UNIVERSITÀ DEGLI STUDI DI PISA

Facoltà di scienze matematiche fisiche e naturali

Dipartimento di Informatica

p2p sicuro mediante macchine virtuali

Tesi di laurea Triennale

Federico Tonelli

Tutore Accademico: Fabrizio Baiardi

Anno Accademico 2007/08

Ai miei nonni

Sommario

Negli ultimi anni si è sviluppato un grande interesse per la virtualizzazione, la tecnologia che permette di astrarre delle risorse per creare degli ambienti di esecuzione software, detti Macchine Virtuali. Le tecnologie di virtualizzazione offrono molti vantaggi nel campo della sicurezza, infatti, permettono di esaminare lo stato della memoria, dei registri del processore, o dei dispositivi I/O della macchina virtuale. Un altro tema di grande interesse è quello delle reti peer-to-peer che stanno sostituendo quelle client-server.
Il lavoro svolto dimostra come sia possibile sfruttare i vantaggi della virtualizzazione per poter creare una rete peer-to-peer sicura e che elimini peer maliziosi.

Indice

Elenco delle figure

Ringraziamenti

Per prima cosa vorrei ringraziare la mia famiglia, che mi ha permesso di raggiungere questo importante traguardo. Ringrazio Rita, che con pazienza e amore mi ha aiutato a superare tutte le difficoltà affrontate in questi anni. Stima e ringraziamenti vanno anche al Prof. Fabrizio Baiardi, per la sua disponibilità e per il suo supporto durante tutto il tirocinio. Ringrazio, Daniele Sgandurra e Dario Maggiari, per le idee e per i consigli che mi hanno dato, senza di loro sarei sempre alle prese con l'installazione di Xen. Infine, ringrazio tutti i gli amici che ho incontrato durante i corsi, per le ore passate con loro e per le cene alla *DABBE*, in particolare ringrazio Marco, mio grande amico, che mi ha sostenuto ed aiutato durante tutti questi anni.

Capitolo 1

Introduzione

1.1 Obiettivi del lavoro

Negli ultimi anni si è verificato un notevole aumento nel grado di sofisticazione degli attacchi verso i sistemi informatici. Ad esempio, i rootkits, gli strumenti software utilizzati dagli attaccanti per celare la loro presenza e per mantenere il controllo del sistema compromesso, si sono evoluti da quelli eseguiti a livello utente a quelli a livello kernel. In questo contesto, una tecnica per rilevare e contrastare le intrusioni è quella dell'introspezione. Questa tecnica, infatti, ci permette di analizzare lo stato di un sistema per ricostruire le strutture dati critiche utilizzate all'interno del sistema stesso e controllarne la loro consistenza.

Questa tesi definisce ed implementa uno strumento che controlli, mediante introspezione, il corretto comportamento di applicazioni p2p per la condivisione di file. In particolare, si considera Gnutella.

Lo strumento sviluppato, che d'ora in avanti chiameremo *attestatore*, deve rilevare e segnalare eventuali comportamenti anomali dovuti ad attacchi esterni, malware o spyware all'interno dell'applicazione stessa. Per esempio, deve segnalare tutti quei casi in cui il codice eseguito da un peer viene modificato a causa di uno stack o di un buffer overflow.

L'attestatore utilizza una tecnologia di virtualizzazione, Xen, per implementare in modo distribuito una analisi completa delle strutture dati e del codice dell'applicazione p2p.

Per integrare l'attestatore con Gnutella sono state fatte alcune modifiche al protocollo, in particolare, poichè si vuole garantire la sicurezza della rete, sono state modificate tutte quelle parti che prevedevano un'interazione tra i vari peer.

Grazie a queste modifiche e all'attestatore, la rete p2p creata sarà più sicura e più robusta rispetto agli attacchi che un nodo malintenzionato può portare, questo perchè i nodi che l'attestatore segnala come insicuri, verranno allontanati dalla rete.

Infine, abbiamo utilizzato la tecnologia di virtualizzazione, poichè questa ci permette di analizzare lo stato e la consistenza del kernel ad un livello

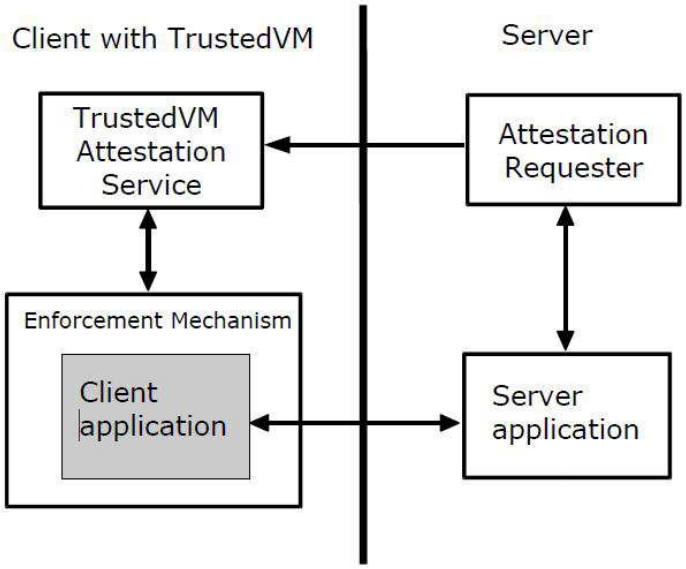


Figura 1.1: Architettura TrustedVM

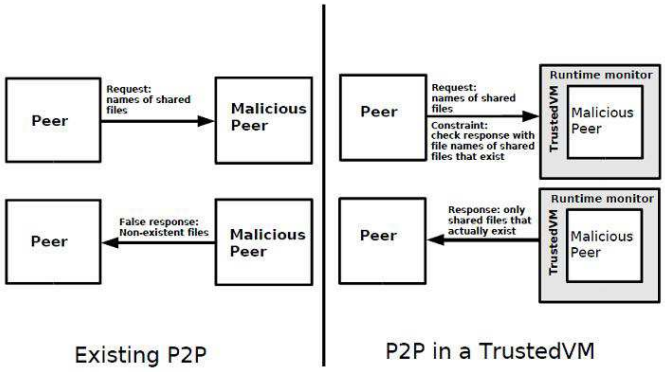


Figura 1.2: P2P con TrustedVM

inferiore rispetto a quello che un attaccante può ottenere, quindi per un peer malizioso diventa molto difficile evadere i controlli dell'attestatore.

1.2 Stato dell'arte

Attualmente, sono stati pubblicati molti lavori che trattano la sicurezza di un sistema attraverso la tecnologia di virtualizzazione e l'attestazione. In particolare, molti di questi studi utilizzano un TPM per garantire la sicurezza e l'integrità della macchina. Grazie alla virtualizzazione vengono create le cosidette ***Trusted Virtual Machine***. Tramite queste macchine vengono attestate le proprietà del codice che si sta eseguendo, come mostrato in figura 1.1.

Le trustedVM, quindi, possono anche essere utilizzate nell'architetture P2P; tramite l'attestazione remota un nodo può controllare esplicitamente

i requisiti di sicurezza, come mostrato in figura 1.2.

Altri approcci utilizzano protocolli basati sulla crittografia per assumere integra e sicura una macchina. Altri ancora sfruttano un "trusted" boot per garantire la sicurezza di un sistema, infatti il processo di boot termina nel caso in cui il controllo d'integrità fallisca.

Anche per quanto riguarda l'introspezione esistono molte tecniche presentate, tra queste citiamo *ReVirt* che supporta il recovery, il checkpoint e il roll-back delle MV e utilizza la tecnica chiamata *virtual-machine replay* per rieseguire un sistema, istruzione per istruzione, all'interno di una MV, per ripristinare il sistema dopo compromissioni. Un altro esempio è *XENKimono*, che rileva le violazioni della sicurezza del kernel di un sistema operativo a tempo di esecuzione, utilizzando le tecnologie di virtualizzazione per effettuare i controlli attraverso una macchina distinta da quella monitorata.

Infine, occorre considerare che l'evoluzione tecnologica nel campo dell'hardware ha notevolemente contribuito allo sviluppo di applicazioni che richiedono una maggiore potenza di calcolo. Senza contare che i numerosi benefici apportati dalla virtualizzazione, hanno spinto i produttori di componenti hardware a facilitare il processo di virtualizzazione delle risorse, tramite supporti hardware e firmware, ne sono un esempio l'INTEL-VT e l'AMD-VT.

1.3 Organizzazione della relazione

I capitoli successivi esaminano in maniera più approfondita i concetti presentati in questa introduzione.
La struttura dei capitoli è la seguente:

Capitolo 2 - Attestazione

Questo capitolo descrive i vari aspetti dell'attestazione, dal suo significato al suo utilizzo. Inoltre, analizza alcune tecniche di attestazione, come l'attestazione con crittografia o l'attestazione nel TPM.

Capitolo 3 - Peer-to-Peer

Questo capitolo analizza la storia e l'architettura delle reti peer-to-peer. In particolare, descrive le differenze architetturali tra Reti Centralizzate, Reti Decentralizzate e Reti Ibride. Inoltre, descrive una breve valutazione della sicurezza di queste reti e su alcuni tipi di attacchi.

Capitolo 4 - Gnutella

Questo capitolo descrive la storia e il protocollo di Gnutella, una rete Peer-to-Peer di condivisione di file aperta. Inoltre, spiega nello specifico i messaggi di Handshake, di Query e Query Hit, di Ping e Pong ed infine di Push

Capitolo 5 - Attestazione di reti p2p

Questo capitolo descrive l'implementazione dell'attestazione nel protocollo di Gnutella, oggetto del tirocinio. Inoltre, introduce la virtualizzazione, soffermandosi su Xen, un esempio di tecnologia di virtualizzazione. Infine, descrive la libreria d'introspezione utilizzata per implementare l'attestazione.

Capitolo 6 - Risultati Sperimentali

Questo capitolo descrive i test eseguiti per verificare le prestazioni e l'effecienza dell'attestazione introdotta nel protocollo di Gnutella.

Il progetto è completamente riproducibile. A tale proposito, sia l'Attestatore che le modifiche fatte al codice di Gnutella sono tutte documentate. Inoltre, sono stati utilizzati solo strumenti Open Source. Il codice realizzato per questo lavoro è pubblicato negli Appendici A e B, sempre sotto licenza Open Source, nella speranza che possa essere utile per studi futuri.

Capitolo 2

Attestazione

Con il termine sicurezza informatica si indica il ramo dell'informatica che si occupa della difesa da possibili rischi e violazioni sui dati. L'obiettivo della sicurezza informatica è quello di garantire l'affidabilità dei sistemi e dei dati, durante tutto il periodo di utilizzo dei dati stessi. Esistono due strategie di implementazione della sicurezza nei sistemi:

- **Passiva:** si basa sull'utilizzo di tecniche incentrate sulla difesa del software dagli attacchi.
- **Attiva:** rende i dati intrinsecamente sicuri, garantendone la **confidenzialità** e l'**integrità**.

Il problema della sicurezza informatica dei sistemi software, si è sviluppato con la crescita dell'utilizzo delle reti. Infatti, la straordinaria diffusione, verificatasi negli ultimi 3 anni, degli strumenti informatici e degli apparati di telecomunicazione che consentono un accesso ad internet, ha fatto si che il numero delle informazioni scambiate a livello mondiale sia aumentato vertiginosamente.

Bisogna considerare che internet è un canale privilegiato di intrusione e un moltiplicatore di possibilità di intrusione con un effetto esponenziale.

Solitamente, gli attacchi al codice di un processo si basano su *buffer o stack overflow*. Per questo, le principali strategie di rilevazione di un attacco sfruttano le sequenze di invocazioni al sistema operativo, generate dal processo, ed il valore dei loro parametri per definire i modelli che identificano la corretta esecuzione di un'applicazione. Esistono due tipi di analisi per la generazione di questi modelli.

Analisi Statica

L'analisi statica produce informazioni che permettono di valutare a *runtime* l'integrità dell'esecuzione di un processo. Infatti, un codice integro sarà definito da una serie *legale* di invocazioni al sistema operativo durante il suo tempo di esecuzione. Tutte le possibili sequenze sono definite, staticamente, da una grammatica che rappresenta il modello di riferimento a cui ogni processo deve attenersi, infatti, questo modello definisce tutte e sole le sequenze di invocazioni legali.

Ogni volta che il comportamento di un processo non è coerente con la grammatica, l'analisi lo interpreta come un possibile attacco all'integrità del processo o al sistema operativo stesso.

Oltre alla grammatica, l'analisi statica può produrre un insieme di asserzioni sul valore delle variabili del processo che possono essere utilizzate per definire ulteriori vincoli sulla correttezza dell'esecuzione.

L'analisi statica, comunque, è definita a partire da un'analisi del codice sorgente del processo, codice che non sempre è disponibile.

Analisi Dinamica

L'analisi dinamica, a differenza di quella statica, è di più semplice implementazione, però richiede una quantità maggiore di risorse per l'apprendimento e la memorizzazione delle informazioni relative al processo.

Questo tipo di analisi presenta anche un altro problema, infatti, può accadere che il modello prodotto possa essere poco accurato a causa delle condizioni particolari in cui è stato eseguito il processo durante la fase di apprendimento. Ad esempio, se durante la fase di costruzione del modello, il processo venisse attaccato con successo, il comportamento del processo verrebbe interpretato come corretto e quindi, successivamente, non verrebbe rilevato l'attacco.

La sicurezza all'interno dei programmi software, non solo deve effettuare l'eliminazione della produzione di danni irreparabili all'interno del sistema, ma anche gestire il controllo degli accessi da parte di possibili utenti maliziosi. Se la sicurezza del programma è garantita, è possibile parlare di affidabilità del programma stesso. L'Institute of Electrical and Electronics Engineers (IEEE) ha catalogato tutti gli errori software conosciuti, per poter garantire una risoluzione veloce. Questo ha incrementato la produzione di software protetto che garantisce l'assenza di condizioni conflittuali capaci di produrre errori al sistema. Lo sviluppo di sistemi di sicurezza sempre più avanzati ha portato ad un aumento della complessità delle tecniche di attacco.

2.1 Sicurezza nelle Reti

Una rete Peer-To-Peer (P2P) è stata definita come un'insieme di sistemi e applicazioni che implementano le funzioni critiche di distribuzione delle risorse in un ambiente decentralizzato. Sistemi che utilizzano una rete P2P sono sistemi come Napster e Gnutella.

Tra i vari aspetti delle reti P2P, è interessante trattare la sicurezza di questo tipo di reti e i servizi che devono essere garantiti per avere un sistema affidabile.

Le architetture per la sicurezza, associate ai modelli di riferimento delle International Organization for Standardization/International Telecommunication Union (ISO/ITU), sono un framework utile per la sicurezza della rete. Il sistema, per essere sicuro, deve essere governato da un insieme di

servizi di sicurezza creati a questo scopo e da meccanismi utilizzati per soddisfare questi servizi. L'insieme dei potenziali servizi di sicurezza è diviso in cinque aree fondamentali:

- Riservatezza
- Integrità
- Autenticazione
- Controllo degli accessi
- Non-ripudio

Il sistema P2P garantisce anche l'anonimato nel senso stretto del termine, cioè non ci possono essere meccanismi per rilevare l'identità dell'altro peer. L'anonimato nel P2P è caratterizzato da quattro distinte categorie: autore, server, distributore, documento. La mancanza di una di queste classi di anonimato violerebbe i diritti della privacy. In generale, le ricerche sulla sicurezza del P2P sono state ostacolate da molti problemi, tra questi, è stata rilevata la mancanza di un'identità stabile e l'incapacità di fornire servizi di autenticazione stabili, che, a loro volta, sono alla base della sicurezza dei servizi di controllo accessi, mantenendo la riservatezza e l'integrità della rete stessa.

2.1.1 Trusted Computing

Utilizzando un protocollo P2P, vogliamo che il nostro sistema sia capace di bloccare gli utenti maliziosi all'interno della rete. Ciò implica che, i client, siano in grado di individuare i peer che possono essere dannosi per il sistema ed isolarli dalla rete. Inizialmente i sistemi operativi venivano gestiti da una lista di affidabilità, oggigiorno però, i sistemi, sono completamente gestiti dagli utenti e per questo gli host non possono più fidarsi dell'affidabilità del sistema.

Con il termine affidabilità vogliamo indicare la sicurezza che la macchina esegua esattamente ciò che gli è stato detto di fare.

I possibili tipi di sistema esistenti sono:

- **Closed**: Un closed system può essere considerato affidabile quando è prodotto da una casa produttrice affidabile.
- **Open**: Un open system non sempre è considerato affidabile.

Per garantire l'affidabilità di un open system è necessario implementare il Trusted Computing (TC), una nuova tecnologia che ha come obiettivo quello di rendere i dispositivi, come computer e telefoni cellulari, più sicuri mediante l'uso di opportuni componenti hardware e software. Questa tecnica di protezione si è sviluppata basandosi sul concetto di crittografia simmetrica, con lo scopo di identificare ogni sistema tramite un codice univoco e di cifrare e firmare le informazioni tramite l'unica chiave di sistema.

