perdendo o guadagnando tempo in relazione l'uno con l'altro.

Purtroppo, l'orologio software non è molto preciso, vediamone le ragioni: il tempo viene contato attraverso degli interrupt, un messaggio spedito regolarmente dall'orologio interno; perchè il tempo sia aggiornato, la CPU deve catturare l'interrupt, elaborarlo e aggiungere una certa quantità di tempo (determinata dalla frequeza dell'interrupt), chiamata *tick*, al contatore del tempo. Spesso però, la CPU ha compiti più importanti da svolgere che aspettare gli interrupt dell'orologio, e questo comporta alcuni ritardi nell'esecuzione dell'aggiornamento. Inoltre, ogni variazione alla frequenza degli interrupt causa variazioni dell'orologio, rendendolo uno scarso orologio. Altro fatto importante è che non è in grado di mostrare tutti i valori possibili del tempo, in quanto è limitato dall'intervallo tra due succssivi interrupt: possono essere mostrati solo multipli di questo intervallo.

Anche l'orologio hardware presenta degli inconvenienti: come detto, funziona a batteria (anche se spesso viene utilizzato un accumulatore, ma l'idea rimane la stessa), ma cosa succede se questa si sta scaricando? La frequenza di aggiornamento aumenterà piano piano, portando l'orologio a segnare un'ora sbagliata. In aggiunta, questo orologio viene aggiornato ogni secondo, cioè non è in grado di rappresentare le frazioni di secondo.

L'orologio interno è solitamente un chip collegato ad un'oscillatore, generalmente un cristallo di quarzo, che genera interrupt dopo un certo numero di oscillazioni; la qualità del cristallo è spesso scarsa e questo comporta che siano molto sensibili alle variazioni di temperatura che portano il quarzo a vibrare più o meno velocemente, facendo dunque sbagliare anche l'orologio interno.

1.3 Qualità dell'orologio

Discutendo di orologi e della loro qualità, è facile incontrare in alcuni termini di uso comune in questo ambito, che abbiamo cercato di elencare qui sotto:

Risuluzione Il più piccolo incremento di tempo possibile.

Precisione Il più piccolo incremento di tempo possibile che può essere notato da un programma. Alternativamente, "la precisione è l'incertezza casuale di un valore misurato, espresso dalla deviazione standard o da un suo multiplo".

Jitter La variazione delle differenze di letture consecutive del tempo.

Accuratezza Determina quanto vicino è il nostro orologio ad un riferimento ufficiale di UTC. Alternativamente, "l'accuratezza è la vicinanza dell'accordo tra il risultato di una misurazione ed un valore corretto per il misurando".

Affidabilità Determina il tempo per cui un orologio riesce a mantenersi entro una specificata accuratezza.

Stabilità Determina quanto bene l'orologio riesce a mantenere una frequenza constante.

1.4 Perchè sincronizzarsi?

Pensiamo ad un appuntamento. Cosa succederebbe se gli orologi delle due persone segnassero orari diversi? Incontrarsi all'ora stabilita potrebbe essere problematico, in relazione alla differenza di orari ed alla pazienza del primo arrivato. Per alcune persone cinque minuti possono essere considerati "fisiologici" e quindi tollerati, ma per un computer, meno elastico di noi, questo ritardo potrebbe creare seri problemi.

La sincronizzazione temporale è di fondamentale importanza in modo particolare nell'interazione via rete: soltanto attraverso la condivisione del medesimo orario è possibile la coordinazione di eventi tra entità. È anche parte integrante di sistemi di sicurezza e crittografia: il logging richiede un preciso ordinamento temporale tra gli eventi in un sistema (in senso lato, si pensi per esempio ad un cluster), mentre per la crittografia, la sincronizzazione tra i comunicanti, consente l'uso e la validazione di messaggi temporizzati.

Vediamo in maggior dettaglio quali campi dell'informatica risultano maggiormente interessati, fino a risultarne vincolati, dalla sincronizzazione temporale:

Accuratezza dei log I file di log sono utili, in modo particolare, in caso di guasti o intrusioni in sistemi informatici, sia perchè rivelano questi eventi sia perchè consentono di seguire l'evoluzione del problema. Molto spesso capita che un server centrale gestisca i log di molte applicazioni differenti eseguite su alcuni server: si capisce bene che, per seguire l'evoluzione degli eventi, è necessario che gli orologi dei vari server siano sincronizzati, così da poter ottenere una coerente successione temporale degli accadimenti.

Monitoraggio In alcuni casi, gli amministratori di sistema utilizzano sistemi di monitoraggio remoti, come `RMON`, usualmente utilizzati per ricostruire le cause di un problema di rete. Dal momento che devono coordinare informazioni provenienti da una moltitudine di sorgenti, la sincronizzazione del tempo è di particolare importanza.

Calcolo distribuito Nel caso in cui, oltre a mantenere un log comune, diversi server collaborino alla soluzione di uno stesso problema, ci troviamo di fronte a quello che viene detto *calcolo distribuito*. È ormai molto frequente l'installazione di cluster di calcolo anche in piccole e medie imprese e quindi il problema di dover sincronizzare i server che partecipano al cluster, per evitare comportamenti inconsistenti, diviene ancora più attuale.

Diagnosi e recupero di attacchi di rete Con l'esplosione delle comunicazioni su Internet e con sempre più aziende che si affacciano sulla Grande Rete, aumentano i rischi di attacchi. All'aumentare della complessità della topologia della rete, aumenta anche l'esposizione a possibili attacchi. Si devono tenere sotto controllo diversi segmenti di rete, molti router, alcuni Access Point; si deve quindi centralizzare le informazioni provenienti da tutti questi punti in un unico nodo (sotto Linux si può utilizzare l'accoppiata `snort-logsnorter`) in modo da consentire un'analisi comparata delle informazioni, che dovranno essere quindi coerenti temporalmente.

Timestamp dei file I moderni file system mantengono diverse informazioni sui file che gestiscono e, tra queste, sono presenti la data di creazione, di modifica e molte altre. Quando il file system non risiede su una sola macchina (file system distribuito) oppure risiede su un'altra macchia (file system remoto), la data di modifica riveste molta importanza: per esempio NFS scarta un'aggiornamento di un file se la copia inviata risulta precedente a quella memorizzata in locale. Si capisce bene che avere (in questo caso) il proprio orologio in ritardo rispetto a quello del server NFS porta alla possibile perdita di tutte le modifiche apportate.