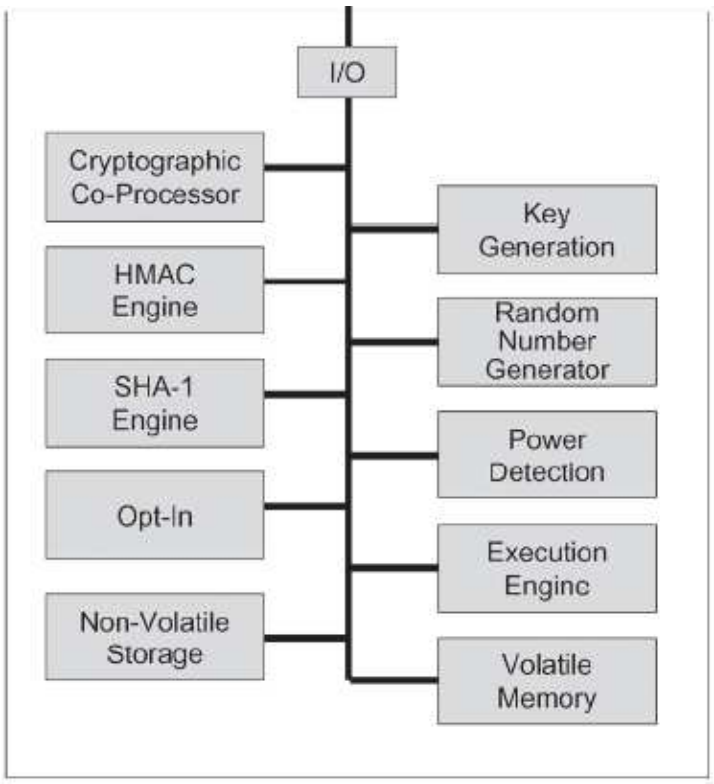


Figura 2.1: Architettura TPM

Per raggiungere tale scopo, è necessario utilizzare un componente hardware: il Trusted Platform Module (TPM)(Figura:2.1).

Il TPM è un chip dotato di una coppia di chiavi crittografiche, univoca per ogni chip, impossibili da modificare anche per il proprietario e capace di generare altre chiavi per la crittografia di dati o dispositivi. Normalmente, il TPM, è fissato sulla scheda madre del pc.

La natura di questo dispositivo è far si che i dati memorizzati siano protetti, sia da attacchi da parte di software esterni, sia da attacchi fisici. Il TPM ha cinque fondamentali capacità:

Endorsement Key (chiave di approvazione): garantisce l'integrità di sistema utilizzando due chiavi RSA a 2048bit che identificano univocamente il TPM. Questa coppia di chiavi viene generata al momento della produzione del chip.

Secure I/O (Input/Output sicuro): le operazioni di input/output vengono rese sicure tramite meccanismi di cifratura a chiave simmetrica. Tutte le informazioni che transitano su bus del sistema sono cifrate.

Memory curtaining (separazione della memoria): prevede che l'hardware impedisca ad ogni programma la lettura e la scrittura di zone di memoria utilizzate dagli altri applicativi in esecuzione.

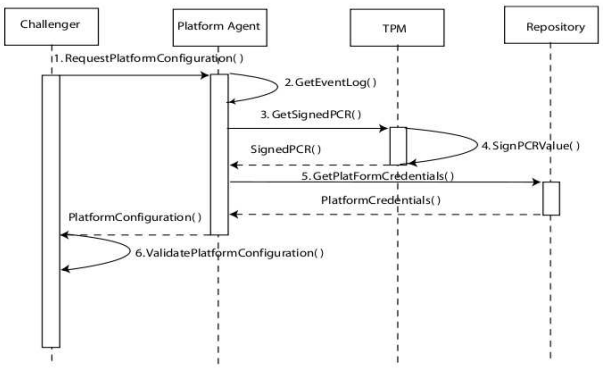


Figura 2.2: Protocollo di attestazione

Sealed storage / Trusted storage (memoria sigillata / memoria fidata): protegge le informazioni per mezzo della cifratura usando una chiave che deriva dallo stato del sistema. Questo vuol dire che tali informazioni possono esssere fruibili solo con la stessa combinazione di software ed hardware.

Remote attestation (attestazione remota): permette a terzi di interrogare il sistema e di conoscerne lo stato. Ciò permette di evitare che informazioni riservate siano inviate da un computer compromesso o non sicuro.

2.1.2 Attestazione remota nel TPM

Quando un verificatore vuole ispezionare la configurazione della piattaforma di un host, richiede un insieme specifico di valori del Platform Configuration Register (PCR), che contiene le firme rappresentative della sequenza di eventi avvenuti su quella piattaforma. La sicurezza della validità dei valori del PCR è garantita tramite il valore della firma sul registro prodotto dal TPM.

Come si è detto, la tecnologia TPM, utilizza una componente con una coppia di chiavi simmetriche private chiamate Attestation Identity Key (AIK) per migliorare le operazioni di autenticazione. Con la piattaforma Trusted Computing Group (TCG), la coppia di chiavi del AIK viene usata come alias per un altra coppia di chiavi Endorsement Key (EK).

Questo meccanismo può utilizzare differenti AIK ogni volta in cui valuta i valori di PCR, rendendo tali attestazioni non rintracciabili.

La ragione per cui viene utilizzato questo meccanismo anonimo è che l'EK identifica univocamente una piattaforma e, potenzialmente, il gestore della piattaforma.

Come evidenziata in fig. 2.2, il protocollo prevede i seguenti passi:

- Un Challenger desidera ispezionare uno o più dei valori PCR presenti su una piattaforma.
- Un Platform Agent rileva che le voci corrispondenti nell'SML sono la richiesta dei valori PCR.
- Il TPM invia alla Platform Agent i valori PCR richiesti.
- Il TPM attesta i valori contenuti nella richiesta PCR prucendo una firma mediante l'AIK.
- La Platform Agent unisce le credenziali che servono per il TPM. I Valori del PCR autenticati con le loro corrispondenti entrate SML e le informazioni rilevanti vengono rimandati al Challenger.
- Il Challenger verifica i dati risultanti. La raccolta dei valori viene calcolata partendo dai valori nelle posizioni del SML e comparata con il valore delle firme del PCR. Le credenziali della Platform vengono valutate e le firme vengono controllate.

L'attestazione remota presenta alcuni problemi: non viene fornita alcuna informazione riguardo al comportamento del programma, è un protocollo statico e non è flessibile in quanto è senza aggiornamenti o *patch*.

Analizziamo in dettaglio i problemi che sorgono con l'utilizzo dell'attestazione remota.

L'attestazione remota certifica quale particolare codice binario è in esecuzione sulla macchina remota, ma i programmi attestati possono essere insicuri o possono avere errori.

Essendo statica, l'attestazione non può trasmettere informazioni dinamiche, come, ad esempio, lo stato dei programmi a runtime o le proprietà dei programmi in input. Per quanto riguarda gli aggiornamenti e le patch, il verificatore utilizza una lista approvata di software, occorre anche considerare che la maggior parte di questi software ha bisogno di un flusso continuo di patch e aggiornamenti.

Vi è un tipo di attestazione, chiamato Direct Anonymous Attestation (DAA), che utilizza molte delle tecniche crittografiche più sofisticate per mantenere la privacy degli utenti, senza introdurre richieste di speciali garanzie da terze parti.

Il DAA è un protocollo crittografico che permette l'autentificazione remota di un computer che rispetta le specifiche del TCG, preservando l'anonimato dell'utente. Questo protocollo prevede che, per l'autenticazione corretta, siano necessari tre dispositivi, un elaboratore dotato di TPM, l'abilitatore DAA e il verificatore DAA, e due fasi differenti.

Nella prima fase l'abilitatore deve verificare che l'elaboratore dotato di TPM non sia stato compromesso. Terminata questa prima fase fornisce le credenziali DAA all'elaboratore che, verranno utilizzate nella fase di firma.

Usando un protocollo di tipo zero-knowledge, il verificatore può controllare la bontà delle credenziali, senza conoscere l'identità del possessore. E' possibile anche usufruire della funzione di revoca dell'anonimato. In questo caso prima della fase di firma, si può scegliere una terza parte che, in caso di TPM compromesso sarà in grado di identificare il TPM.

2.1.3 Attestazione remota

L'attestazione remota è il metodo che una applicazione può usare remota per autenticarsi, o per verificare in remoto l'autenticità di una applicazione. Come indica il nome, l'attestazione remota sfrutta una connessione di rete e, sapendo che le reti di connessione sono gestite dal sistema operativo, non possiamo fidarci del protocollo di attestazione senza fidarci del sistema operativo stesso. Esistono vari metodi per implementare l'attestazione remota, qui di seguito illustreremo i più utilizzati.

2.1.3.1 Attestazione con crittografia

La via più comune per implementare l'attestazione remota è utilizzare la crittografia. La parte validante calcola un codice hash dell'immagine dell'applicazione che deve essere convalidata. Questa parte, in generale il sistema operativo, firma questo hash e il richiedente confronta l'hash ricevuto con un hash di un'immagine valida nota.

Il protocollo può essere rafforzato con un *bit commitment*, per garantire che il calcolo del codice hash sia stato effettivamente eseguito. Ma questo, non garantirebbe che il codice hash sia stato calcolato dall'immagine del programma in esecuzione. Quindi risulta evidente la necessità di fare affidamento sul sistema operativo.

Nel contesto del P2P i codici hash sono problematici. Il codice hash risultante è dipendente dall'immagine del programma, quindi con l'utilizzo di piattaforme diverse e diverse versioni di software possono risultare codici hash diversi.

2.1.3.2 Attestazione semantica

L'attestazione semantica è un concetto proposto da Haldar, Chandra e Franz per superare le carenze del sistema tradizionale di attestazione basata sulla crittografia. Haldar, Chandra e Franz propongono un metodo che consente, tramite macchine virtuali, l'attestazione basata sulle proprietà di una applicazione ad alto livello, su una piattaforma indipendente. Questo lo rende il miglior metodo da utilizzare su piattaforme P2P. L'attestazione con crittografia può essere utilizzata per verificare alcune particolari versioni di una applicazione in esecuzione sull'host, ma sarebbe più interessante se l'applicazione in questione rispettasse alcune politiche di sicurezza. Attraverso l'utilizzo di una macchina virtuale e un codice in byte, della piattaforma indipendente, che include anche informazioni di alto livello riguardo l'applicazione, come ad esempio un java byte code, gli autori indicati hanno defini-

to un sistema che consente l'attestazione delle applicazioni. L'attestazione Semantica consente:

- **Attestazione delle proprietà delle classi**
- **Attestazione delle proprietà dinamiche**
- **Attestazione delle proprietà arbitrarie**
- **Attestazione delle proprietà di sistema**

L'attestazione semantica permette di superare i problemi più critici dell'attestazione tramite crittografia. Questo tipo di attestazione può essere usata per garantire che l'applicazione e il suo ambiente soddisfino le condizioni necessarie per il comportamento atteso e, visto che il sistema è una piattaforma indipendente e la verifca viene eseguita sulla base delle proprietà dell'applicazione, diverse versioni dell'applicazione non provocano ulteriori problemi. Naturalmente, questa soluzione è corretta solo se le applicazioni non sono state modificate troppo da versione a versione.

Considerando una rete P2P, l'attestazione semantica è nettamente migliore rispetto a quella con crittografia. Attraverso l'utilizzo dell'attestazione semantica, la rete P2P è capace di prevenire personificazioni e situazioni in cui i peers possano fornire contenuti corrotti, prevenendo così i due più grandi problemi.

2.1.4 Attestazione remota nelle reti P2P

Una rete P2P che non può validare i suoi peer e isolare quelli dannosi è alla mercè di un qualunque malintenzionato. Un aggressore può sfruttare la rete per futuri attacchi, oppure diffondere nella rete pacchetti falsi, con false informazioni.

Una tipica situazione di attacco si ha quando l'aggressore è esso stesso parte della rete P2P e vuole sfruttare la buona volontà della rete. Questo tipo di attacco costringe, per esempio, molti host a mandare dei pacchetti di risposta molto lunghi a destinatari sbagliati.

I modelli dell'attestazione possono non risolvere il vincolo della rete P2P di essere decentralizzata e non gestita, ma possono essere molto utili se è già esistente una lista di peers affidabili. In questo caso la lista può segnalare ai peer autorizzati l'aggressore e quindi causare l'isolamento del peer illegale.

L'attestazione semantica, come è stata proposta da Haldar, Chandra e Franz, è un metodo soddisfacente per essere utilizzato in una rete decentralizzata P2P, come quella di Gnutella.

Capitolo 3

Peer-to-Peer

3.1 La storia

All'inizio del 1999, Shawn Fanning, uno studente universitario della Northeastern University di Boston, diede vita ad uno strumento chiamato Napster. Fanning concepì Napster come un servizio che potesse permettere agli utenti di elencare, condividere file in formato mp3 e scaricare tali file attraverso la rete Napster stessa. Il computer centrale avrebbe mantenuto una lista dei file che gli utenti intendevano condividere e tale lista sarebbe stata poi aggiornata dal software degli utenti stessi attraverso il log on ed il log off dal sistema. All'indomani dell'implementazione dello strumento di Fanning, la rete Napster è stata usata da 30 milioni di utenti, con oltre 800 mila nodi che effettuavano un accesso simultaneo. L'unico limite superiore sul numero di utenti di Napster era imposto dalla banda passante del computer centrale,che limitava il numero possibile di utenti che potevano accedere al server Napster ed utilizzare i servizi da esso forniti.

A causa però delle violazioni delle leggi di copyright e per conseguente ordine della Corte Suprema degli Stati Uniti, Napster fu costretto a imporre delle complesse limitazioni sui file condivisi, degradando pertanto la qualità del servizio. Infine, nel luglio 2001, Napster dovette chiudere definitivamente i suoi server.

Al di là delle violazioni di copyright, uno dei motivi per cui la rete Napster fallì nel fornire una elevata qualità del servizio (QoS) ai suoi utenti è dovuto alla sua caratteristica di rete peer-to-peer con forti elementi di centralizzazione.

Infatti, l'utente inviava la propria richiesta al server Napster, il quale eseguiva una ricerca nel proprio database; una volta trovata la lista di host da cui scaricare il file richiesto, il server la inviava all'utente che aveva fatto la richiesta. A questo punto, l'utente si connetteva direttamente ad uno degli altri utenti presenti nella lista ricevuta.

Avendo un singolo punto d'accesso, il server Napster, fa sì che l'intera rete collassi nel caso in cui tale punto di accesso sia disabilitato. Non solo: il server Napster ha il totale controllo di tutti i dati che vengono scambiati in rete ed è il solo responsabile dei loro contenuti.

Una soluzione per fornire una migliore qualità del servizio nell'ambiente di reti peer to peer è quella di usare invece un modello decentralizzato, che preveda quindi molteplici punti di accesso.

In questo modo se alcuni di tali punti di accesso sono disabilitati, la rete può continuare a "sopravvivere".

Uno di questi modelli peer to peer è quello della rete di Gnutella, basato sull'omonimo protocollo.

3.2 Che cos'è una rete peer-to-peer

Prima di analizzare nel dettaglio Gnutella, vediamo dunque che cosa intendiamo con *peer* e reti *P2P*
Una rete P2P è un tipo di rete che permette ad un gruppo di utenti di computer di interconnettersi e di condividere le risorse in loro possesso. All'interno di tali reti, ogni computer (*peer*) ha capacità e responsabilità equivalenti. Ciò differisce dalle architetture client/server in cui alcuni computer (*server*) sono dedicati al servizio di altri computer (*client*).

3.3 Architettura di una rete peer-to-peer

Una rete P2P è dunque un tipo di rete che permette agli utenti che possiedono un certo tipo di applicazione di connettersi fra loro e accedere direttamente ai file di ognuno. Un altro aspetto fondamentale di questo tipo di reti è che la condivisione di file e lo scambio diretto fra un utente e l'altro, rende inutile l'utilizzo di un server centrale in cui memorizzare i dati e le risorse. Da quanto detto finora, possiamo distinguere tra due approcci generali nell'architettura delle reti peer-to-peer: reti centralizzate e decentralizzate.

Noi ci occuperemo con particolare attenzione alle reti decentralizzate e daremo solo una definizione di quelle centralizzate.