Autenticazione Windows 2000 utilizza come protocollo di autenticazione di default Kerberos, il quale utilizza anche parte dell'orario della workstation come parte del processo per generare il ticket di autenticazione. Se la differenza di tempo tra i domain controller supera la tolleranza concessa da Kerberos, potrebbe non essere possibile autenticarsi/collegarsi ad essi.

Inoltre, diversi protocolli prevedono l'utilizzo di un timestamp nei messaggi ed una finestra temporale di validità: se l'orario tra i due comu-

nicanti differisce in misura considerevole, i messaggi potrebbero venir scartati anche se validi.

Operazioni programmate Nei sistemi *nix `cron` e `crontab` svolgono il compito di eseguire determinati programmi ad un tempo specificato. Solitamente si tratta di operazioni lunghe e non interattive, come il backup dei dati che comunemente viene svolto in orari notturni. La sincronizzazione di un host è necessaria per garantire che le operazioni vengano svolte quando previsto. In caso di utilizzo di diverse macchine, la sincronizzazione diventa critica in quanto si deve garantire che le attività programmate siano coordinate correttamente.

Transazioni La necessità di sincronizzazione nella gestione delle transazioni non è una novità: fin dagli anni '60 IBM ha riconosciuto la criticità di questo compito nell'ambito della gestione di un gran volume di transazioni. La necessità di precisione dell'ordine dei decimi di secondo (a volte anche inferiore) è data dal fatto che è necessario porle nel corretto ordine di esecuzione, soprattutto nel caso siano quasi simultanee. Un esempio di quanto sia importante questo aspetto è dato dalle transazioni borsistiche: oggi giorno tutte le operazioni in Borsa vengono svolte telematicamente e se tutti i terminali non fossero correttamente sincronizzati con il sistema di gestione delle transazioni, i risultati potrebbero essere inconsistenti, per non dire catastrofici.

Sviluppo software Lo sviluppo software può essere un'attività altamente distribuita in quanto i team di sviluppo mantengono il codice su differenti server, spesso differenti anche geograficamente. I sistemi di controllo della versione (per esempio, CVS) vengono utilizzati per gestire la compilazione del codice; spesso, per decidere se ricompilare del codice, viene utilizzato il timestamp del file: nel caso non ci sia sincronia tra workstation e server CVS si potrebbero verificare delle incongruenze, in quanto la compilazione non utilizzerebbe la versione più aggiornata (di fatto verrebbe utilizzato il codice compilato in precedenza). Questo tipo di problema è difficilmente individuabile: infatti è spesso attribuito a problemi nel codice invece che alla cattiva amministrazione di questo.

E-Mail Ormai, lo vediamo ogni giorno, l'e-mail è diventato uno strumento indispensabile per comunicare, una vera rivoluzione. Ogni e-mail contiene il timestamp di quando è stata spedita, relativo al mittente; se questo timestamp è evidentemente errato, ciò può creare confusione nel destinatario.

Questo lungo elenco di applicazioni critiche rispetto al tempo ci dovrebbe far capire come la gestione dell'orario e della sua sincronizzazione, rispetto ad una fonte "sicura", sia realmente una necessità.

Il protocollo NTP (Network Time Protocol) è stato sviluppato proprio per questo compito: sincronizzare un host utilizzando un server in grado di fornire un orario il più corretto possibile.

2 NTP (Network Time Protocol)

In questa sezione vedremo in maggior dettaglio il protocollo NTP, mostrando come consenta la sincronizzazione del tempo; faremo comunque maggior riferimento al protocollo Simple (S)NTP, in quanto è da questo che è stato preso spunto per l'implementazione che discuteremo in seguito: diverse parti dei protocolli sono in comune ed una discussione delle differenze è presente nel seguito di questa sezione.

In quanto segue cercheremo di utilizzare le informazioni più aggiornate possibile: in questo momento ci troviamo nel pieno sviluppo nella nuova versione NTPv4 ancora non rilasciata ufficialmente, ma il cui sviluppo continua alacremente, quindi nuove feature possono essere aggiunte o possono essere modificate quelle presenti (come successo per il campo Reference Identifier e per l'autenticazione); per informazioni più dettagliate su NTP/SNTP, si consultino [RFC 2030], [RFC 1305], [RFC XXXX].

2.1 Introduzione

Come anticipato, NTP fornisce i meccanismi per la sincronizzazione e la distribuzione coordinata del tempo in una rete ampia ed eterogenea. Per ottenere ciò, una rete operante in una configurazione semi-organizzata e gerarchica di time server consente una loro mutua sincronizzazione e la ridistribuzione del tempo ai client che ne facciano richiesta.

Dal 1985, quando venne proposto per la prima volta NTP ([RFC 958]), molte modifiche, ma soprattutto molti miglioramenti, sono state apportate al protocollo, cercando sempre di mantenere la compatibilità con le versioni precedenti: con l'avanzare della tecnologia le necessità sono cambiate, era quindi necessario aggiornare gli algoritmi ed i modi di trasmissione (IPv6 in primis). Metodi più raffinati sono stati introdotti per calcolare l'entità della modifica all'orologio locale durante la sincronizzazione e per la scelta del miglior server a cui collegarsi. La grande attenzione che ormai si fa alla precisione del proprio orologio in molti campi rendono NTP lo standard de facto per la sincronizzazione del tempo su Internet.

Questo protocollo si basa su IP ed UDP, il quale fornisce un meccanismo di trasporto senza connessione; altri protocolli sono disponibili per gestire il tempo, come il Time Protocol o il messaggio ICMP Timestamp, ma NTP è specificatamente progettato per mantenere accuratezza e robustezza anche in presenza di una complessa rete sottostante.

2.2 La rete NTP

Come descritto in precedenza, ogni computer è equipaggiato con un orologio, anche se spesso di scarsa qualità. Esistono però degli speciali ricevitori che consentono di ricevere il segnale orario da diverse sorgenti, come ad esempio il GPS. Dato il costo e la convenienza di questo metodo, non è pensabile equipaggiare ogni calcolatore di questi apparati; è invece possibile equipaggiare alcuni computer in modo che agiscano come server primari del tempo, in grado poi di sincronizzare i restanti computer. Questo è il modo di funzionamento di NTP.

NTP è un protocollo distribuito per la sincroizzazione del tempo basato principalmente su un'architettura client-server: un client fa una richiesta di sincronizzazione ed il server risponde a questa richiesta.

L'organizzazione dei nodi avviene come un grafo ad albero: i server primari, quelli che si sincronizzano direttamente a qualche fonte esterna (GPS, orologi atomici, fonti radio), si trovano alla radice, detta *stratum 1*, mentre ai livelli successivi si trovano i server secondari ed i client; in figura 1 possiamo vedere uno schema della rete NTP con soltanto i primi tre strati della gerarchia:

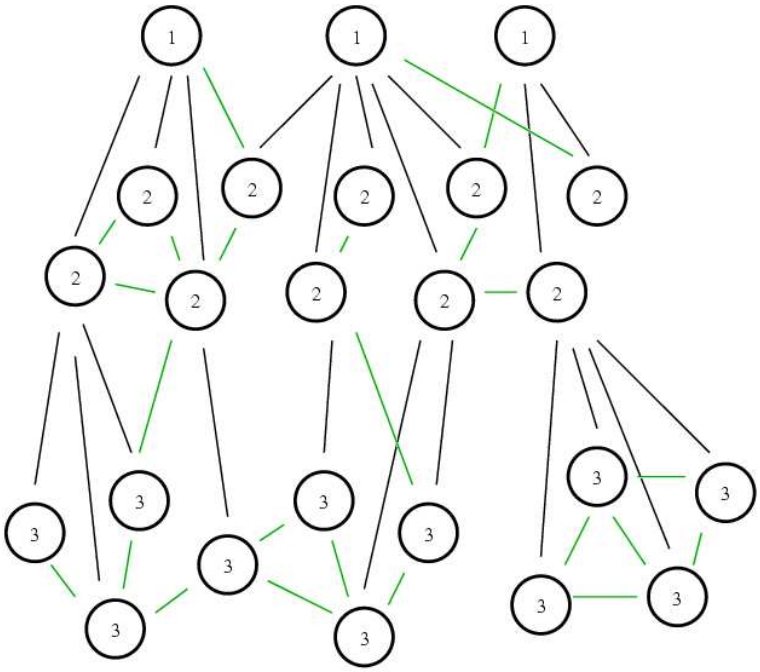

Figura 1: Schema della rete NTP con solo 3 livelli di nodi

L'organizzazione dei nodi prevista da NTP è la seguente: i nodi direttamente connessi a fonti di sincronizzazione esterne sono i server più precisi

disponibili nella rete NTP. Per mantenere questa precisione è anche necessario mantenere un carico di lavoro basso per questi server, per cui un numero adeguato di server stratum 2 si collegano ad essi, in modo da poter distribuire a loro volta il tempo attraverso NTP direttamente ai client che ne facciano richiesta: l'utilizzo di server stratum 1 da parte di client è espressamente sconsigliato.

I server secondari hanno una precisione minore dei server primari e questo degrado è dovuto ai percorsi di rete intrapresi dai pacchetti e dalla stabilità dell'orologio locale; questa gerarchia, però, consente di mantenere i server primari al massimo dell'efficienza. Possiamo quindi vedere lo stratum come un indice della precisione del server, dove numeri piccoli indicano maggiore precisione.

Una organizzazione come quella di NTP consente all'archietettura di scalare molto bene anche all'aumentare dei nodi: secondo [MINAR99], ci sono più di 175.000 nodi che eseguono NTP su Internet (sebbene siano stati registrati quasi 650.000 diversi nodi: il conteggio precedente annovera solo gli host che rispondono alle query, senza contare le singole workstation che si sincronizzano, i client insomma) dei quali circa 300 server primari (in realtà 957, ma solo 300 affidabili), oltre a più di 25.000 server stratum 2 ed 85.000 server stratum 3.

Il documento di Minar citato poc'anzi è una buona fonte per esaminare lo stato di salute della rete NTP: a fronte di un aumento vertiginoso degli host che utilizzano NTP per sincronizzare i propri orologi (se contiamo i contatti, dal survey compiuto da Guyton nel 1994, [GUYTON94], che ha registrato 15.000 macchine, quello di Minar ne ha contate oltre 640.000, un aumento di oltre 40 volte!) il numero di fonti primari è aumentato in maniera altrettanto considerevole (66 contro 957), ed anche la popolazione dei server secondari, stratum 2 e 3, si è molto accresciuta (rispettivamente da 1.400 a 26.000, da 3.300 a 85.000, approssimativamente).

Nonostante questo, la precisione di NTP nel sincronizzare il tempo è aumentata, sicuramente merito anche della moderna tecnologia, che consente la disponibilità di linee veloci in larga misura.

2.3 Il formato del timestamp NTP

Il formato che NTP (ed anche SNTP) utilizza per indicare il tempo è il seguente:

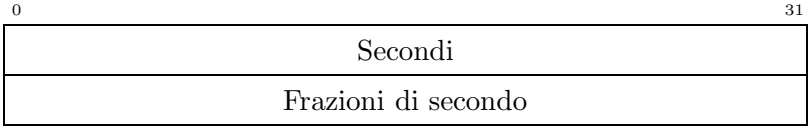

Figura 2: Formato del timestamp di NTP

un numero senza segno a 64 bit che rappresenta il numero di secondi trascorsi dal 1 Gennaio 1900. Come si vede, i primi 32 bit sono dedicati ai secondi, mentre i restanti 32 sono dedicati alle frazioni di secondo (eventualmente riempiti a destra con degli 0): questo formato consente di rappresentare $2^{32}-1$ secondi (corrispondente a oltre 136 anni: questo intervallo è chiamato *era NTP*) con una precisione di poco superiore ai 200 picosecondi (1 picosecondo $= 10^{-12}$ secondi).

Il fatto che NTP sia stato utilizzato proficuamente per 18 anni, fa supporre che possa essere utilizzato anche nel 2036, anno in cui il formato con cui il protocollo rappresenta il tempo andrà in overflow; per quell'epoca si dovrà trovare un modo per gestire i successivi timestamp. Una possibile soluzione, proposta dallo stesso Mills, è quella di considerare il primo bit del timestamp: se uguale ad 1, il timestamp si riferirà all'intervallo 1968*-2036*, mentre se 0, apparterrà all'intervallo 2036*-2104* (dove abbiamo indicato l'anno seguito da un "*" per indicare che l'intervallo non è relativo all'inizio dell'anno solare, ma che avrà inizio o fine in quell'anno, senza indicare il giorno preciso).