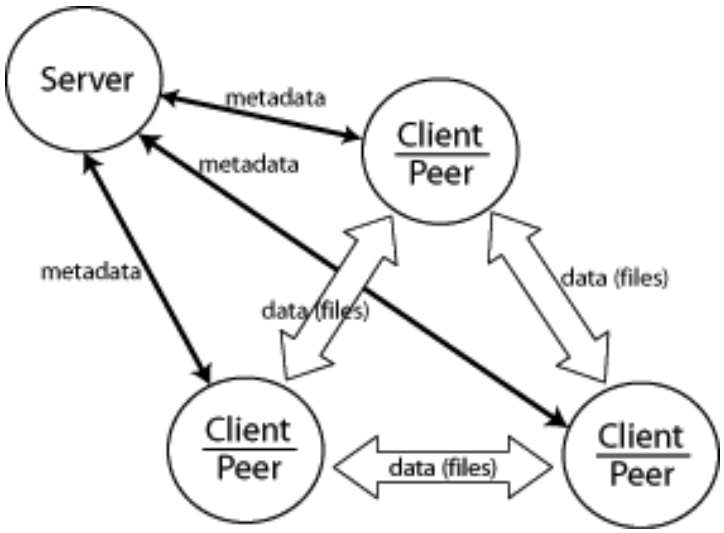


Figura 3.1: Architettura Reti Centralizzate

3.3.1 Reti Centralizzate

Questa categoria di reti P2P, cui appartiene Napster, utilizza un sistema centrale che dirige il traffico fra i singoli utenti. I server centrali mantengono delle directory che memorizzano indicazioni sui file condivisi dagli utenti. Tali server consentono agli utenti di accedere ai propri database e forniscono le indicazioni attraverso le quali l'utente che ha fatto una richiesta può stabilire una connessione diretta con un utente che in quel momento condivide il file cercato. I server non contengono comunque dei dati di alcun tipo, servono solamente a far comunicare i vari nodi, come era il caso di Napster, oppure a raccogliere le varie informazioni per poterle poi rielaborare per secondi fini. Questo avviene in reti che condividono le risorse hardware dei vari nodi. (Figura:3.1)

3.3.2 Reti Decentralizzate

Architetture di questo tipo non comprendono server che gestiscono i file condivisi dagli utenti della rete. Ogni nodo quindi ha una sua autonomia ed è libero di scegliere quali e quante risorse condividere. Questo porta anche ad una robustezza della rete anche dal punto di vista legale. Infatti, in questo tipo di reti l'anonimità di coloro che condividono i file è sostanzialmente garantita, pertanto non è possibile perseguire gli utenti o questo tipo di reti. Inoltre, la mancanza di un server centrale fa sì che la rete resti sempre connessa, poichè non c'è dipendenza da un server che introduca colli di bottiglia e sia soggetto a flooding di richieste da parte degli utenti. (Figura:3.2)

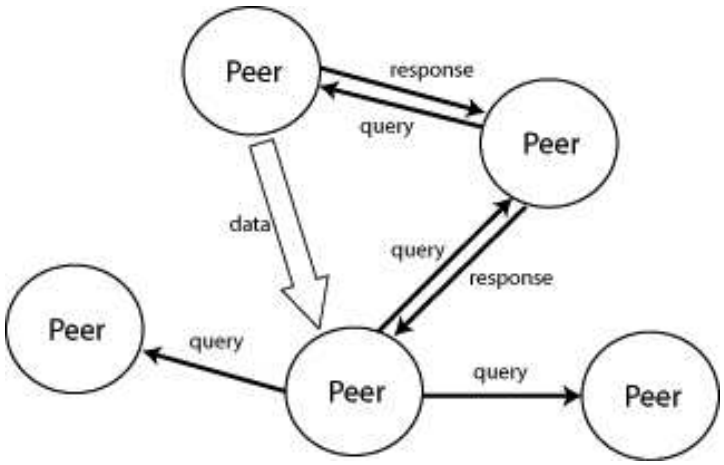


Figura 3.2: Architettura Reti Decentralizzate

Questa architettura offre quindi i seguenti vantaggi:

- Sono presenti più punti di accesso e quindi reti di questo tipo non soffrono piu la presenza di colli di bottiglia.
- E' piu robusta sia dal punto di vista legale che da quello topologico.

Quest'architettura ha comunque uno svantaggio, quello di non rendere accessibile l'intera rete a ciascun nodo. Di conseguenza, non è sempre possibile rispondere ad una richiesta, anche se qualche node della rete potrebbe soddisfarla.

3.3.3 Reti Ibride

Sono state sviluppate delle reti P2P ibride, che presentano cioè entrambi gli aspetti dei modelli centralizzati e decentralizzati. In sostanza si tratta di reti che hanno una struttura decentralizzata ma che comprendono al loro interno sottoreti centralizzate.

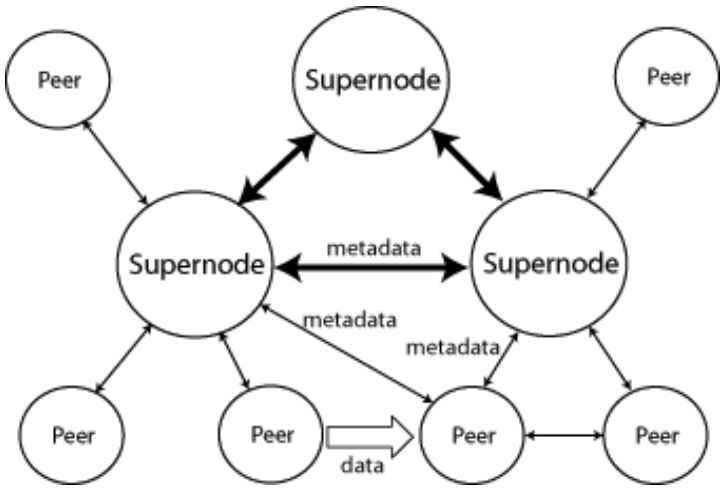


Figura 3.3: Architettura Reti Ibride

Uno dei software che utilizza questo tipo di struttura è eMule. La maggior parte dei peer sono collegati in qualche modo ad una specie di supernodo, chiamato SuperPeer o UltraPeer a seconda del software utilizzato, a cui vengono inoltrate tutte le richieste di risorse di qualunque tipo. Un SuperPeer una volta ricevuta la richiesta la inoltra, gli altri SuperPeer a cui è collegato con una struttura totalmente decentralizzata. In questo modo, qualunque client può ricevere una risposta positiva ad una richiesta anche se l'oggetto richiesto risiede fisicamente su un peer diverso. (Figura:3.3)

3.4 Gli attacchi e la sicurezza nel P2P

3.4.1 Gli attacchi

Nonostante l'apparente robustezza e tolleranza ai guasti anche le reti P2P possiedono vulnerabilità che possono comprometterne l'intera struttura. Le reti decentralizzate, anche se non richiedono la presenza di un nodo per registrarsi, come avviene nelle reti centralizzate, sono comunque soggette ad alcuni tipi di attacchi. In particolare, la rete Gnutella soffre di un paradosso. Infatti se da un lato richiede un elevato numero di utenti affinchè la qualità del servizio sia sempre elevata, dall'altro questo numero la rende sempre più rilevante agli occhi della censura e quindi un obiettivo appetibile a fianco delle reti P2P centralizzate.

Un tipo di attacco effettuato su queste reti è lo *spamming*.
Lo spamming è un problema ben conosciuto nel mondo della posta elettronica ma, mentre un e-mail contente spam si può cestinare o filtrare, questo non può avvenire nelle reti P2P.

Tra le varie tecniche di attacco utilizzate ne citiamo alcune:

- Una tecnica utilizzata dai nodi che intendono fare spamming è di fornire sempre e comunque una risposta a qualunque tipo di richiesta da loro inoltrata, inducendo il nodo che ha fatto la richiesta a scaricare i dati desiderati presso un nodo sotto attacco.

- Un secondo tipo di tecnica è quella di fornire dati sbagliati, tecnica anche questa incontrollabile: infatti, poichè i dati sono memorizzati nei nodi della rete e passano attraverso i nodi stessi durante la richiesta e lo scambio di informazioni, è possibile che alcuni utenti forniscano volontariamente dei dati errati. Questo fa sì che le informazioni che transitano nella rete non siano corrette, di conseguenza la rete sia inaffidabile. Una rete in cui le risorse condivise sono false diventa ovviamente una rete sempre meno utilizzabile. Gli utenti che inoltrano queste informazioni si trovano quindi a far parte passivamente del cosiddetto Distributed Denial of Service Attack (DDoS), ovvero di un metodo di attacco eseguito da un insieme di computer mirato ad impedire che la rete fornisca un servizio.

- Infine è stato notato che molti nodi, sfruttano l'imperfetta gestione delle query dei client, non per creare un attacco di tipo DDoS, ma a scopi commerciali, inviando cioè come risposte alle varie query una pagina web la quale una volta aperta redireziona automaticamente l'utente verso siti commerciali.

Purtroppo non esistono contromisure per questo tipo di attacchi, nè nei protocolli nè nei client. Inoltre, è praticamente impossibile individuare i nodi che lanciano un particolare attacco, a meno di non chiudere tutta la rete, eventualità che andrebbe proprio a favore di coloro che eseguono questi tentativi di distruzione della rete stessa.

3.4.2 La sicurezza

Un'altro problema importante delle reti P2P riguarda il concetto di sicurezza.

A differenza delle architetture client/server nelle reti P2P non è necessario eseguire alcun tipo di autenticazione che verifichi l'integrità e la veridicità dei dati, inoltre si ha una palese rivelazione dell'indirizzo IP degli utenti che usano i vari client. Oltre a questi problemi abbiamo il pericolo di furto di password e dati, violazione delle leggi di sicurezza, utilizzo di reti per scopi personali o attività illegali. Sono state fatte alcune proposte per la sicurezza per ciascuna categoria di reti P2P. Tra queste, nel caso di reti distribuite con condivisione di file, gli utenti possono recuperare dei dati che non violano le leggi di copyright, oppure sono disponibili in rete solo per membri registrati. Pertanto, questo tipo di reti racchiude un meccanismo che garantisce i diritti digitali. Uno dei metodi proposti a tale scopo è l'utilizzo del watermarking, la cui presenza andrebbe però a scapito della banda passante dell'utente.

Anche la diffusione di virus e worms è uno degli elementi da sottolineare all'interno del P2P.
Ad esempio nel 2001 si diffuse il worm Gnuman il cui obiettivo erano gli utenti di Gnutella; il worm era un file eseguibile con un nome che matchava le parole utilizzate nelle ricerche all'interno della rete, facendo credere agli utenti che il file segnalato contenesse i dati desiderati. In realtà, una volta eseguito, il worm tentava di diffondersi verso altri utenti Gnutella.

Infine, come per la maggior parte dei prodotti freeware, i client delle reti P2P, sopratutto quelli impiegati per lo scambio di dati, hanno il grave difetto di poter favorire la diffusione di spyware. Per spyware s'intende una qualunque tecnica che raccolga informazioni su persone o organizzazioni senza che queste ne siano a conoscenza. Uno spyware generalmente può entrare in un computer attraverso un virus o in seguito all'installazione di un programma.

Anche se chi inserisce uno spyware dentro il proprio programma garantisce che la privacy dell'utente resta riservata, all'interno del computer viene installato un piccolo server che manda informazioni sulle abitudini dell'utente, ad esempio su quali sono i nostri siti preferiti, sulla durata di visita di un sito, sui tipi di banner che l'utente clicca e molte altre cose. Il problema è che l'utente può controllare ciò che realmente viene inviato solo eseguendo e modificando il codice e poichè la tecnologia permette di trasmettere molto più che un banner pubblicitario, come ad esempio fare lo scan di un hard disk, leggere i cookies, cambiare la pagina principale del nostro browser, molti non gradiscono la presenza di questo software.
Ed è proprio sulla sicurezza da malware, worm e spyware che si basa il progetto descritto nei prossimi capitoli.

Capitolo 4

Gnutella

4.1 La storia

La prima implementazione del protocollo è stata il programma Gnutella scritto da Justin Frankel e Tom Pepper per la Nullsoft all'inizio del 2000. Il programma fu distribuito un giorno solo, il 14 marzo 2000 e fu scaricato da migliaia di persone, grazie all'annuncio apparso su Slashdot. Il codice sarebbe stato rilasciato in seguito probabilmente sotto licenza GNU GPL. Il giorno dopo AOL, che aveva da poco acquisito la Nullsoft, bloccò la distribuzione del programma per motivi legali e diffidò la Nullsoft dal continuarne lo sviluppo. Nonostante questo la rete gnutella sopravvisse sostenuta dalle migliaia di copie scaricate il primo giorno che continuavano a distribuire il programma. In pochi giorni poi il protocollo fu reingegnerizzato e nacquero nuovi programmi liberi in grado di accedere a questa rete.

Nonostante il nome contenga la parola GNU, essa non è parte del progetto GNU, per questo motivo la FSF, custode del progetto GNU, ne ha chiesto il cambiamento del nome. La seconda parte del nome deriva dalla Nutella, molto apprezzata dagli sviluppatori di questa rete.

Gnutella è un tipico esempio di overlay network.

La prima versione di Gnutella creava dei colli di bottiglia dato che trattava tutti gli utenti in modo egalitario. In seguito, grazie a Vincent Falco, un programmatore di West Palm Beach creatore di BearShare, uno dei più popolari client Gnutella, gli utenti furono raggruppati in base alla loro velocità di connessione, evitando così il rallentamento dell'intera rete da parte degli utenti con velocità inferiore. Attualmente, esistono molte applicazioni che utilizzano il protocollo di Gnutella e che permettono quindi la creazione di una rete estesa e robusta.

E' da sottolineare anche la grande versatilità dei client Gnutella, questi infatti sono disponibili per i sistemi operativi principali quali Windows (cfr. BearShare, Gnucleus, LimeWire, Morpheus, Phex, Shareaza, Xolox), Linux/Unix (cfr. Gtk-Gnutella, LimeWire, Mutella, Phex, Qtella) e Macintosh (cfr. LimeWire, Phex).

Gnutella permette quindi a tutti i suoi utenti di comunicare e lo fa in modo che chiunque possa far parte della rete risultante, a prescindere dal sistema operativo.

4.2 Il protocollo

Gnutella è una rete Peer to Peer di condivisione di file aperta. L'approccio è di tipo Peer to Peer puro, ovvero non esiste il ruolo di directory (invece presente in programmi quali Napster o eMule); ogni peer (o nodo equivalente) "conosce" alcuni vicini, facendo sì che le richieste vengano propagate all'interno della comunità sfruttando la relazione di vicinanza. Per ovviare al problema che le richieste continuino a girare in circolo, viene normalmente indicato il numero massimo di "salti" (hop) di una richiesta sui peer contigui a partire dal nodo che l'ha generata.

Gnutella è una rete completamente serverless, distribuita (Figura:4.1). I nodi sono trattati allo stesso modo, indipendemente dalla banda e dal numero di file condivisi. Ogni nodo si occupa sia di fornire i file che di inviare e rispondere alle richieste di routing degli altri nodi, compito riservato ai server in una rete centralizzata. Ogni nodo, quindi, è sia un client che un server: è definito a proposito servent.

Ciò consente una forte stabilità della rete, nella quale possono entrare e uscire contnuamente nodi senza modificare le prestazioni.

Per la condivisione dei file nella rete Gnutella, un utente, ad esempio il nodo A, dovrà utilizzare uno dei servent Gnutella che può lavorare sia come *SERVer* che come *cliENT*. Il nodo A si connetterà poi ad un altro nodo B, già connesso alla rete Gnutella, il quale comunicherà la presenza del nuovo nodo A a tutti i suoi vicini.

Questo si ripete ricorsivamente per tutti i nodi e i loro vicini, in modo che la rete venga via via a conoscenza della presenza del nuovo nodo A.

Una volta annunciata l'esistenza di tale nodo al resto della rete, l'utente potrà quindi lanciare delle query sui dati condivisi attraverso la rete.

L'annuncio della presenza di un nuovo nodo ha comunque vita limitata poichè la diffusione del messaggio di arrivo ha un contatore detto Time To Live (TTL): ad ogni passo o hop, il contatore del TTL viene decrementato di uno fino a quando non raggiunge il valore nullo. A questo punto la diffusione del messaggio viene bloccata ed il messaggio stesso viene eliminato. Per evitare che un utente aumenti troppo il TTL dei suoi messaggi, che possono essere messaggi sia di presenza ma anche di richiesta di risorse, i client Gnutella, generalmente, scartano i pacchetti con un TTL troppo elevato.

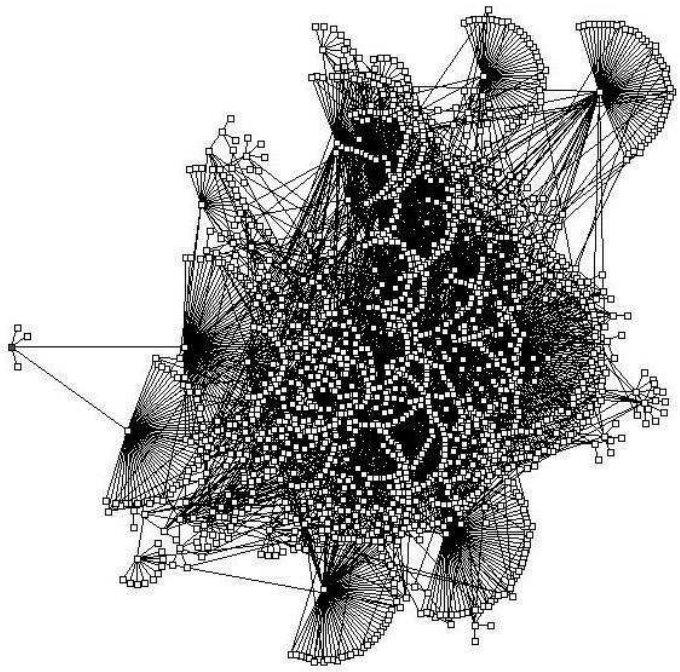


Figura 4.1: Schema di una rete serverless

Tipicamente il TTL è fissato a sette passi.