2.4 Il formato dati di NTP

NTP utilizza UDP per trasmettere i propri messaggi, utilizzando la porta 123. In figura 3 possiamo vedere il formato del pacchetto:

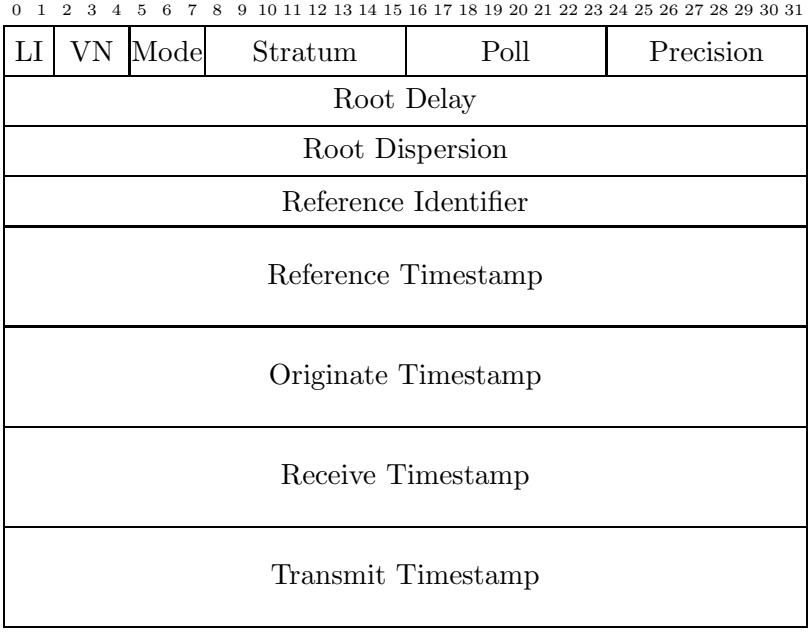

Figura 3: Il pacchetto di NTP

in questa figura non sono stati rappresentati i campi relativi all'autentica-

zione, in quanto NTPv3 utilizza un formato diverso da quello del futuro NTPv4. Diamo dunque una descrizione dei campi presenti nel messaggio NTP:

Leap Indicator (LI) Codice a due bit di avviso per un leap second da inserire o cancellare dall'ultimo minuto del corrente giorno. Si possono avere i seguenti casi:

00	nessun avviso
01	l'ultimo minuto ha 61 secondi
10	l'ultimo minuto ha 59 secondi
11	condizione di allarme (orologio non sincronizzato)

Version Number (VN) Un intero a tre bit che indica il numero di versione di NTP/SNTP. Un valore pari a 3 indica NTPv3, 4 indica NTPv4.

Mode Un intero a tre bit che indica la modalità della comunicazione, i cui valori sono definiti come segue:

0	riservato
1	attiva simmetrica
2	passiva simmetrica
3	client
4	server
5	broadcast
6	riservato per i messaggi di controllo NTP
7	riservato per usi provati

Stratum Un intero di otto bit che indica lo stratum dell'orologio locale, i cui valori sono definiti come segue:

0	non specificato o non disponibile
1	riferimento primario
2-15	riferimento secondario (via NTP/SNTP)
16-255	riservato

Nella versione 4 del protocollo, lo stratum 0 è un particolare messaggio inviato dal server al client chiamato Kiss-of-Death, utilizzato per informare il client che qualcosa non è andato come doveva; dal momento che NTPv4 è ancora in fase di bozza, questo messaggio verrà solo menzionato brevemente nella descrizione del pacchetto ed in nessun'altra parte.

Poll Interval Un intero di otto bit senza segno che indica il massimo intervallo tra due successivi messaggi in secondi come esponente del 2; indicando con PI il valore di questo campo, allora l'intervallo verrà calcolato come 2^{PI}.

Precision Un intero di otto bit con segno che indica la precisione dell'orologio locale in secondi come esponente del 2; indicando con PR il valore di questo campo, la precisione verrà calcolato come 2^{PR}.

Root Delay Un numero in virgola fissa a 32 bit con segno che indica il ritardo di roundtrip dalla sorgente di riferimento primario in secondi, con punto decimale fra i bit 15 e 16.

Root Dispersion Un numero in virgola fissa a 32 bit senza segno che indica l'errore nominale relativo alla sorgente di riferimento primario, in secondi, con punto decimale tra i bit 15 e 16.

Reference Identifier Una stringa di 32 bit che identifica la particolare sorgente di riferimento. In NTPv4, nel caso di stratum 1 è una stringa di quattro caratteri ASCII che identifica la fonte di sincronizzazione esterna (in stratum 0 rappresenta il messaggio spedito al client) mentre per stratum superiori è l'indirizzo IPv4 della sorgente di sincronizzazione.

Reference Timestamp L'ora in cui è stato impostato o corretto per l'ultima volta l'orologio locale, nel formato NTP.

Originate Timestamp L'ora in cui la richiesta ha lasciato il client per il server, nel formato NTP.

Receive Timestamp L'ora in cui la richiesta è arrivata al server, nel formato NTP.

Transimit Timestamp L'ora in cui la risposta ha lasciato il server per il client, nel formato NTP.

2.5 Modalità di comunicazione ed indirizzamento

Come visto trattando il campo **Mode** nella descrizione del pacchetto NTP, ci sono diverse modalità per la comunicazione tra client e server, vediamole più in dettaglio, indicando fra parentesi il valore del campo **Mode** corrispondente:

Attiva simmetrica (1) Un host che opera in questo modo, spedisce periodicamente dei messaggi, senza curarsi della raggiungibilità o dello stratum del nodo, annunciando il desiderio di sincronizzare ed essere sincronizzato.

Passiva simmetrica (2) Associazione generalmente creata all'arrivo di un messaggio da un nodo operante in modalità Attiva simmetrica e persiste fintantoche il nodo è raggiungibile ed operativo e lo stratum risulta minore o uguale a quello dell'host; in caso contrario, viene dissolta. Operando in questa modalità l'host annuncia il suo desiderio di sincronizzare ed essere sincronizzato.

Client (3) Un host che opera in questo modo, spedisce periodicamente dei messaggi senza curarsi della raggiungibilità o dello stratum del nodo; l'host annuncia il suo desiderio di essere sincronizzato, ma non di sincronizzare il nodo.

Server (4) Associazione generalmente creata all'arrivo di un messaggio di richiesta da un nodo operante in modalità Client e persiste solo per il tempo necessario a rispondere a questa richiesta, dopo di che viene dissolta. Operando in questa modalità, l'host annuncia il suo desiderio di sincronizzare, ma non di essere sincronizzato.