4.2.1 Dalla versione 4.0 alla versione 6.0

A causa della struttura totalmente decentralizzata, Gnutella richiede degli algoritmi scalabili per effettuare operazioni quali il routing e la ricerca. Infatti il protocollo utilizza per queste operazioni un flooding della rete con i messaggi che vengono ricevuti e trasmessi all'interno della rete stessa, al fine di stabilire delle connessioni o di fare una specifica richiesta. In questi anni sono state introdotte alcune novità per quanto riguarda il protocollo Gnutellabasate sull'utilizzo dei cosidetti "Ultrapeers", con la conseguente modifica del protocollo. Inoltre, anche la fase iniziale di handshaking è stata modificata. Infatti, un servent A, che supporta il nuovo protocollo, non potrà sapere se un servent B con cui è connesso sarà in grado di capire i suoi messaggi, quindi quando un nodo manda la richiesta di connessione manderà anche alcuni header HTTP con cui specifica il suo User Agent più altre proprietà. A questo punto il servent gli risponde consentendogli o meno la connessione e, in caso affermativo, manderà a sua volta alcuni header HTTP.

Solo dopo questa fase si passa al vero scambio di messaggi.

Gli Ultrapeers sono stati introdotti per ovviare al problema della scalabilità di Gnutella che, come abbiamo visto, è fortemente ridotta per l'utilizzo del flooding per diffondere le richieste di ciascun utente. Nella rete Gnutella tutti i nodi erano trattati allo stesso modo prescindendo dalla loro banda passante oppure dal loro potere computazionale. La possibilità di avere una rete stabile senza dipendenze particolari da un host è un elemento fondamentale che ha portato Gnutella ad una grandissima popolarità. Ma la sovrabbondanza di tutti i messaggi che transitano attraverso la rete Gnutella, inclusi i messaggi di Ping e le varie query ed l'impossibilità di assicurare la disponibilità di un dato che non sia sufficientemente vicino all'host che lo richiede, hanno intaccato la scalabilità del protocollo. Tutto questo ha portato, e porta in parte anche ora, a dei gravi problemi di gestione della banda di connessione.

L'idea di base è stata quella di dividere in due grandi categorie i nodi che fanno parte della rete: le foglie e gli Ultrapeers. Questa suddivisione avviene all'interno della rete dinamicamente ed automaticamente. Le foglie mantengono solo le connessioni con gli Ultrapeers mentre quest'ultimi mantengono da dieci a cento connessioni con le foglie e fino ad un massimo di dieci connessioni con gli altri Ultrapeers. Lo scopo degli Ultrapeers è quello di proteggere tutti i nodi foglia da tutti i ping e da tutte le richieste che viaggiano sulla rete Gnutella. Infatti, saranno gli Ultrapeers ad inoltrare una query verso un nodo foglia solo se questo può soddisfare una determinata richiesta.

Un nodo può eleggersi automaticamente Ultrapeer se soddisfa una serie di requisiti (sistema operativo, bandwidth, firewalled, etc..) Viceversa, un superpeer può suggerire ad un altro superpeer di diventare un nodo foglia qualora quest'ultimo non abbia sufficienti collegamenti alle foglie.

4.2.2 I vantaggi e gli svantaggi

La rete gnutella è una rete P2P totalmente decentralizzata ed in quanto tale gode degli stessi vantaggi e svantaggi di tutte le reti P2P decentralizzate.

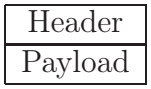
Gli aspetti positivi sono i seguenti:

- La rete non sfrutta nessun server centrale, pertanto non si possono creare colli di bottiglia e non si ha dipendenza dalla presenza di un solo server per l'accesso alla rete.
- Non è penalmente perseguibile, sia per l'assenza di un server centrale sia perchè i file Gnutella hanno l'aspetto di un qualunque altro file.
- E' compatibile con qualunque tipo di file, e quindi non è limitato come Napster al solo scambio di file mp3.
- Consente agli utenti dietro un firewall di connettersi ugualmente alla rete, tramite messaggi HTTP.
- E' totalmente freeware

D'altro canto la rete presenta anche aspetti negativi:

- Per connettersi è necessario di almeno un altro utente vicino e connesso.
- Ogni utente ha un "raggio d'azione" limitato, causato dal TTL che fa sì che i messaggi non raggiungano mai quei nodi che si trovano oltre il numero di "hop" prefissato.
- Non è scalabile. Ad esempio, l'arrivo di una richiesta con un TTL pari a 10 porta allo scambio fino ad un milione di messaggi nel caso che ogni vicino sia connesso ad altri 6 nodi.
- Si ha una grossa difficoltà ad autenticare la fonte da cui provengono i dati ottenuti.

Tutte le comunicazioni Gnutella utilizzano il protocollo TCP/IP.
Una volta stabilita la connessione TCP/IP tra due servent, inizia lo scambio di messaggi tra di loro. Il primo messaggio è di handshake ed in seguito, una volta connessi, vengono scambiati tutti gli altri tipi di messaggi. La struttura dei messaggi scambiati è sempre la stessa per ogni tipo.



In particolare l'header è composto da 23bytes divisi secondo il seguente schema:

Bytes	Descrizione
0-15	ID/GUID (Globally Unique ID) Messaggio
16	Tipo messaggio
17	TTL
18	Numero di Hop
19-22	Lunghezza del Payload

ID Messaggio: E' una stringa di 16 byte che identifica unicamente il messaggio nella rete. I servent dovrebbero porre ad 1 tutti i bit nell'ottavo byte a 0 i bit nel quindicesimo byte e tutti gli altri bit devono essere generati casualmente.
Tipo Messaggio: Indica il tipo del messaggio (Ping, Pong, Bye, Push, Query, Query Hit).
TTL: Time-to-Live del messaggio
Numero di Hop: Indica il numero di salti che ha gia compiuto il messaggio.
Lunghezza del Payload: Indica la lunghezza del messaggio immediatamente successivo all'header. La lunghezza deve essere inferiore a 4Kb.

4.2.3 Handshake

Definisce una sequenza di azioni che un servent deve eseguire dopo aver stabilito una connessione TCP/IP con un altro servent.
La fase di handshake prevede l'esecuzione di 8 passi:

1. Il nodo A stabilisce una connessione TCP/IP con il nodo B
2. A invia a B una stringa "GNUTELLA CONNECT/0.6"
3. A invia un header con tutte le proprie specifiche a B
4. B risponde con una stringa "GNUTELLA/0.6 200 OK"
5. B invia un header con tutte le proprie specifiche ad A come in (3)
6. A invia una stringa "GNUTELLA/0.6 200 OK" come in (4) dopo aver parsato l'header inviatogli da B, altrimenti chiude la connessione
7. A invia un header con delle specifiche che possono essere utili a B come in (3)
8. Sia A che B si scambiano messaggi utilizzando le specifiche ottenute ai punti (3) e (5)

Un esempio di handshake può essere il seguente:

Nodo A	Nodo B
GNUTELLA CONNECT/0.6 User-Agent: BearShare/1.0 Pong-Caching: 0.1 GGEP: 0.5	
	GNUTELLA/0.6 200 OK User-Agent: BearShare/1.0 Pong-Caching: 0.1 GGEP: 0.5 Private-Data: 5ef89a
GNUTELLA/0.6 200 OK Private-Data: a04fce	

4.2.4 Query e Query Hit

Questi due tipi di messaggi vengono utilizzati per richiedere una risorsa desiderata (Query) ed inviare una risposta nel caso che la risorsa cercata matchi con l'insieme dei dati condivisi in rete (Query Hit). Nello specifico, un messaggio di Query viene inoltrato a tutti i nodi vicini per richiedere la risorsa. Il messaggio definisce la velocità minima richiesta e la stringa di ricerca. Un messaggio di Query Hit viene inviato in risposta ad una query se il servent che la riceve trova un match fra l'insieme dei suoi dati e la stringa contenuta nel messaggio di Query Il payload del messaggio descrive il numero e la lista degli elementi con cui si ritiene ci sia un match, la porta del servent e la sua velocità. Questo messaggio viene inviato unicamente sulla connessione da cui proviene la query.

4.2.5 Push

Questo tipo di messaggio viene inviato se il servent riceve un messaggio di tipo Query Hit da un servent che non supporta connessioni in entrata, ad esempio perchè si trova dietro ad un firewall. Il messaggio indicherà l'identificativo del file l'indirizzo IP e la porta con cui richiedere il file.

4.2.6 Ping e Pong

Il Ping è il primo messaggio inviato alla rete per annunciare agli host vicini la propria presenza. Un servent che riceve un Ping dovrebbere rispendere con un Pong; il Ping ha payload vuoto e viene reinoltrato, al momento della ricezione, su tutte le connessioni attive presenti. Il Pong invece è il messaggio inviato in risposta al Ping, esso include l'indirizzo IP del servent Gnutella connesso e le informazioni relative alla porta su cui la connessione è attiva, sul numero di file condivisi e sulla quantità di KB di dati messi a disposizione per la rete dal nodo. Il Pong viene inoltrato solo sulla connessione da cui proveniva il Ping.

Capitolo 5

Attestazione di reti p2p

5.1 Il problema

Come già detto, una rete P2P non può convalidare i suoi peers e quindi isolare quelli maliziosi, tutto è lasciato al buonsenso dei peers stessi. Un attaccante potrebbe sfruttare la rete per diversi tipi di attacco o potrebbe riempire la rete di contenuti falsi (Figura:5.1).

L'attestazione offre molti vantaggi in questo contesto, perchè permette di mantenere la rete P2P sicura e lontana da peer maliziosi. L'idea di base è creare una rete con soli peers sicuri e autenticati. Se un nuovo peer vuole entrare nella rete, dovrà prima attestarsi e, solo dopo, potrà essere accettato nella rete (Figura:5.2).

Inoltre, qualora un peer della rete diventi malintenzionato, la rete provvede ad eliminarlo e ad avvertire tutti gli utenti connessi che quel peer non è più sicuro.

Questo progetto è stato sviluppato sul client gnutella, installato su Debian Etch paravirtualizzato su una Macchina Virtuale (MV) creata da Xen 3.1.0.

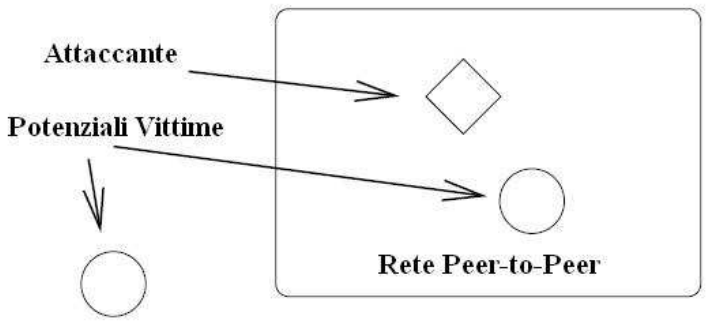


Figura 5.1: Schema di un attacco

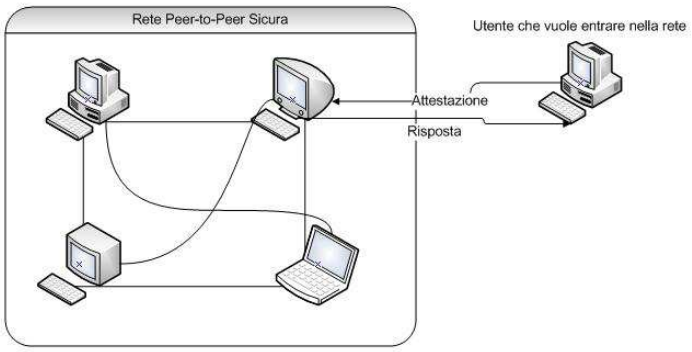


Figura 5.2: Rete P2P sicura

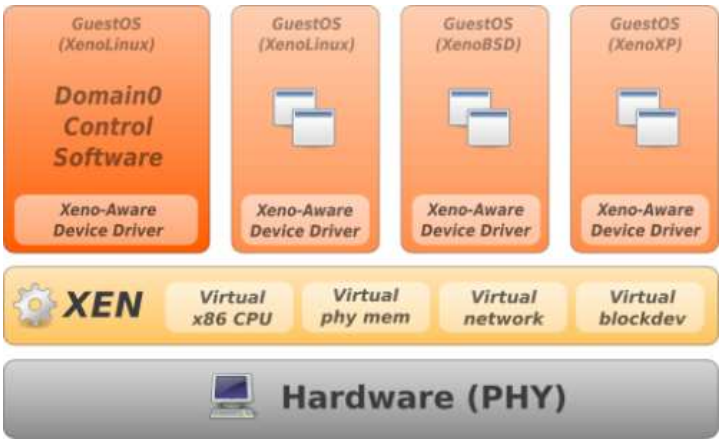


Figura 5.3: Esempio di virtualizzazione

5.2 Virtualizzazione

La virtualizzazione consiste nell'implementazione virtuale di una risorsa fisica ed è una strategia applicabile sia a risorse software che hardware. Una possibile definizione di virtualizzazione può essere: *"Una metodologia per condividere risorse, anche hardware, di un computer tra più istanze di esecuzione, applicando concetti o tecnologie come partizionamento hardware o software, time sharing, emulazione, quality of service. Questa istanza viene chiamata MV."*

La virtualizzazione consente di replicare in modo logico le risorse hardware e permette di ottimizzare lo sfruttamento delle prestazioni offerte dalle più recenti componenti hardware.(Figura:5.3)

Infatti, negli ultimi anni c'è stata una notevole evoluzione nel campo della virtualizzazione, riconducibile all'evoluzione tecnologica in campo hardware. Memoria e CPU vengono infatti virtualizzati e condivisi tra le applicazioni, per far sì che le risorse a disposizione vengano sfruttate al meglio, dando inoltre l'illusione ad ogni MV di essere l'unica in esecuzione. Anche dispositivi di input, video e schede di rete, ad esempio, vengono virtualizzate e gestite in modo che sia possibile un accesso concorrente a queste risorse delle macchine virtuali. I numerosi benefici, apportati dalla virtualizzazione, hanno spinto i produttori di componenti hardware a facilitare il processo di virtualizzazione delle risorse, tramite supporti hardware e firmware. I due esempi maggiori sono l'Intel Virtualization Tecnology (INTEL-VT) e l'AMD Pacifica Virtualization Tecnology (AMD-VT).

5.2.1 Virtual Machine Monitor

Il software di virtualizzazione, detto anche Virtual Machine Monitor (VMM), è in grado di tenere traccia dello stato di ogni sistema operativo in esecuzione nelle MV, in maniera simile a quella con cui un sistema operativo tradizionale tiene traccia dello stato dei processi e, inoltre, fornisce protezione e isolamento tra le macchine virtuali.

Il VMM offre molte vantaggi:

- **Consolidamento:** si possono installare più sistemi operativi su una sola macchina fisica per poter sfruttare al massimo le potenzialità hardware di quest'ultima.
- **Migrazione:** si possono spostare una o più MV da una macchina fisica ad un'altra per poter bilanciare il carico tra di loro.
- **Salvataggio e Ripristino di stati:** si può salvare lo stato di una o più macchine virtuali o ripristinare un vecchio stato salvato per tornare al punto precedente della sua storia, questa operazione si chiama *rollback*.
- **Isolamento:** è possibile garantire un alto livello di protezione tra le varie MV, quindi, non è possibile che differenti MV interferiscano fra di loro o violino i meccanismi di protezione.
- **Affidabilità:** i crash delle applicazioni su una MV non possono interferire con le altre macchine virtuali.
- **Sospensione:** è possibile fermare l'esecuzione di una MV per poi farla ripartire in un secondo momento ripartendo dal punto in cui si era fermata.

Esistono due tipi principali di VMM:

- **Unhosted VMM:** il software di virtualizzazione si interpone tra il livello hardware/firmware della macchina ed il sistema operativo. Xen è un esempio di questa tecnologia. (Figura:5.4 (a))
- **Hosted VMM:** il software di virtualizzazione è a sua volta un processo all'interno del sistema operativo. Il VMM si interpone tra il sistema operativo e le applicazioni che esso esegue. VirtualBox è un esempio di questa tecnologia. (Figura:5.4 (b))

Un componente software, eseguito su una MV, si comporta come se fosse eseguito direttamente sull'architettura fisica e ne avesse pieno controllo. La performance di un sistema guest installato su un Unhosted VMM è molto vicina a quella originale, cioè a quella che si può ottenere senza utilizzare il software di virtualizzazione. Infatti, l'interfaccia virtuale è identica, o molto simile, a quella reale e quindi l'overhead dovuto al trasferimento delle richieste tra le due interfacce è ridotto drasticamente. Quindi, il VMM interviene solo quando deve mantenere il controllo di una MV, ad esempio quando deve garantire l'isolamento tramite la gestione degli spazi di indirizzamento del guest e dei processi in esecuzione. In tutti gli altri casi, la MV utilizza direttamente le risorse poichè le istruzioni non privilegiate possono essere eseguite direttamente sulla macchina reale. Infine, la dimensione ridotta, come numero di istruzioni, di una VMM rispetto ad un sistema operativo, rende il VMM stesso più efficiente nell'esecuzione di operazioni e permette di verificarne formalmente la correttezza.

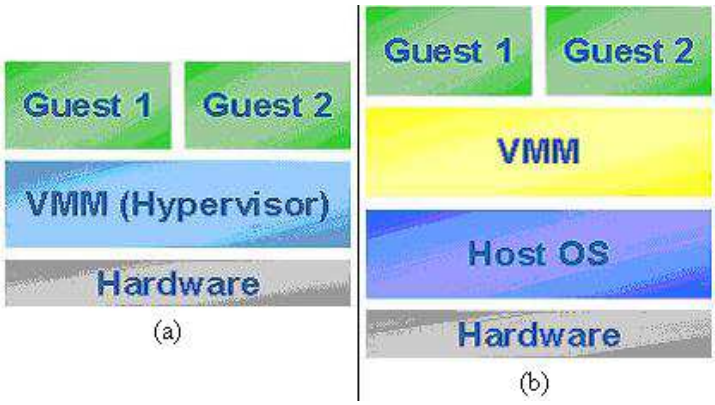


Figura 5.4: Type-I Unhosted VMM (a); Type-II Hosted VMM (b)

5.2.2 Livelli di privilegio

I processori supportano politiche di protezione che utilizzano il concetto di ring (Figura:5.5).

Il software eseguito a ring-0 gode dei privilegi più elevati, mentre quello eseguito a ring-3 possiede i privilegi più bassi. I privilegi permettono, ad esempio, di eseguire istruzioni privilegiate senza causare eccezioni. Solitamente, i sistemi operativi x86-Compatibili sono eseguiti a ring-0, mentre le applicazioni vengono eseguite a ring-3. Ovviamente, un VMM non può permettere che un GuestOS venga eseguito a ring-0. Viene quindi utilizzata la tecnica del *Ring Deprivileging*, che consiste nell'eseguire un GuestOS ad un ring con privilegi più bassi.
In particolare esistono due modelli principali:

- **0/1/3:** i GuestOS vengono eseguiti a ring-1, mentre le applicazioni a ring-3.
- **0/3/3:** i GuestOS e le applicazioni vengono eseguite a ring-3.

5.2.3 Virtualizzazione Completa e Paravirtualizzazione

Esiste anche una distinzione tra le varie modalità di virtualizzazione:

- **Virtualizzazione Completa:** il GuestOS può essere eseguito su una MV senza nessuna modifica al codice.
- **Paravirtualizzazione:** il codice del GuestOS deve essere modificato per poter essere eseguito su una MV

Nel primo caso, il VMM esporta un'interfaccia virtuale del tutto identica all'hardware reale, quindi deve adottare delle tecniche per catturare ed eseguire operazioni privilegiate richieste dal GuestOS.

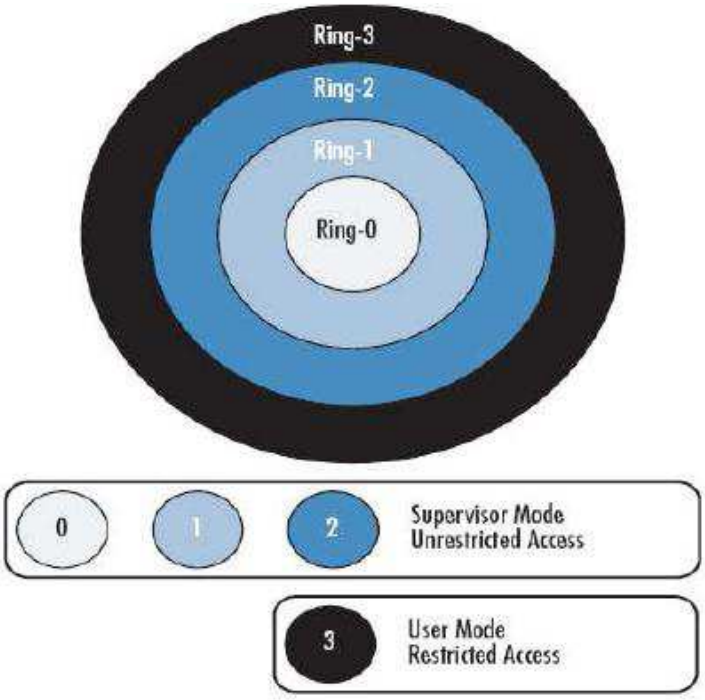


Figura 5.5: Livelli di privilegio

Nel secondo caso, invece, viene esportata un'interfaccia simile a quella sottostante. Per questo sono necessarie delle modifiche al codice del sistema ospite, ma non alle applicazioni eseguite sul sistema operativo.

Uno dei principali vantaggi della paravirtualizzazione sono migliori prestazioni rispetto alla virtualizzazione completa poichè, non è necessario effettuare la traduzione dinamica delle istruzioni. Per contro però, il numero di sistemi operativi, che hanno la possibilità di utilizzare tale metodo è minore, poichè possono essere eseguiti solo i sistemi operativi di cui è possibile modificare il codice.

5.3 Xen

5.3.1 La Storia

Xen è un Type-I VMM Open Source nato nel 2003 come progetto di ricerca all'università di Cambridge. Attualmente, è giunto alla versione 3.3.1 ed è sviluppato principalmente dalla XenSource Foundation. Nato per supportare la paravirtualizzazione, dalla versione 3.0 Xen inizia a supportare anche la virtualizzazione completa, grazie alla quale è possibile eseguire anche Windows come sistema Guest, ottenendo ottimi livelli di performance. Xen è uno tra i più veloci software di virtualizzazione, grazie anche alla sua leggerezza in termini di linee di codice. Infatti non supera le 150000 righe. Questo porta ad un basso overhead ed a prestazioni dei sistemi Guest simili a quelli nativi.

5.3.2 L'architettura

Xen è basato sul modello 0/1/3, quindi i sistemi operativi vengono eseguiti a ring-1 mentre le applicazioni a ring-3. Inoltre, supporta sia architetture a 32 che a 64 bit.

Il VMM, detto anche *hypervisor*, implementa delle API che permettono di controllare i GuestOS che possono interagire direttamente o indirettamente con il livello hardware/firmware sottostante.
In Xen, le macchine virtuali prendono il nome di domini e possono essere di due tipi:

- **Dominio-0 o Dom0:** rappresenta una macchina privilegiata che gestisce e controlla tutti gli altri domini attraverso il *Management API* e il *Management Code*.
- **Dominio-U o DomU:** sono le macchine virtuali in cui vengono eseguiti i sistemi operativi ospiti.

L'architettura generale di Xen è mostrata in fig. 5.6.
Durante il boot, Xen, deve inizialmente caricare il kernel del Dom0.

Il Dom0 è una delle più importanti componenti perchè fornisce i driver delle periferiche e le interfacce utente per i dispositivi presenti. Inoltre, gestisce la schedulazione delle CPU virtuali, assegnate ai DomU, l'allocazione della memoria e l'accesso ai device fisici, come dischi e schede di rete.

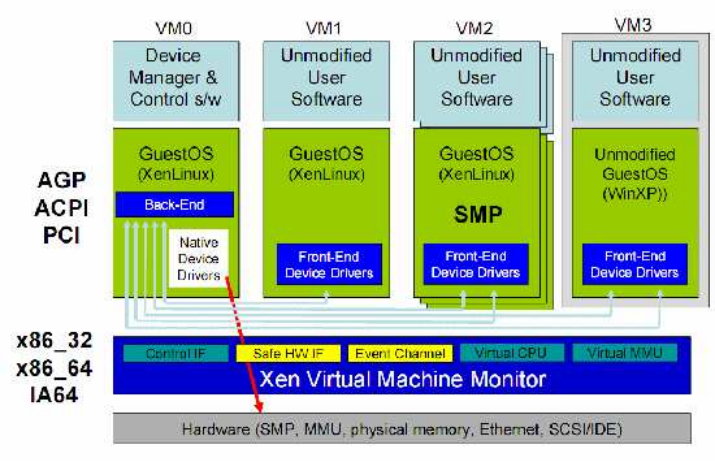


Figura 5.6: Architettura di Xen

Infine, Xen offre un'interfaccia d'uso all'hypervisor, tramite due processi demone:

- **xend:** fornisce un'interfaccia amministrativa all'hypervisor, permettendo agli utenti di definire precise policy.
- **xenstored:** fornisce un'interfaccia per lo *XenStore* database.

Invece il DomU è un sistema molto più limitato del Dominio-0, perchè non può invocare hypercall o accedere direttamente all'hardware della macchina reale. Dato che le periferiche sono virtuali e generiche, il DomU non deve far altro che implementare un driver generico per ogni tipo di dispositivo. Questo è un vantaggio molto importante, perchè anche sistemi per cui sono stati implementati pochi driver possono essere eseguiti come GuestOS sfruttando i driver del Dom0.

Xen supporta solo un Dom0 e molti DomU sulla stessa macchina fisica. Inoltre, è possibile migrare uno o più DomU da una macchina reale ad un'altra sospendendo l'esecuzione della MV, salvando lo stato e ripristinandolo su una macchina fisica diversa.

5.3.3 Virtualizzazione delle Schede di Rete

Una volta caricato il Dom0, Xen crea sette coppie di connessioni virtuali della scheda ethernet utilizzabili dal Dominio-0. Ogni coppia deve essere pensata come due interfacce ethernet connesse tra loro da un cavo ethernet interno (Figura:5.7).

Questa coppia è composta da due schede virtuali, o logiche, chiamate ***vethX*** e ***vifY.X***. Ogni scheda di rete fisica eth0, eth1, ... è virtualizzata da una scheda di rete virtuale veth0, veth1, ...

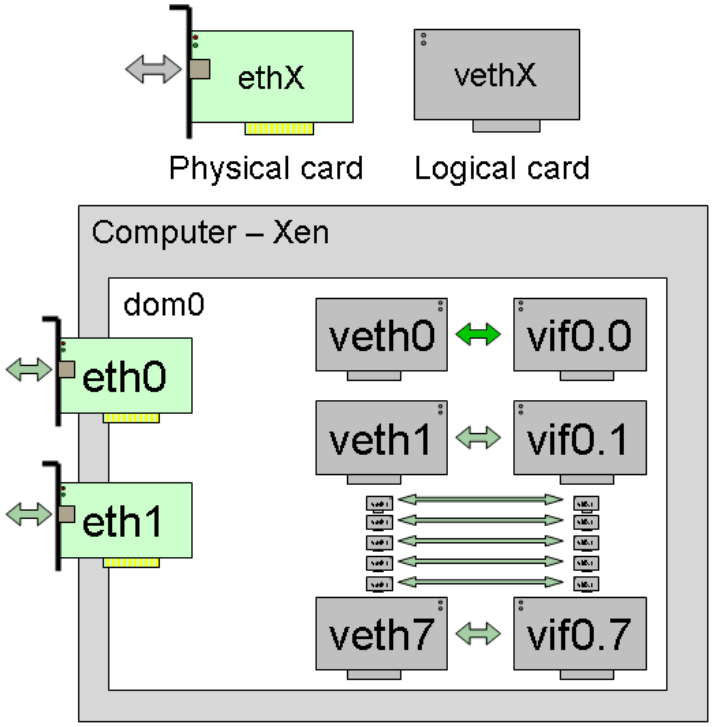


Figura 5.7: Virtualizzazione delle schede di rete

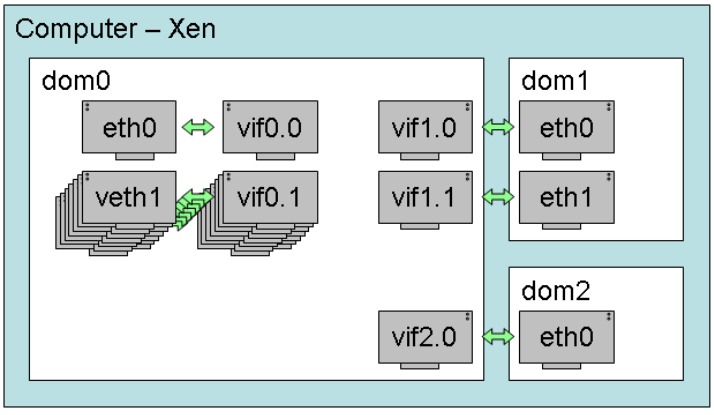


Figura 5.8: Connessioni logiche tra i DomU

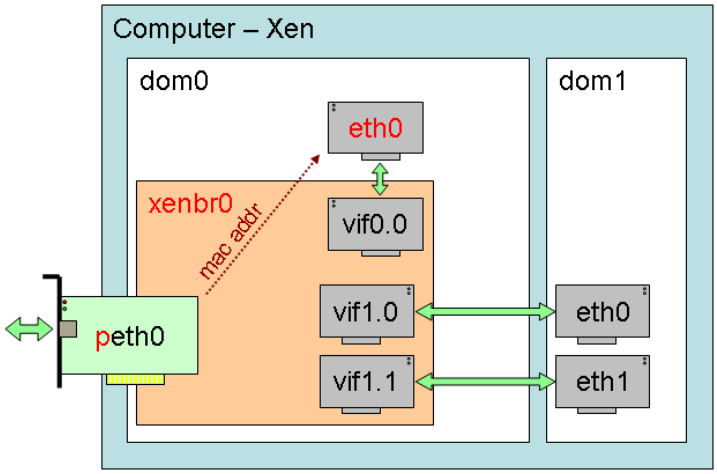


Figura 5.9: Connessioni logiche con brigde

Ogni scheda logica è associata ad una scheda virtuale *vifY.X* dove la **X** corrisponde al numero della scheda logica *veth* mentre la **Y** corrisponde all'id del DomU (Figura:5.8).

Ogni volta che viene creata una nuova istanza di un DomU, viene assegnato al dominio un nuovo id crescente. Il primo DomU creato avrà id numero 1, il secondo avrà id numero 2 anche se il primo DomU non è più in esecuzione. Per ogni nuovo DomU, Xen crea una coppia di connessioni di rete virtuali tra il nuovo DomU e il Dom0 (Figura:5.8). Infine, quando un DomU viene eliminato, anche le connessioni virtuali vengono distrutte.

Per default, Xen, utilizza un *bridge* con il Dominio-0 per permettere a tutti i DomU di risultare connessi alla rete come un singolo host (Figura:5.9).

Xen permette anche di utilizzare l'*iptables*, situata nel Dom0, per gestire l'instradamento dei pacchetti provenienti dai sistemi guest.

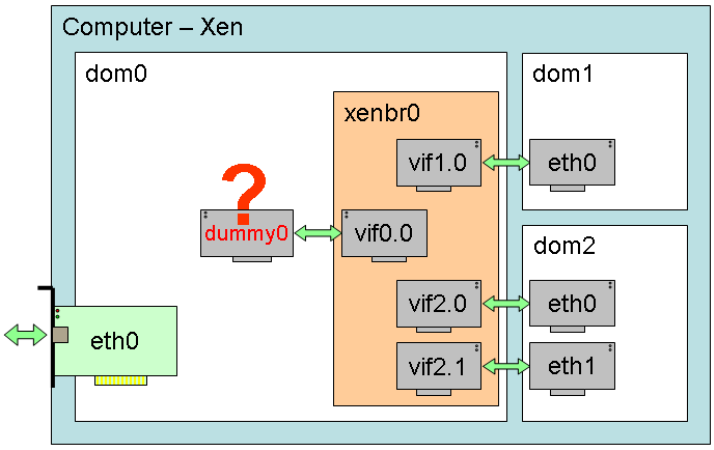


Figura 5.10: Connessioni logiche con dummy

Questo è possibile perchè:

- Arriva alla scheda di rete reale **eth0**
- Viene gestito dal driver Ethernet del Dom0
- Appare come se fosse stato ricevuto da **peth0**
- peth0 trasmette il pacchetto ricevuto al bridge, in questo caso **xenbr0**
- Il *bridge* distrubuisce il pacchetto, come uno *switch*, nell'interfaccia *vif* giusta grazie l'indirizzo MAC
- Il pacchetto può essere filtrato tramite l'iptables
- Se non è stato filtrato, il pacchetto giunge a destinazione nell'interfaccia *vif* corretta

Se sono necessari più indirizzi pubblici in uscita da una macchina reale, è possibile utilizzare i moduli *dummy*. In questo modo, si crea un *bridge* "trasparente" tra *dummy0* e *xenbr0* che è, di default, "collegato" ad *eth0*. In questo modo, è quindi possibile creare una sorta di LAN fra i DomU e Dom0 utilizzando Dom0 come gateway per la rete esterna, avendo anche il vantaggio di non modificare alcuna configurazione di rete per i DomU qualora venga cambiato indirizzo di rete per eth0. Il risultato è mostrato nelle figure 5.10 e 5.11.