Broadcast (5) Un host che opera in questo modo, spedisce periodicamente dei messaggi senza curarsi della raggiungibilità o dello stratum del nodo. Operando in questa modalità (solitamente si tratta di un server su un LAN), l'host annuncia il suo desiderio di sincronizzare tutti i nodi, ma non di essere sincronizzato da alcuno di essi.

Diversi modi di indirizzamento sono possibili in NTP affinchè client e server comunichino:

Unicast La classica modalità punto-a-punto: un client unicast spedisce una richiesta ad un server al suo indirizzo unicast ed attende una risposta.

Multicast Modalità punto-a-multipunto: un server multicast spedisce periodicamente un messaggio destinato all'indirizzo di broadcast o al gruppo multicast locali sui quali sono in ascolto i client. Il gruppo multicast assegnato da IANA a NTP è 224.0.0.1.

Anycast Modalita multipunto-a-punto: un client anycast spedisce periodicamente una richiesta destinata all'indirizzo di broadcast o al gruppo multicast locali; uno o più server anycast rispondono alla richiesta con il loro indirizzo unicast: il client utilizza la prima risposta per continuare la comunicazione in modalità unicast.

La modalità che viene usata maggiormente è quella unicast, in quanto su Internet non è possibile utilizzare broadcast ed il multicast non è molto diffuso. In una LAN in cui sia presente un server NTP si può usare proficuamente le modalità anycast oppure la modalità multicast, in caso il numero dei nodi interessati alla sincronizzazione tramite NTP sia piccolo rispetto alla dimensione della rete.. Inoltre, la modalità anycast consente una sorta di autoconfigurazione del client: viene spedita una richiesta in broadcast (multicast) sulla rete locale a cui risponderanno dei server di cui non si sapeve l'esistenza; è quindi possibile impostare soltanto l'indirizzo di broadcast (multicast) locale e lasciare che il client si sincronizzi, a questo punto in modalità unicast, al primo server NTP che risponde. Una possibilità che è disponibile con questi due modi è quella di cambiare indirizzo IP al server NTP (o cambiare proprio server) senza dover riconfigurare tutti i client.

2.6 Il funzionamento di NTP

Finora abbiamo detto come il principale compito di NTP sia di sincronizzare l'orologio con uno più preciso, ma non abbiamo ancora descritto come è in grado di svolgere questo compito: cercheremo di vederlo adesso.

Un client, quando vuole sincronizzare il proprio orologio manda un pacchetto NTP visto sopra con tutti i campi a 0 tranne **VN**, **Mode** e **Transmit Timestamp** che imposta rispettivamente a 4 (la versione attualmente disponibile), 3 (modalità client) e l'ora in cui il pacchetto è stato spedito; da notare come non sia necessario che il suo orologio sia corretto. Impostare il valore del campo **Transmit Timestamp** è importante sia come semplice metodo per verificare che la risposta del server sia legittima sia per i calcoli che vedremo in seguito.

Il server, quando riceve un pacchetto di richiesta di sincronizzazione, copia dalla richiesta il campo **Transmit Timestamp** nel campo del messaggio di risposta **Originate Timestamp** e riempie i restanti campi in modo opportuno, con particolare attenzione per i campi timestamp. Quando la risposta del server è ricevuta, il client determina il **Destination Timestamp** come il tempo di arrivo in accordo col proprio orologio nel formato NTP.

Chiamando i timestamp in maniera abbreviata nel seguente modo

Timestamp	ID	quando viene generato
Originate Timestamp	T1	richiesta inviata dal client
Receive Timestamp	T2	richiesta ricevuta dal server
Transmit Timestmap	T3	risposta inviata dal server
Destination Timestamp	T4	risposta ricevuta dal client

possiamo definire due quantità molto importanti: il *roundtrip delay* d e l'*offset* t dell'orologio:

$$d = (\mathrm{T4} - \mathrm{T1}) - (\mathrm{T3} - \mathrm{T2}) \qquad t = \frac{(\mathrm{T2} - \mathrm{T1}) + (\mathrm{T3} - \mathrm{T4})}{2}$$

Spieghiamo di cosa si tratta:

Delay *d* Rappresenta il tempo che una richiesta impiega per raggiungere il server sommato al tempo che la risposta impiega per tornare indietro; la prima quantità della relazione rappresenta il tempo, relativo al client, tra la trasmissione della richiesta e la ricezione della risposta, mentre il secondo addendo rappresenta il tempo di elaborazione sul server della richiesta, prima che spedisca la relativa risposta.

Offset *t* Rappresenta la differenza di tempo tra due orologi e, in questo frangente, è l'ammontare di tempo da sommare al proprio orologio per sincronizzarlo con quello della sorgente di riferimento. Questo valore è il risultato più importante del protocollo NTP, poichè consente di ottenere la sincronizzazione del proprio orologio.

Si noti come entrambi questi valori siano con segno: però, mentre l'offset può normalmente avere segno negativo (l'orologio locale è in anticipo rispetto all'orologio di riferimento), un delay negativo è possibile soltanto nelle modalità simmetriche.

Una proprietà molto importante di questi due valori è che il loro calcolo non è influenzato dal cambiamento di *era NTP*: se l'*era* è la stessa per tutti i timestamp, allora verrà cancellata nelle equazioni; anche se il cambio di *era* avviene tra la richiesta e la risposta non si verificano problemi, in quanto la differenza tra T4 e T1 sarà molto elevata, ma si andrà a cancellare in entrambe le equazioni.

La sincronizzazione, per sua natura, necessita di lunghi periodi e molteplici comparazioni per ottenere e mantenere un orario accurato: l'accuratezza raggiunta è proporzionale al tempo speso per raggiungerla. In effetti, il calcolo proposto qui si riferisce al protocollo SNTP, mentre NTP presenta una suite di algoritmi per la selezione della sorgente di riferimento e per la correzione degli errori molto raffinata, ma che non vedremo (per una trattazione completa si veda [RFC 1305], mentre per una breve discussione si legga la sezione seguente).

2.7 Differenze tra NTP ed SNTP

Come dice il nome stesso SNTP è una versione semplificata del più completo NTP. Molto spesso non si ha bisogno dell'interno set di algoritmi fornito con NTP; cercheremo di dare una breve descrizione delle differenze tra i due protocolli, rimandando a [RFC 1305] e [RFC 2030] per una trattazione più esaustiva.

NTP prevede l'utilizzo di tecniche sofisticate per cercare di ottenere un valore il più preciso possibile. Ogni host sceglie una serie di server a cui collegarsi, NTP allora cercherà di costruire uno spanning-tree pesato, in cui la metrica consiste dello stratum più la *distanza di sincronizzazione*, cioè la dispersione sommata a metà del delay assoluto. Questo è fatto per cercare il percorso di sincronizzazione che impieghi il minor numero di server per raggiungere la radice.

NTP, inoltre, fornisce anche una serie di tecniche per ridurre gli effetti di errori statistici dovuti a fallimenti di componenti di rete, a imprecisioni delle sorgenti di riferimento o ai mezzi di propagazione: questa procedura di filtro è utilizzata per selezionare il miglior offset.

È importante osservare che, anche per NTP, non è sempre obbligatorio che questi algoritmi vengano utilizzati: mentre per un client che desidera mantenere la semplicità, è tollerato che non implementi queste tecniche, da un server che sincronizzi una comunità considerevole di client, ci si aspetta che questi metodi siano utilizzati, in maniera da mantenere la maggior accuratezza possibile.

Il fatto che SNTP non implementi queste tecniche, soprattutto lato ser-

ver, rende quasi obbligatorio utilizzare i server SNTP come stratum 1, cioè direttamente connessi con una fonte di sincronizzazione esterna scelta che consente di ridurre le operazioni da svolgere sul server.

Un fatto molto importante da notare è come i due protocolli possano coesistere tranquillamente, anche in versioni differenti: per un server NTP o SNTP, un client NTP o SNTP è indistinguibile; per un client NTP o SNTP, un server NTP o SNTP è indistinguibile.

3 Implementazione in Java

In questa sezione vedremo come una parte del protocollo SNTP sia stata implementata utilizzando Java. In figura 4 possiamo vedere il diagramma UML delle classi che verrà esplorato nel seguito di questa sezione.

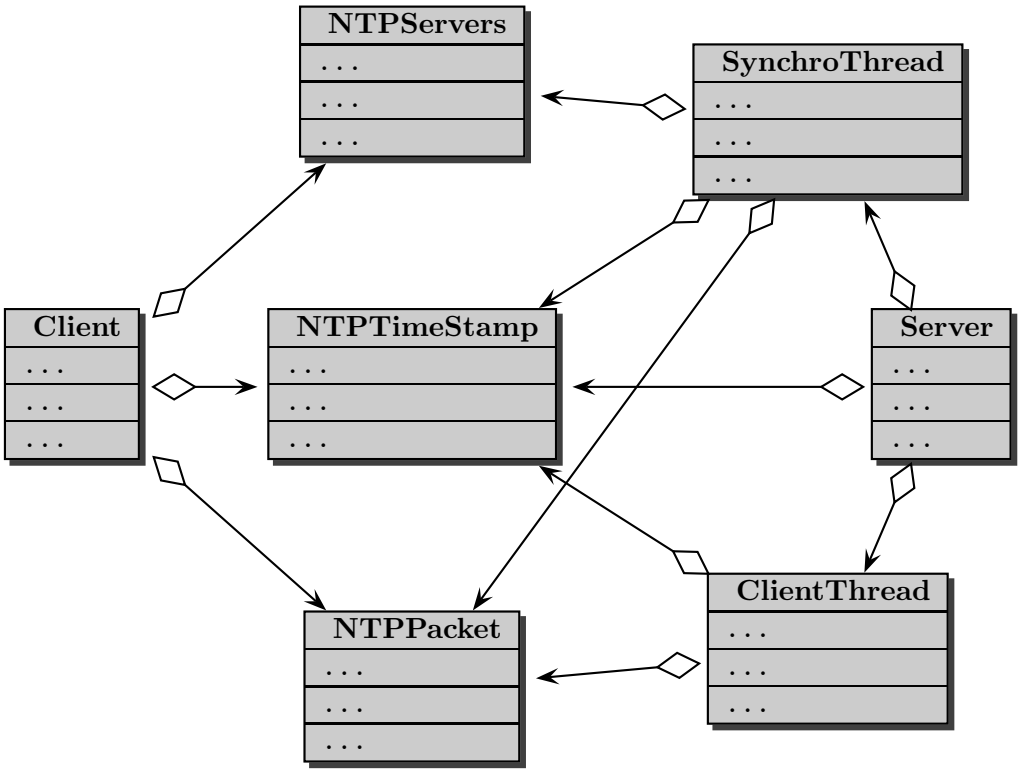

Figura 4: Diagramma UML delle classi

3.1 Assunzioni preliminari

Implementare l'intero protocollo NTP o SNTP che sia sarebbe stato un gran lavoro che sarebbe in buona parte stato vanificato da alcune caratteristiche intrinseche di Java: il fatto di non essere un linguaggio compilato in formato macchina, ma in bytecode, per poi essere interpretato dalla Java Virtual Machine (JVM), introduce un ulteriore livello di esecuzione diminuendo l'efficienza generale. Inoltre, se consideriamo che la precisione nella misura del tempo in Java raggiunge i millisecondi, mentre quella ottenibile utilizzando chiamate dirette al kernel raggiunge, in alcuni casi, ordini dei nanosecondi, ci porta a ritenere che molte parti del protocollo possano essere decurtate.

Per questo, gli algoritmi di selezione della sorgente di sincronizzazione e di controllo degli errori non sono stati implementati in quanto avrebbero introdotto maggiore complessità a fronte di un guadagno minimo in termini di precisione. La decisione, allora, di usare come base SNTP era quasi naturale.

3.2 Classi di utilità generale

Dal momento che alcune caratteristiche erano presenti sia nel client che nel server, si è scelto di incapsulare queste funzionalità in alcune classi generali, descritte qui di seguito.

3.2.1 La classe `NTPTimeStamp`

Questa classe è di fondamentale importanza, infatti è quella che si occupa di gestire il timestamp di NTP. Le date in Java iniziano i 1° Gennaio 1970 e questa classe si occupa di fornire le utility di conversione tra formato Java ed NTP e viceversa, oltre ad alcuni metodi utili nella creazione del pacchetto NTP.

3.2.2 La classe `NTPPacket`

Questa classe è deputata all'elaborazione di un pacchetto NTP in arrivo ed alla creazione di un pacchetto da spedire. Il costruttore che prende come argomento un array di `byte` (solitamente l'array letto dal `DatagramSocket` e che rappreseta la risposta del server) crea un oggetto, dal quale sarà poi possibile ottenere tutte le informazioni contenute al suo interno, tramite i metodi forniti da questa classe.