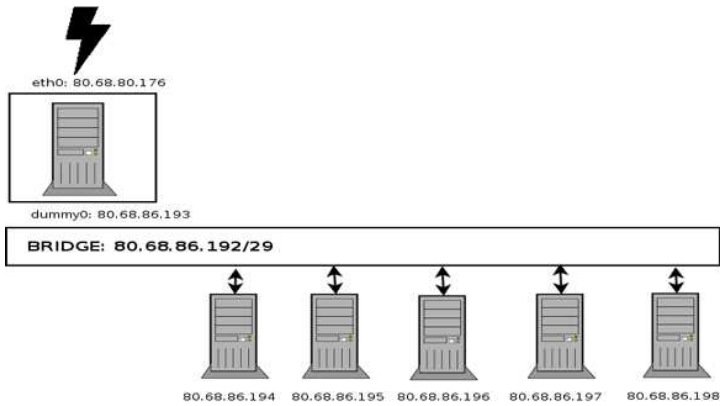


Figura 5.11: Esempio di connessione con dummy

5.4 Introspezione

Una tecnica generica e indipendente dal livello architetturale per rilevare intrusioni ad un sistema informatico è quella dell'introspezione. Questa tecnica permette di analizzare lo stato di un sistema per ricostruire le strutture dati critiche utilizzate dal sistema e controllarne la loro consistenza. Nel nostro caso, l'introspezione permette di controllare lo stato del kernel delle MV in esecuzione su una macchina fisica. In una macchina fisica su cui sono installate più MV, la libreria d'introspezione viene installata nel Dominio-0 che avrà il compito di controllare tutte le altre. Questa tecnica è difficile da evadere, o da attaccare direttamente in quanto i controlli di consistenza vengono effettuati ad un livello più basso rispetto a quello che un attaccante può ottenere, cioè il livello kernel.

Il progetto utilizza una libreria d'introspezione, chiamata **XEN-VMI**, sviluppata, per Xen, dal Dipartimento d'Informatica dell'Università di Pisa, e disponibile all'indirizzo *http://www.di.unipi.it/~sgandurra/projects/projects.php*.

5.4.1 Implementazione della libreria Xen-VMI

La libreria Xen-VMI è stata realizzata utilizzando il linguaggio C. Inoltre per realizzare Xen-VMI, è stata utilizzata la libreria XenAccess, in particolare per ottenere gli indirizzi virtuali delle strutture dati critiche del kernel in esecuzione su una MV da monitorare e per mappare le relative pagine all'interno dello spazio di indirizzamento dell'introspettore, un modulo delegato ai controlli. Infine, è stata utilizzata la libreria OpenSSL per eseguire alcune funzioni, come il calcolo degli hash di regioni critiche di memoria del kernel. L'architettura di Xen-VMI è mostrata in figura 5.12

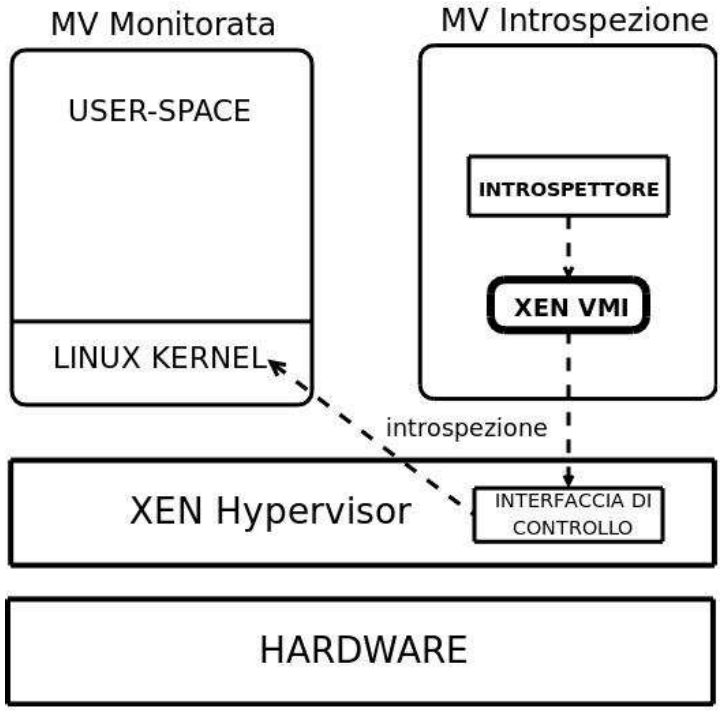


Figura 5.12: Architettura di Xen-VMI

I passi fondamentali per ricostruire le strutture dati ed effettuare i controlli di consistenza sono essenzialmente sei:

1. Si congela la MV che si vuole controllare
2. Si mappano le pagine di memoria che contengono le strutture dati critiche del kernel della MV che si vuole monitorare nello spazio di indirizzamento dell'introspettore. In questo passo si utilizza il file System.map, relativo al kernel in esecuzione sulla MV per ottenere l'indirizzo virtuale associato al simbolo relativo alla struttura dati ricercata.
3. Viene effettuato il "casting" della memoria mappata per ottenere le strutture dati corrispondenti a quelle utilizzate dal kernel.
4. Se all'interno della struttura dati è presente un puntatore, ad esempio se la struttura è una lista linkata, Xen-VMI mappa la pagina contenente l'indirizzo virtuale puntato ed effettua il casting alla struttura dati opportuna come al passo 3
5. L'introspettore effettua i controlli di consistenza sulla struttura dati ricostruita. Se ci sono altri puntatori, torna al passo 3
6. Se non ci sono violazioni all'integrità delle strutture dati controllate, l'introspettore riavvia l'esecuzione della MV

L'introspettore definisce diverse funzioni per controllare l'integrità di una MV, alcune di queste sono elencate di seguito.

Funzione per Rilevare Modifiche al Kernel

Questa funzione rileva attacchi sia al codice che a strutture critiche del kernel. L'introspettore controlla le pagine di memoria della MV da controllare e che contengono:

- Il codice del kernel, contenuto tra gli indirizzi riferiti dai simboli *_text* e *_etext*
- La tabella dei puntatori per le chiamate di sistema, contenuta nell'array riferito dal simbolo *sys_call_table*
- La tabella dei descrittori delle interruzioni, contenuta nella tabella riferita dal simbolo *idt_table*. In realtà Xen modifica proprio questa tabella con una tabella gestita da lui stesso, chiamata *trap_table*, quindi i controlli sono effettuati su quest'ultima tabella.

Controlli sui Loadable Kernel Module

Linux supporta il concetto di Loadable Kernel Module (LKM) per poter estendere dinamicamente un kernel in esecuzione, inserendo nuove funzionalità solo quando queste sono effettivamente richieste.

Sebbene questa modalità per estendere il kernel in esecuzione sia conveniente ed efficace, introduce serie minacce per la sicurezza del sistema. Ad esempio, dopo aver compromesso un sistema, un attaccante può utilizzare questo metodo per inserire un modulo nel kernel per nascondere le proprie attività, modificando opportune chiamate di sistema. Questa funzione di Xen-VMI ha il compito di ricostruire la lista riferita dal simbolo *modules* per ottenere la lista dei moduli inseriti nel kernel.

Successivamente, la funzione verifica l'integrità di tutti i moduli e, inoltre, se essi sono autorizzati. A questo scopo, prima che una MV possa essere utilizzata, vengono caricati manualmente tutti i moduli del kernel di cui Linux ha un effettivo bisogno e, per ogni modulo, questa funzione calcola l'hash delle pagine in memoria contenenti il codice e salva questo valore, associato al nome del modulo, su file.

Successivamente, quando la MV è in esecuzione, l'introspettore invoca questa funzione per verificare che tutti i moduli in esecuzione siano moduli autorizzati e il cui codice non sia stato modificato, confrontando i valori degli hash calcolati in precedenza con quelli calcolati durante la nuova invocazione. Una differenza nei valori degli hash o nel nome implica o che un modulo autorizzato è stato modificato o che un modulo non autorizzato è stato inserito nel kernel.

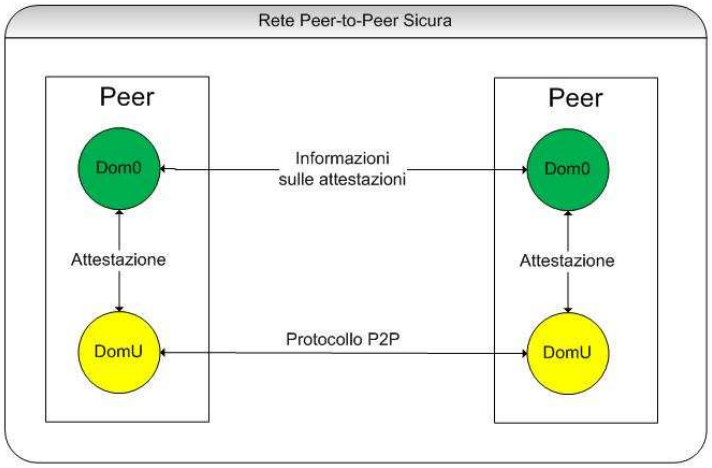


Figura 5.13: Architettura di rete P2P sicura

5.5 Attestazione su Gnutella

5.5.1 L'idea

L'idea base del progetto è quella di avere una rete P2P sempre sicura e i peer al suo interno sicuri ed attestati.

Per garantire la sicurezza e l'integrità di un nodo della rete, è necessario che una macchina riconosciuta come sicura attesti un peer e gli permetta di essere riconosciuto come un nodo sicuro. Per fare questo sono state usate, per ogni macchina fisica, due macchine virtuali. Come è già stato detto, Xen crea un solo **Dom0** e molti **DomU**, nel nostro caso ci siamo serviti del Dom0, che abbiamo assunto essere una macchina sicura, e di un DomU, sul quale è installata un'applicazione P2P, nel nostro caso Gnutella.

Poichè il Dom0 è assunto come macchina sicura, provvederà ad attestare, grazie alla libreria XEN-VMI, ogni volta che la rete lo reputerà necessario, il suo DomU. In questo modo, se il DomU è corrotto, il Dom0 provvederà ad avvertire gli altri utenti della rete che il proprio peer non è più sicuro. Lo schema è mostrato in fig. 5.13.
I controlli di attestazione vengono effettuati ad ogni interazione tra i peer, cioè durante le tre fasi elencate di seguito:

- **Handshake**
- **Download**
- **Ping**

5.5.2 Il protocollo

Il protocollo, rispetto alla normale applicazione, non ha subito molte modifiche.

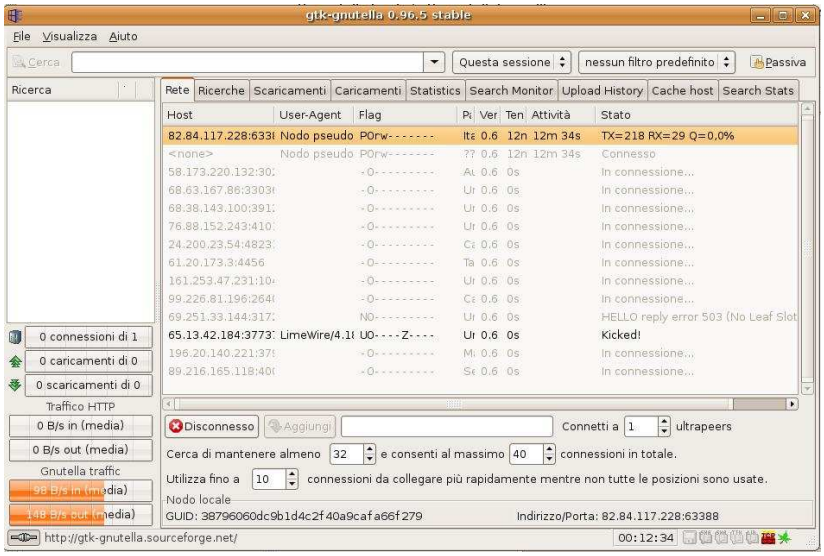


Figura 5.14: Espulsione di un nodo

Le modifiche più importanti sono:

- Un nuovo thread per lo scambio di informazioni tra Attestatore e applicazione
- L'aggiunta di nuovi campi d'informazione nei messaggi scambiati tra le due applicazioni
- Nuove opzioni nel file di configurazione dell'applicazione

Scendiamo nel dettaglio per spiegare meglio ogni modifica singolarmente.

Nuovo Thread

Appena avviato, Gnutella, apre un nuovo thread, chiamato *handler_attestator*, che gestisce lo scambio di messaggi tra attestatore e applicazione. In particolare, il thread riceve informazioni dall'attestatore se qualche nodo non è più sicuro in modo che l'applicazione possa isolarlo dalla rete. Il nodo viene espulso, in gergo *bannato*, e l'applicazione provvede a mostrare all'utente l'avvenuta espulsione con il termine ***Kicked!***, come mostrato in figura 5.14.

Nuovi campi

L'aggiunta di nuovi campi nei messaggi scambiati tra i vari peer è necessaria per garantire la sicurezza della rete. Ad esempio, sono stati aggiunti due campi, che contengono l'indirizzo IP e la porta della macchina attestatrice del nodo.

Questo argomento è descritto dettagliatamente in seguito, caso per caso.

Nuove Opzioni

Gnutella utilizza un file nel quale sono salvate tutte le configurazioni dell'applicazione, chiamato *config_ gnet*. In questo file sono stati inseriti alcuni campi che permettono all'applicazione di creare i nuovi messaggi. Tra questi campi c'è l'IP e la porta dell'Attestatore, o, ad esempio, il numero di tentativi di connessione che l'attestatore deve fare prima di dichiarare che un nodo non è sicuro.

5.5.3 Handshake Sicuro

La fase di handshake, rispetto a quella originale è stata modificata per implementare l'attestazione del nodo che vuole connettersi alla rete sicura. La sequenza di esecuzione modificata utilizza 16 passi rispetto agli 8 della versione non sicura.

1. Il nodo A stabilisce una connessione TCP/IP con il nodo B
2. A invia a B una stringa "GNUTELLA CONNECT/0.6"
3. A invia un header con tutte le proprie specifiche a B con i nuovi campi per l'attestazione
4. B manda un messaggio al proprio attestatore specificando che A vorrebbe connettersi
5. L'attestatore di B(**AttB**) richiede l'attestazione di A, attraverso il protocollo UDP, all'attestatore di A(**AttA**)
6. AttA usa la libreria Xen-VMI per attestare A ed invia il risultato, sempre tramite UDP, ad AttB
7. AttB se legge la risposta invia il risultato a B, altrimenti può ritentare il passo 5 a seconda del numero di volte scritto nel proprio file di configurazione
8. B riceve la risposta da AttB, se questa è positiva risponde con una stringa "GNUTELLA/0.6 200 OK" e procede, altrimenti chiude la connessione TCP/IP con A
9. Se B ha inviato la stringa al passo precedente allora invia un header con tutte le proprie specifiche ad A come in (3)
10. A procede come B al passo 3, manda al proprio attestatore, AttA, un messaggio specificando che B vorrebbe connettersi
11. AttA richiede l'attestazione di B, attraverso il protocollo UDP, ad AttB
12. AttB usa la libreria Xen-VMI per attestare B ed invia il risultato, sempre tramite UDP, ad AttA
13. AttA se legge la risposta invia il risultato ad A, altrimenti può ritentare il passo 5 a seconda del numero di volte scritto nel proprio file di configurazione
14. A riceve la risposta da AttA, se questa è positiva invia una stringa "GNUTELLA/0.6 200 OK" come in (4) dopo aver parsato l'header inviatogli da B, altrimenti chiude la connessione
15. A invia un header con delle specifiche che possono essere utili a B come in (3)
16. Sia A che B si scambiano messaggi utilizzando le specifiche ottenute ai punti (3) e (5)

L'architettura di un handshake sicuro è mostrata in fig. 5.15

Analizzando la figura, un esempio di messaggi scambiati potrebbero essere i seguenti. In corsivo possiamo vedere i campi aggiunti:

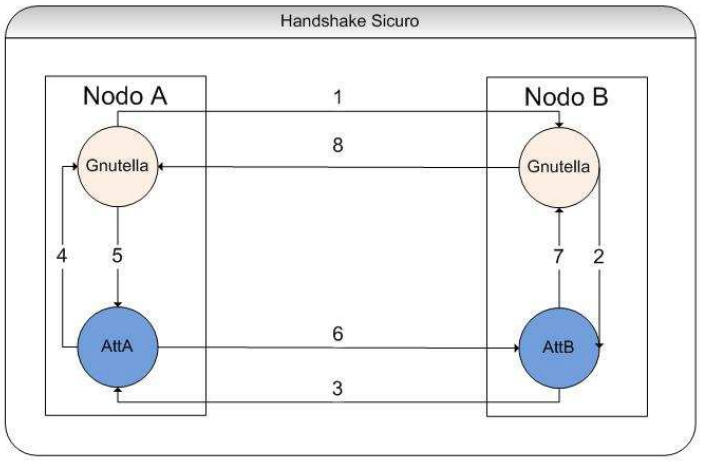


Figura 5.15: Architettura di un Handshake Sicuro

1)
GNUTELLA CONNECT/0.6
Node: 123.123.123.123:1234
Pong-Caching: 0.1
GGEP: 0.5
Ip-Att: 123.123.123.123
Port-Att: 6666
AttEveryXPing: 5
Dom-Name: Foo

2)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_HANDSHAKE
Ip-Node: 123.123.123.123
Port-Node: 1234
Ip-Att: 123.123.123.123
Port-Att: 6666
AttEveryXPing: 5
Dom-Name: Foo

3)
ATTESTATION MESSAGE
Code-Message: REQUEST_ATTESTATION
Dom-Name: Foo

4-5)
Chiamata della libreria Xen-VMI per l'attestazione di Foo

6)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_HANDSHAKE
Attestation: OK

7)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_HANDSHAKE
Attestation: OK

8)
GNUTELLA/0.6 200 OK
User-Agent: gtk-gnutella/0.95
Pong-Caching: 0.1
GGEP: 0.5
Ip-Att: 234.234.234.234
Port-Att: 8888
AttEveryXPing: 5
Dom-Name: Bar
Private-Data: 5ef89a

Una volta che Foo riceve il messaggio di OK, esegue gli stessi passi da 2 a 8 con i suoi dati.