Sono inoltre presenti altri due costruttori: il primo, rivolto al processo client, riceve come argomento un oggetto della classe `NTPTimeStamp` (che diventerà il valore del campo **TransmitTimestamp**, l'orario di spedizione della richiesta) e restituisce un oggetto che incapsula una richiesta di sincronizzazione. L'altro costruttore, invece, prende una lunga lista di parametri ed è orientato al processo server, che deve creare una risposta alla richiesta

appena pervenuta ed i parametri sono quelli che andranno a riempire i campi del pacchetto NTP.

Infine, sono presenti due metodi utili per verificare che il pacchetto ricevuto sia corretto: uno è rivolto al server che vuole verificare la richiesta appena pervenuta, l'altro al client che vuole controllare che la risposta ricevuta sia relativa ad una sua richesta e che i dati in essa contenuti siano validi.

3.2.3 La classe `NTPServers`

Questa è una classe molto particolare. Quando abbiamo parlato del protocollo NTP si è anche detto che i nodi sono caratterizzati anche da un livello nella rete NTP, detto *stratum*. Possiamo trovare in [MILLS] una lunga lista di server straum 1 e 2.

All'interno di questa classe abbiamo inserito due array contenenti i server elencati da Mills, avendo cura di scegliere i nodi che si trovassero in Europa e che avessero una buona reazione alle richieste.

Due metodi statici consentono di ottenere un server primario o secondario scelto casualmente da questi elenchi.

3.3 Il client NTP e la classe `Client`

Il client ha il compito di collegarsi ad un server, calcolare l'offset tra l'orologio locale e quello del server tramite le informazioni fornite nella risposta e quindi correggere, se necessario, l'orologio locale.

Purtroppo, a causa di alcuni problemi con la JVM, la correzione dell'orologio locale non è stata implementata: il client restituisce il valore della correzione da apportare all'orologio, sarà poi l'utente che deciderà se e come modificare l'ora.

3.3.1 Switch di funzionamento

Alcuni opzioni sono state pensate per consentire di impostare alcuni valori di esecuzione e poter variare alcune parti del funzionamento del client:

-help Mostra una breve descrizione degli switch di funzionamento elencati di seguito.

-to ⟨**secs**⟩ Consente di impostare il timeout di attesa per una risposta da parte del server contattato; deve valere $5 \leq \langle \text{secs} \rangle \leq 60$ secondi; se non specificato, il timeout di default è 10 secondi.

-verbose Consente di impostare il livello di messaggi stampati a video: se non è presente viene stampato soltanto il valore di correzione dell'orologio locale, oltre a pochi altri dati ed agli eventuali messaggi di errore,

mentre se è presente, consente una visione delle informazioni presenti nella risposta del server contattato.

-p Consente di collegarsi ad un server stratum 1; di default, il client si connette ad un server stratum 2.

-s ⟨**server**⟩ Consente di collegarsi al server specificato in *server*.

-4 Imposta la modalità multiserver: il client si connette a 4 server, i primi tre stratum 2 e l'ultimo stratum 1, per ottenere un valore più preciso nella correzione da apprortare all'orologio locale.

-daemon ⟨**timeout**⟩ Imposta la modalità *daemon*: il client non termina l'esecuzione dopo il primo collegamento, ma continua a sincronizzarsi dopo un intervallo di tempo modificabile con *timeout*; deve valere $1 \leq \langle\text{timeout}\rangle \leq 60$ minuti, se non specificato, il timeout di default è 5 minuti.

Tra questi switch esiste una priorità di utilizzo, cerchiamo di spiegare come avviene la scelta delle opzioni:

- *-help* mostra le informazioni a video e dopo termina l'esecuzione;
- *-verbose* e *-to* sono indipendenti da ogni altro switch e quindi non partecipano a questo ordinamento;
- *-daemon* ha priorità maggiore rispetto a tutti gli altri switch; consente l'opzione *-p*;
- *-s* ha priorità maggiore rispetto a *-p* e *-4*;
- *-p* ha priorità maggiore rispetto a *-4*

La scelta delle opzioni avviene seguendo questa regola: se è presente un'opzione, quelle con priorità minore vengono ignorate silenziosamente (nessun messaggio a video viene stampato per avvisare di questo ignorameto).

3.3.2 Funzionamento del client

Risulta adesso semplice descrivere il funzionameto del client: una volta avviato il programma, eventualmente specificando le opzioni precendenti, il client seleziona il (o i) server a cui collegarsi, dopo di che esegue un metodo che scambia informazioni con il server corrente per ottenere un valore di *offset* e *delay*, in modo da poter correggere l'orologio locale.

3.4 Il server NTP e le classi `Server`, `SynchroThread` e `ClientThread`

La parte di maggior difficoltà (ed interesse) implementativa è naturalmente il server. Si è optato per un'implementazione multithreaded in modo da poter gestire richieste di sincronizzazione multiple che avvenivano in contemporanea al processo che mantiene la sincronia del server stesso.

In seguito descriveremo le classi che verranno eseguite come thread, per poi passare ad esaminare il funzionamento del server.

3.4.1 La classe `SynchroThread`

Come si è già evidenziato in precedenza, l'attività di sincronizzazione è un processo continuo, ed in particolar modo per un server, che deve mantenere il proprio orologio il più preciso possibile.

Questa classe è quella deputata a svolgere l'azione periodica di sincronizzazione dell'orologio, collegandosi ad un server primario scelto a caso dalla lista della classe `NTPServers`.

L'operazione principale del server è quella di fornire un servizio quando gli viene richiesto, in questo caso rispondere alle richieste dei client per sincronizzare i loro orologi. Tutto il resto, sono operazioni al contorno che, seppur necessarie, devono interferire il meno possibile con le operazioni di routine. Per questo, si è scelto di porre l'attività di sincronizzazone in un thread separato dal server, consentendo di continuare a servire i clienti e di sincornizzare l'orario in maniera indipendente.

Questa classe si comporta esattamente come un client: si collega ad un server, ottiene una risposta e da questa calcola i valori relativi al clock locale e li imposta sul server tramite un metodo messo a disposizione da quest'ultimo.