Come possiamo notare quest'architettura garantisce che entrambi i nodi si attestino reciprocamente per garantire la sicurezza sia alla rete che al nuovo nodo che vuole entrare nella rete.

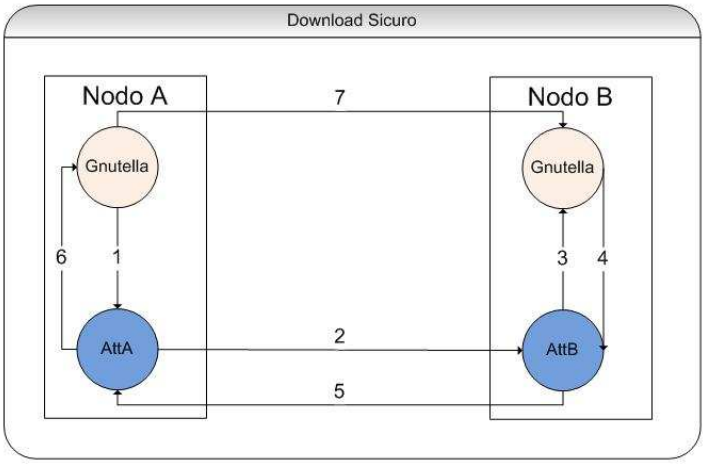


Figura 5.16: Architettura di un Download Sicuro

5.5.4 Download Sicuro

Per fase di download, non intendiamo solo l'atto in cui un peer "scarica" un file da un altro peer, ma ogni scambio di file e informazioni tra due peer. Ad esempio, esso può avvenire anche durante la richiesta di una query. Anche in questo caso, come per l'handshake, l'architettura non cambia, ma questa volta non si ha una doppia attestazione, ma solo l'attestazione del peer da cui si deve scaricare.
La fig. 5.16 è mostra l'architettura.
Nel seguito possiamo vedere nel dettaglio i messaggi scambiati tra i due peer.

1)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_DOWNLOAD
Ip-Node: 123.123.123.123
Port-Node: 1234
Dom-Name: Foo

2)
ATTESTATION MESSAGE
Code-Message: REQUEST_ATTESTATION
Dom-Name: Foo

3-4)
Chiamata della libreria Xen-VMI per l'attestazione di Foo

5)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_DOWNLOAD
Attestation: OK

6)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_DOWNLOAD
Attestation: OK

7)
GET /get/2468/Foobar.mp3 HTTP/1.1
User-Agent: Gnutella
Host: 123.123.123.123:6346
Connection: Keep-Alive
Range: bytes=0-

Come possiamo notare, i messaggi di download sono rimasti invariati, rispettando gli RFC di Gnutella. L'attestazione, in questo caso, avviene prima di richiedere un file al peer. In questo modo, il peer che vuole ottenere un file da un utente è sicuro che l'utente sia affidabile.

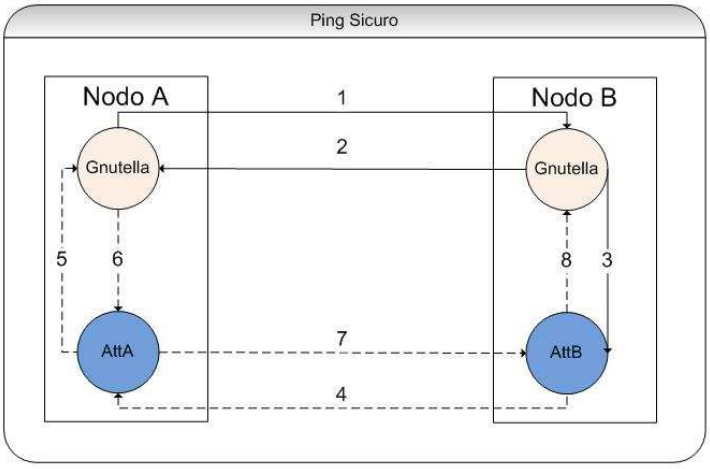


Figura 5.17: Architettura di Ping e Pong Sicuri

5.5.5 Ping e Pong Sicuri

Il ping e il pong sono i messaggi più scambiati all'interno delle reti P2P. Essi, inoltre, vengono scambiati anche con frequenza fissata. Per questo motivo, l'attestazione sui Ping e Pong è leggermente differente dagli altri due casi. Se così non fosse, la rete potrebbe saturarsi per i messaggi di attestazione. Per ovviare a questo problema, è stato deciso di attestare i nodi solo dopo un numero di ping decisi dalla rete. Si utilizza a questo scopo un campo supplementare, chiamato **AttEveryXPing**, nei messaggi scambiati tra i peer. L'architettura è mostrata in figura 5.17 Analizziamo i messaggi in dettaglio:

1)
Messaggio di PING

2)
Messaggio di PONG

3)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_PING
Ip-Node: 123.123.123.123
Port-Node: 1234
Ip-Att: 123.123.123.123
Port-Att: 7777
AttEveryXPing: 5
Dom-Name: Foo

4)
ATTESTATION MESSAGE
Code-Message: REQUEST_ATTESTATION
Dom-Name: Foo

5-6)
Chiamata della libreria Xen-VMI per l'attestazione di Foo

7)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_PING
Attestation: OK

8)
ATTESTATION MESSAGE
Code-Message: ATTESTATION_ON_PING
Attestation: OK

Possiamo notare che le linee 4-8 sono tratteggiate, perchè i messaggi sono scambiati solo se è stato raggiunto il numero di ping necessari per richiedere l'attestazione. L'attestazione sui ping ci permette di mantenere sempre sicura una rete P2P. Infatti, qualora un nodo sicuro diventasse inaffidabile la rete, nel giro di pochi minuti, riuscirebbe a rilevarlo ed a cacciarlo. Da notare, inoltre, che un peer risponde subito al Pong senza attendere il risultato dell'attestazione. Questo permette di non bloccare la rete, infatti il peer in attesa di un Pong potrebbe interpretare in modo erroneo il ritardo di risposta. In questo caso, si preferisce attestare il nodo in "parallelo" al Pong. Se il peer attestato risultasse non affidabile, la rete lo eliminerebbe comunque, non appena ricevuta l'informazione dall'attestatore.

5.5.6 L'Attestatore

L'attestatore è stato realizzato utilizzando il linguaggio C e permette la comunicazione tra la libreria Xen-VMI e il DomU su cui è installato Gnutella. Non appena avviato, l'attestatore crea un thread che gestisce le connessioni su una determinata porta, scelta da un file di configurazione. Il thread gestisce tutte le comunicazioni, in entrata, dirette all'attestatore. Quando riceve un messaggio, legge subito il **Code-Message** grazie al quale può gestire le varie richieste.

Sono cinque i tipi di codice che possono arrivare all'attestatore:

- **ATTESTATION_ON_HANDSHAKE:** indica che si sta richiedendo l'attestazione di un nodo che vuole unirsi alla rete
- **ATTESTATION_ON_DOWNLOAD:** indica che si sta richiedendo l'attestazione di un nodo da cui vogliamo ottenere un file o una query
- **ATTESTATION_ON_PING:** indica che si sta richiedendo l'attestazione di un nodo che ha mandato un messaggio di Ping
- **ATTESTATION_ON_REQUEST:** indica che si sta richiedendo l'attestazione di un nodo segnalato come non valido
- **REQUEST_ATTESTATION:** indica è stato richiesto l'attestazione del proprio DomU

L'attestatore ricorda tutti i nodi connessi al proprio client, grazie ad una lista come quella seguente:

Nome	**Descrizione**
Ip-Node	Indirizzo IP del nodo
Port-Node	Porta del nodo
Ip-Att	Indirizzo IP dell'attestatore del nodo
Port-Att	Porta dell'attestatore del nodo
Domain-Name	Nome della macchina virtuale su cui è installato il servent
AttEveryXPing	Numero di ping da effettuare prima di richiedere l'attestazione
nPing	Numero di ping effettuati
secure	Ultimo risultato dell'attestazione sul nodo
next	Puntatore al nodo successivo nella lista

L'attestatore, infine, quando scopre che un nodo non è affidabile, provvede, grazie alla lista mostrata sopra e al codice ATTESTATION_ON_REQUEST, ad informare tutti i nodi vicini che quel nodo non è più sicuro. In questo modo, ogni nodo che riceverà questo tipo di messaggio provvederà a controllare la veridicità del messaggio e, nel caso, ad allontanare il nodo sospetto.

Capitolo 6

Risultati Sperimentali

6.1 Efficienza del protocollo

Per quanto riguarda l'efficienza del protocollo, ovvero la capacità di rilevare peer non affidabili, possiamo dire che tutte le interazioni fra i nodi sono controllate, poichè, handshake, push, query e ping sono sempre preceduti da un attestazione del nodo. L'unica eccezione è quella del ping, che viene attestato in "parallelo" per migliorare le prestazioni.

Quindi, l'efficienza del protocollo è data solo dall'efficienza dell'introspettore Xen-VMI.
Per testare l'introspettore sono stati eseguiti alcuni test:

- Modifica di una entry nella tabella delle interruzioni *trap_table*
- Modifica di un puntatore nella tabella *sys_call_table* e del codice di una chiamata di sistema

L'introspettore rileva tutte le modifiche effettuate nei test, sia nelle chiamate di sistema che nell'handler delle chiamate di sistema. Inoltre, l'introspettore rileva le modifiche ai puntatori nella tabella delle chiamate di sistema o nella tabella contenente le entry per le interruzioni. Infine, l'introspettore rileva sia che un modulo inserito non è autorizzato sia che il codice di un modulo autorizzato è stato modificato.

Non appena l'introspettore rileva un anomalia, l'attestatore provvede ad informare non solo il suo nodo ma anche tutti i nodi connessi. In questo modo, l'isolamento di un nodo non sicuro avviene in un brevissimo lasso di tempo, garantendo l'efficienza e la sicurezza della rete P2P.

6.2 Prestazioni del protocollo

Per valutare le prestazioni del protocollo dobbiamo valutare quanto overhead abbiamo introdotto rispetto alla normale esecuzione del protocollo. Per fare questo, controlliamo tutto il codice che abbiamo aggiunto.

Prima di ogni interazione tra due nodi è stata aggiunta la richiesta di attestazione, tramite questo codice:

```
g_message("*** Attestation node %s is starting ***",host_addr_to_string(n->addr));
nodeOk = send_to_attestator(ATTESTATION_ON_*,host_addr_to_string(n->addr),n->port,n->att_addr,n->att_port,
GNET_PROPERTY(attestation_xPing),n->dom_name)
if (!nodeOk) { /* Attestazione fallita */
      g_message("*** %s is not a secure node ... kicked! ***",host_addr_to_string(n->addr));
      n->flags |= NODE_F_NOREAD;
      /* Rimozione + Ban del nodo */
      node_remove(n, _("Kicked!"));
      ban_record(n->addr,"NotSecure");
      goto free_gnet_response;
}else
      g_message("*** %s is a secure node ... sending OK message! ***",host_addr_to_string(n->addr));
```

Dove ATTESTATION_ON_* sta a significare ATTESTATION_ON_HANDSHAKE, ATTESTATION_ON_DOWNLOAD e ATTESTATION_ON_PING a seconda dell'interazione.

Dal codice possiamo notare che il corpo dell'IF introduce un ritardo trascurabile. Analizziamo la funzione *send_to_attestator*, che invia la richiesta di attestazione all'attestatore.

```
gboolean send_to_attestator(int code, const gchar * ip_node, int port_node, const gchar * ip_att, int port_att, int
xPing, const gchar * domain) {
      int sd,bsent,nattempt;
      struct sockaddr_in attestator;
      int slen = sizeof(attestator);
      char * msg, *buff;
      gboolean result;
      struct timeval tv;
      tv.tv_sec = GNET_PROPERTY(num_sec_wait);
      tv.tv_usec = 0;
      addr_init(&attestator,ATTESTATOR_PORT,inet_addr(ATTESTATOR_IP));
      msg = (char*) malloc(MAXSIZE*sizeof(char));
      memset(msg, '\0', MAXSIZE + 1);
      result = FALSE;
      switch(code){
      case ATTESTATION_ON_HANDSHAKE:
      sprintf(msg,"ATTESTATION MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Ip-Att: %s\r\n"
      "Port-Att: %d\r\n"
      "AttEveryXPing: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_HANDSHAKE,ip_node,port_node,ip_att,port_att,xPing,domain);
      break;
      case ATTESTATION_ON_DOWNLOAD:
      sprintf(msg,"ATTESTATION MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_DOWNLOAD,ip_node,port_node,domain);
      break;
      case ATTESTATION_ON_PING:
      sprintf(msg,"ATTESTATION MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Ip-Att: %s\r\n"
      "Port-Att: %d\r\n"
      "AttEveryXPing: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_PING,ip_node,port_node,ip_att ?  ip_att :  "999.999.999.999",port_att,xPing,domain ?
domain : "noDomain");
      break;
      }
      if ((sd=socket(AF_INET,SOCK_DGRAM,0))<0){
      g_warning("Error while conctacting attestator\n");
      return FALSE;
      }
      nattempt = 0;
      setsockopt(sd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
send_request:
      sendto(sd,msg,MAXSIZE,0,&attestator, sizeof(attestator));
```

```
fprintf(stdout,"SEND_TO_ATTESTATOR: ATTESTAZIONE RICHIESTA:\n%s",msg);
switch(code){
case ATTESTATION_ON_HANDSHAKE:
buff = (char*) malloc(MAXSIZE*sizeof(char));
memset(buff, '\0', MAXSIZE + 1);
bsent = recvfrom(sd,buff,MAXSIZE,0,(struct sockaddr *)&attestator,&slen);
if ( errno == EWOULDBLOCK ){
if ( bsent < 0 ){
if ( GNET_PROPERTY(num_attempt_udp) > ++nattempt ){
goto send_request;
}else
return FALSE;
}
}
fprintf(stdout,"SEND_TO_ATTESTATOR: RESPONSE HAND:\n%s",buff);
result = strcmp(get_param(buff,"Attestation"),"ERR") ? TRUE : FALSE;
break;
case ATTESTATION_ON_DOWNLOAD:
buff = (char*) malloc(MAXSIZE*sizeof(char));
memset(buff, '\0', MAXSIZE + 1);
bsent = recvfrom(sd,buff,MAXSIZE,0,(struct sockaddr *)&attestator,&slen);
if ( errno == EWOULDBLOCK ){
if ( bsent < 0 ){
if ( GNET_PROPERTY(num_attempt_udp) > ++nattempt ){
goto send_request;
}else
return FALSE;
}
}
fprintf(stdout,"SEND_TO_ATTESTATOR: RESPONSE DOWN:\n%s",buff);
result = strcmp(get_param(buff,"Attestation"),"ERR") ? TRUE : FALSE;
break;
}
close(sd);
return result;
}
```

Il codice evidenzia che viene creato un messaggio da mandare all'attestatore utilizzando il protocollo UDP. Questo aumenta sicuramente il tempo di attesa prima che il nodo mandi il messaggio di risposta. In particolare, l'utente può scegliere per quanti secondi attendere la risposta dell'attestatore e per quante volte deve richiedere l'attestazione di un nodo, in caso di *timeout*, prima di dichiare non valido un peer. Questa scelta può essere fatta tramite due opzioni nel file di configurazione, rispettivamente, *num_sec_wait* e *num_attempt_udp*. E' chiaro come una scelta sbagliata di questi parametri incida negativamente sulle prestazioni del protocollo, nel caso in cui l'attestatore non fosse in grado di dare subito il risultato dell'attestazione. E' da notare, infatti, che la *recvfrom* è una chiamata bloccante, quindi, l'applicazione sarà bloccata per tutto il tempo di attesa della risposta.