Per cautelarsi da eventuali problemi di sincronizzazione è stato anche utilizzato un algoritmo che pone il server fuori sincronia quando non si riesce a contattare un certo numero di server. Sembra una situazione rara, ma basti pensare ad un server NTP che si trova sul bordo di una LAN, magari proprio sul gateway, e che fornisce il proprio servizio alle workstation della LAN: se il collegamento verso l'esterno viene a mancare, non è possibile sincronizzare l'orologio del server, ma i client continueranno a fare richiesta per il suo servizio; porre il server fuori sincronia è una politica molto stringente, ma non si è voluto far continuare a lavorare il server in condizioni di impossibilità di aggiornamento.

3.4.2 La classe `ClientThread`

Un server NTP può ricevere anche un elevato numero di richieste al secondo e deve essere in grado di rispondere a tutte.

Per risolvere questo problema si è deciso di porre anche la gestione della risposta ad un client in un thread, definito appunto nella classe `ClientThread`.

Ogni volta che il server riceve un pacchetto di richiesta memorizza il tempo di arrivo e crea un nuovo thread che si occuperà di rispondere al client, consentendo al server di rimettersi subito in ascolto.

Al costruttore di `ClientThread` vengono passate tutte le informazioni che dovranno essere spedite al client; quando poi sarà posto in esecuzione, il thread controllerà se la richiesta ha un formato valido, nel quale caso preparerà il pacchetto che verrà spedito, dopo di che, il thread smetterà di funzionare.

La socket su cui vengono spediti i messaggi è condivisa tra tutti i `ClientThread`, è quindi stato necessario porre la chiamata a `send` in un blocco `synchronized`, in modo che il metodo possa essere invocato soltanto da un thread alla volta.

3.4.3 Switch di funzionamento

Anche per il server sono stati pensati alcuni switch in grado di modificare alcuni parametri di funzionamento, in maniera simile a quanto visto per il client. Vediamoli in dettaglio:

-help Mostra una breve descrizione degli switch di funzionamento elencati di seguito.

-verbose Consente di impostare il livello di messaggi stampati a video: se non è presente vengono stampati a video solo gli eventuali messaggi di errore, mentre se è presente, vengono mostrate le informazioni di esecuzione.

-to ⟨**secs**⟩ Consente di impostare il timeout di aggiornamento (l'intervallo tra un aggiornamento e l'altro); deve valere $5 \leq \langle \text{secs} \rangle \leq 60$ secondi; se non specificato, il timeout di default è 30 secondi.

-ft ⟨**num**⟩ Consente di impostare il numero di tentativi di sincronizzazione prima di invocare il metodo `outOfSync` del server, dichiarandolo fuori sincronia; deve valere $2 \leq \langle \text{num} \rangle \leq 30$ tentativi; se non specificato, il valore di default è 15 tentativi.

Per gli switch del server non è prevista alcuna priorità, in quanto non interferiscono tra di loro; l'opzione *-help* mostra le informazioni e non esegue il server.

3.4.4 Funzionamento del server

Il server fa uso delle due classi precedenti per gestire le richieste dei client e per mantenere il proprio clock aggiornato: è importante spendere alcune parole per descrivere come avviene questa attività.

Poichè non abbiamo voluto modificare l'orologio del calcolatore su cui veniva eseguito il server, si è pensato ad un modo alternativo per mantenere l'orologio aggiornato: dal momento che dall'attività di sincronizzazione si ottiene l'offset del clock locale rispetto al nodo corrente di sincronizzazione, quello che si è fatto è stato registrare questo valore e sommarlo ogni qual volta si avesse bisogno di prendere l'ora. Così facendo, avendo l'accortezza di importare il valore iniziale a 0, alla prima sincronizzazione si ottiene la correzione iniziale dell'orologio che lo porta vicino a quello della fonte di sincronizzazione. Dal secondo collegamento in poi, quello che otterremo sono delle correzioni rispetto a questo valore iniziale che ci portano alla modifica continua del valore dell'offset sul server, in modo da ottenere un valore sempre più preciso.

Il server dispone inoltre di due metodi importanti per l'aggiornamento del suo stato: il primo, `setUpdate`, consente di impostare i valori appena ottenuti dal `SynchroThread`, l'altro, `outOfSync` viene invocato da `SynchroThread` quando non è in grado di sincronizzarsi con nessun nodo e pone il server in uno stato in cui, sebbene ancora risponda ai client, indichi ad essi il suo stato di mancata sincronia, in particolar modo impostando `Stratum` a 0 e `LI` a 3.

Riferimenti bibliografici

[SKOOG] Paul Skoog. *The Importance of Network Time Synchronization.* TrueTime, Inc.. Disponibile presso `http://www.truetime.net/pdf/imp_netsync.pdf`.

[GRAHAM03] Glenn Graham. *Synchronizing Networks with NTP.* O'Reilly Network, Gennario 2003. Disponibile presso `http://linux.oreillynet.com/lpt/a/3076`

[MINAR99] Nelson Minar. *A Survey of the NTP Network.* MIT Media Lab, Dicembre 1999. Disponibile presso `http://www.media.mit.edu/~nelson/research/ntp-survey99/`.

[GUYTON94] James D. Guyton, Michael F. Schwartz. *Experiences with a Survey Tool for Discovering Network Time ProtocolServers.* USENIX Summer, Giugno 1994. Disponibile presso `http://citeseer.nj.nec.com/guyton94experiences.html`.

[RFC 958] David L. Mills. *RFC 958: Network Time Protocol (NTP).* M/A-COM Linkabit, Settembre 1985. Disponibile presso `http://www.faqs.org/ftp/rfc/rfc958.txt`.

[RFC 2030] David L. Mills. *RFC 2030: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI.*

Università del Delaware, Ottobre 1996. Disponibile presso `http://www.faqs.org/ftp/rfc/rfc2030.txt`.

[RFC 1305] David L. Mills. *RFC 1305: Network Time Protocol (Version 3) Specification, Implementation and Analysis*. Università del Delaware, Marzo 1992. Disponibile presso `http://www.faqs.org/ftp/rfc/rfc1305.txt`.

[RFC XXXX] David L. Mills. *RFC XXXX: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*. Università del Delaware, Agosto 2003. Disponibile presso `http://www.eecis.udel.edu/~mills/database/rfc/rfc-xxxx.pdf`.

[MILLS] David L. Mills. *The Network Time Protocol (NTP) Distribution*. Disponibile presso `http://www.eecis.udel.edu/~mills/ntp/html/index.html`.

[NTP.ORG] AAVV. *Network Time Protocol (NTP) project*. Disponibile presso `http://www.ntp.org/`.