Per terminare la valutazione della performance dobbiamo valutare, infine, il tempo necessario all'attestatore di un peer per attestare un altro.
Osservando la parte del codice dell'attestatore che ci interessa, notiamo che il gestore delle richieste si compone solo di uno switch, quindi trascurabile, in quanto ci interessa sapere, invece, il tempo impiegato ad inviare la richiesta all'altro attestatore ed a ricevere la risposta. Osserviamo quindi la funzione corrispondente, chiamata *request_attestation*.

```
int request_attestation(char * ip, int port, char * domain){
    struct sockaddr_in attserver;
    struct timeval tivvu;
    socklen_t sock_size = sizeof(struct sockaddr_in);
    int sockd,byte,nattempt,result;
    int lenght = sizeof(attserver);
    char * mex;
    result = FALSE;
    if ( strcmp(ip,"999.999.999.999") ){
    addr_init(&attserver,port,inet_addr(ip));
    nattempt = 0;
    /* Creazione del socket */
    if ((sockd=socket(AF_INET,SOCK_DGRAM,0)) < 0){
```

```
        fprintf (stderr,"Impossibile inizializzare il socket UDP &d\n", getsockname (sockd, (struct sockaddr*) &attserver,
&sock_size));
        exit(1);
        }
        tivvu.tv_sec = 1;
        tivvu.tv_usec = 0;
        setsockopt(sockd, SOL_SOCKET, SO_RCVTIMEO, &tivvu, sizeof(tivvu));
        mex = (char*) malloc(N*sizeof(char));
        memset(mex, '\0', N + 1);
        sprintf(mex,"ATTESTATION MESSAGE\r\n"
        "Code-Message: &d\r\n"
        "Dom-Name: &s\r\n"
        "\r\n",
        REQUEST_ATTESTATION,domain);
send_request_attemption:
        sendto(sockd,mex,N,0,(struct sockaddr*) &attserver, sizeof(attserver));
        memset(mex, '\0', N + 1);
        byte = recvfrom(sockd, mex, N, 0, (struct sockaddr*) &attserver, &lenght);
        if ( errno == EWOULDBLOCK ){
        if ( byte < 0 ){
        if ( 3 > ++nattempt ){
        goto send_request_attemption;
        }else
        return FALSE;
        }
        }
        result = strcmp(get_param(mex,"Attestation"),"ERR") ? TRUE : FALSE;
        }
        return result;

        }
```

Anche in questo caso notiamo l'inserimento di un messaggio da inviare all'attestatore del peer che si vuole attestare. La velocità di invio e di risposta dipendono dalla congestione della rete.

Quindi, ricapitolando, abbiamo introdotto, per ogni interazione tra peer, quattro messaggi aggiuntivi, due di richiesta e due di risposta. La latenza di tutti i messaggi dipende esclusivamente dalla congestione della rete per i messaggi esterni e dalla velocità della scheda di rete per i messaggi interni. Inoltre, dobbiamo considerare anche il tempo impiegato per l'attestazione vera e propria, effettuata dalla Xen-VMI. Il test, effettuato dagli sviluppatori della libreria, ci assicura che al caso peggiore abbiamo una degradazione del 10% rispetto al caso in cui la libreria non viene utilizzata.

E' chiaro, dunque, che i test effettuati ci assicurano che il protocollo sviluppato non introduce un sostanziale ritardo. Non si può trascurare però la congestione della rete, infatti, una qualsiasi interazione tra due peer utilizzerebbe normalmente due messaggi mentre, col protocollo sviluppato, vengono scambiati sei messaggi al caso ottimo. Questo ci fa capire come in una rete, ad esempio di soli 50 peer, per una sola interazione, viaggino circa 300 messaggi a fronte dei 100 nel caso migliore.

6.3 Limiti del protocollo

Per completare la valutazione delle prestazioni dobbiamo considerare due tipi di limiti nel protocollo.

Il primo riguarda la congestione della rete, che, come abbiamo visto nel paragrafo precedente, limita il numero di peer che possono essere connessi nella rete. Infatti, nel caso di centinaia di peer connessi, i messaggi scambiati salirebbero velocemente alle migliaia, a questo punto inizierebbero a scattare i timeout degli attestatori con conseguente rinvio della richiesta. Nel caso peggiore avremmo una rete satura di richieste, a cui gli attestatori non risponderanno o risponderanno troppo in ritardo, fino ad arrivare all'assurdo in cui ogni peer assume, erroneamente, non sicuri tutti gli altri peer, rendendo inconsistente la rete stessa.

L'altro tipo di limite riguarda la libreria d'introspezione, infatti, al momento, ci sono alcuni attacchi che Xen-VMI non riesce a rilevare. Ad esempio, ci sono alcune regioni critiche del kernel che non sono protette. Inoltre, qualsiasi modifica illecita ai dati dinamici in memoria, come ad esempio lo stack del kernel, non viene rilevata.

Infine, un altro problema nasce se l'attaccante è a conoscenza del fatto che il sistema operativo è in esecuzione all'interno di una MV, in questo caso infatti l'attaccante può tentare di attaccare direttamente il Dom0, che noi abbiamo assunto essere sicuro.

Appendice A

Codice Attestatore

A.1 attestatore.c:

```
 /* File : attestatore.c
    Autore: Federico Tonelli
    Si dichiara che il contenuto di questo file e' in ogni sua parte opera
    originale dell'autore sopracitato. */

#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <dirent.h>
#include <mcheck.h>
#include <netdb.h>

/* Port for Peer Connection*/
#define PORT  9999

/* Ip Client Gnutella */
#define gnutellaClient  "192.168.10.11"

/* Message */
#define N   1024

#define ATTESTATION_ON_HANDSHAKE   1
#define ATTESTATION_ON_DOWNLOAD    2
#define ATTESTATION_ON_PING    3
#define ATTESTATION_ON_REQUEST     4
#define REQUEST_ATTESTATION    5

#define FALSE         0
#define TRUE          1

#define REQUEST "/xen/xen_vmi/xen_check_dom -d"

/* Struttura dei nodi */
typedef struct node_t{
  char * ip_node;
  int port_node;
```

```
  char * ip_att;
  int port_att;
  char * domain_name;

  int AttEveryXPing;
  int nPing;

  int secure;

  struct node_t * next;
}node_t;

/* Maschera dei segnali */
static sigset_t set;

node_t * nodi;

/* HANDLER */
static void *sig_gestore(void *pidFather);
void *peer_gestore(void *param);

void addr_init (struct sockaddr_in *addr, int port, long int ip){
        addr->sin_family=AF_INET;
        addr->sin_port = htons ((u_short) port);
        addr->sin_addr.s_addr=ip;
  memset(&(addr->sin_zero), '\0', 8);
}

char * get_param(char * messaggio, char * daCercare){

  char * pch, *pch2;

  pch = strdup(messaggio);
  pch2 = strstr(pch,daCercare);
  if (!pch2)
    return NULL;
  pch = strtok(pch2," ");
  pch = strtok(NULL,"\r\n");

  return pch;
}

/**
 Ritorna il numero di nodi da una lista

 \param nodi il puntatore al puntatore alla testa della lista

 \retval num il numero di elementi
 */
int numNodi(node_t * nodi)
{
    if (nodi == NULL)
  return 0;
    else
  return (1 + numNodi(nodi->next));
}

/**
 Aggiunge un nuovo nodo alla lista dei nodi

 \param nodi il puntatore al puntatore alla testa della lista
              in cui inserire il nodo

 \param ev puntatore all'evento da inserire

 \retval 0 se l'inserzione e' andata a buon fine
 \retval -1 altrimenti
 */
```

```
int add_node(node_t ** nodi, char * peer_ip, char * peer_port, char *
    att_ip, char * att_port, char * xPing, char * domain)
{
      int k;
      node_t *node;

      node = *nodi;

      k = numNodi(node);

      /* Caso di prima allocazione */
      if (node == NULL) {
    node = (node_t *) malloc(100 * sizeof(node_t));
    node->ip_node = NULL;
    node->ip_att = NULL;
  }
  if (node->ip_node == NULL)
    k = 0;


      /* Copio il nodo in memoria */
      node[k].nPing = 0;
      node[k].secure = 1;
      node[k].ip_node = peer_ip ? strdup(peer_ip) : NULL;
      node[k].port_node = peer_port ? strtol(peer_port,NULL,10) : 0;
  node[k].ip_att = att_ip ? strdup(att_ip) : "999.999.999.999";
  node[k].port_att = att_port ? strtol(att_port,NULL,10) : 0;
  node[k].domain_name = domain ? strdup(domain) : "No";
  node[k].AttEveryXPing = strtol(xPing,NULL,10);
      node[k].next = NULL;

      if (k > 0)
    node[k - 1].next = &node[k];  /* Setto il puntatore precedente */

      *nodi = node;
      return 0;
}

/** Restituisce la lista degli eventi che contiene tutti gli eventi che
    hanno un match con una data
 \param ip l'ip del nodo di cui dobbiamo cercare il match
 \param nodi la lista in cui cercare

 \retval >=0 ovvero la posizione in cui si trova se trovato
 \retval -1 altrimenti
 */
int is_node(char * ip, node_t * nodi)
{
      int n, k, j;

  n = -1;

      if (nodi != NULL) {
    if (nodi->ip_node == NULL)
      return -1;

    k = numNodi(nodi);
    for (j = 0; j < k; j++)
      if (!strcmp(nodi[j].ip_node,ip)){
        n = j;
        break;
      }
      }
      return n;
}

/** Rimuove dalla lista il nodo passato per parametro
 \param ip il nodo da cercare
 \param nodi la lista dei nodi in cui cercare
```

```
 \retval p il puntatore alla lista aggiornata
 */
node_t *remove_node(char * ip, node_t * nodi)
{
      int n, j, k;

      if (nodi != NULL) {
    k = numNodi(nodi);
    n = is_node(ip,nodi);
    if (-1 != n) {
      for (j = n; j < k - 1; j++) {
            nodi[j] = nodi[j + 1];
        if (j)
              nodi[j - 1].next = &nodi[j];
      }
      nodi[j].ip_node = NULL;
      nodi[j].ip_att = NULL;
      nodi[j].domain_name = NULL;
      nodi[j].port_node = 0;
      nodi[j].port_att = 0;
      nodi[j].AttEveryXPing = 0;
      nodi[j].next = NULL;
      nodi[j].secure = 0;
      nodi[j - 1].next = NULL;
        }
  }
      return nodi;
}

void send_kick(char * ip){
        struct sockaddr_in attestatore;
        char * msg;
        int sd,i;

        sd=socket(AF_INET,SOCK_DGRAM,0);
        msg = (char*) malloc(N*sizeof(char));
  memset(msg, '\0', N + 1);
  sprintf(msg,"ATTESTATION_MESSAGE\r\n"
    "Code-Message: %d\r\n"
    "Ip-Node: %s\r\n"
    "\r\n",
    ATTESTATION_ON_REQUEST,ip);

        for(i = 0; i < numNodi(nodi); i++){
          if (strcmp(nodi[i].ip_node,ip)){
            addr_init(&attestatore,nodi[i].port_att,inet_addr(nodi[i].
                ip_att));
      sendto(sd,msg,N,0,(struct sockaddr*) &attestatore, sizeof(attestatore
          ));
    }
  }

  nodi = remove_node(ip,nodi);

  free(msg);
}

int request_attestation(char * ip, int port, char * domain){
  struct sockaddr_in attserver;
  struct timeval tivvu;
  socklen_t sock_size = sizeof(struct sockaddr_in);
  int sockd,byte,nattempt,result;
  int lenght = sizeof(attserver);
  char * mex;

  result = TRUE;

  if ( strcmp(ip,"999.999.999.999") ){
    addr_init(&attserver,port,inet_addr(ip));
    nattempt = 0;
```

```
    /* Creazione del socket */
    if ((sockd=socket(AF_INET,SOCK_DGRAM,0)) < 0){
            fprintf (stderr,"Impossibile inizializzare il socket UDP %d\n",
                        getsockname (sockd, (struct sockaddr*) &
                            attserver, &sock_size));
            exit(1);
    }
    tivvu.tv_sec = 1;
    tivvu.tv_usec = 0;

    setsockopt(sockd, SOL_SOCKET, SO_RCVTIMEO, &tivvu, sizeof(tivvu));

    mex = (char*) malloc(N*sizeof(char));
    memset(mex, '\0', N + 1);

    sprintf(mex,"ATTESTATION_MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      REQUEST_ATTESTATION,domain);

send_request_attemption:

    sendto(sockd,mex,N,0,(struct sockaddr*) &attserver, sizeof(attserver));

    memset(mex, '\0', N + 1);
    byte = recvfrom(sockd, mex, N, 0, (struct sockaddr*) &attserver, &
        lenght);
    if ( errno == EWOULDBLOCK ){
      if ( byte < 0 ){
        if ( 3 > ++nattempt ){
          goto send_request_attemption;
        }else
          return FALSE;
      }
    }

    result = strcmp(get_param(mex,"Attestation"),"ERR") ? TRUE : FALSE;
  }

  return result;
}

int node_attestation(char * domName){
  char* buff;
  char* req;
  char* ris;
  int i,r,idDom;
  FILE *fp;

  buff = (char*) malloc(100*sizeof(char));
  ris = (char*) malloc(100*sizeof(char));
  req = (char*) malloc(100*sizeof(char));

  sprintf(req,"xm domid %s",domName);

  fp=popen(req, "r");
  fgets (buff , 100 , fp);
  idDom = strtol(buff,NULL,10);

  memset(req, '\0', 100);
  sprintf(req,"%s %d -s",REQUEST,idDom);


  fp=popen(req, "r");
  for(i = 0; i < 4; i++)
    fgets (buff , 100 , fp);

  sscanf(buff,"%*s %s", ris);
```

```
  r = !strcmp(ris,"correct");


  if (r){
    memset(req, '\0', 100);
    sprintf(req,"%s %d -k",REQUEST,idDom);
    fp=popen(req, "r");
    for(i = 0; i < 4; i++)
      fgets (buff , 100 , fp);

    sscanf(buff,"%*s %s", ris);
    r = !strcmp(ris,"correct");

    if(r){
      memset(req, '\0', 100);
      sprintf(req,"%s %d -i",REQUEST,idDom);
      fp=popen(req, "r");
      for(i = 0; i < 4; i++)
        fgets (buff , 100 , fp);

      sscanf(buff,"%*s %s", ris);
      r = !strcmp(ris,"correct");
    }
  }

  free(req);
  free(ris);
  free(buff);

  return r;
}

int main(int argc, char **argv)
{
  pthread_t peer_handler,sig_handler;
  int x,ok;

  if ( getuid() ){
    printf("You must run as root!\n");
    return -1;
  }

  nodi = NULL;

  /*Inizializzo la maschera per la gestione dei segnali */

  sigemptyset(&set);

  sigaddset(&set, SIGINT);
      sigaddset(&set, SIGTERM);
  sigaddset(&set, SIGPIPE);

  pthread_sigmask(SIG_SETMASK, &set, NULL);

  /* creo un thread per la gestione dei segnali */
  if ((x = pthread_create(&sig_handler, NULL, &sig_gestore,(void *)
      pthread_self())) != 0)
    {
      fprintf(stderr, "Errore nella creazione del thread handler\n");
      exit(-1);
    }

  /* creo un thread per la gestione dei messaggi con i Peer */
  if ((x = pthread_create(&peer_handler, NULL, &peer_gestore,(void *) NULL)
      ) != 0)
    {
      fprintf(stderr, "Errore nella creazione del thread handler\n");
      exit(-1);
    }
      if ((ok = pthread_detach(peer_handler)) != 0)
```

```
    {
      fprintf(stderr, "Errore nella detach del thread\n");
      return 1;
        }
  pthread_join(sig_handler,NULL);
  return 0;
}


/* @description -- handler per la gestione dei segnali
   @param pidFather -- pid del Main
   @returns -- (-1) si sono verificati errori
         (0) se OK
*/
static void *sig_gestore(void *pidFather)
{

  int signal, sign;

  sign = 0;

  while (!sign) {

    sigwait(&set, &signal);
    if ((signal == SIGINT) || (signal == SIGTERM)){
      sign = 1;
      fprintf(stdout, "Attestator: got signal %d, terminating!\n",signal);
      /* Free di tutte le strutture */
    }
  }

  /* distruggo il server */

      fprintf(stdout, "Attestator: destroyed\n");

      pthread_kill((int)pidFather, SIGKILL);

  return (void *) 0;
}

/* @description -- handler per la gestione dei messaggi con i Peer e con
    gli Attestatori
   @returns -- (>0) si sono verificati errori
         (0) se OK
*/
void *peer_gestore(void *param)
{
  int sd,pos,sock;
       struct sockaddr_in server,gnuclient;
  int slen=sizeof(server);
       char *buff;
  short int nodeOk;

  char *code, *ip_node, *port_node, *ip_att, *port_att, *attxPing, *domName
      ;

       addr_init(&server,PORT,INADDR_ANY);
       addr_init(&gnuclient,6666,inet_addr(gnutellaClient));

       /* Creazione del socket */
       if ((sd=socket(AF_INET,SOCK_DGRAM,0)) < 0){
              fprintf (stderr,"Impossibile inizializzare il socket UDP %d
                  \n",
                              getsockname (sd, (struct sockaddr*) &server
                                  , &slen));
              exit(1);
       }

       /* Lego il socket appena creato all'indirizzo del server */
       if (bind(sd, (struct sockaddr*) &server, sizeof(server)) < 0){
```

```
                fprintf (stderr,"Impossibile aprire una connessione sulla
                    porta %d\n",PORT);
                exit(2);
        }

        printf ("Server in ascolto per PEER sulla porta %d\n",PORT);

        /* Accetto connessioni finchÃ© ce ne sono */
        while(1){
receive:

    /* Alloco memoria per ricevere la stringa */
                buff = (char*) malloc(N*sizeof(char));
    memset(buff, '\0', N + 1);

              recvfrom(sd, buff, N, 0, (struct sockaddr *)&server, &slen);

              printf("Received packet from %s:%d\nData: %s\n\n",
      inet_ntoa(server.sin_addr), ntohs(server.sin_port), buff);

    code = get_param(buff,"Code-Message");
    if(!code){
      free(buff);
      goto receive;
    }

    ip_node = get_param(buff,"Ip-Node");
    pos = is_node(ip_node,nodi);

    switch (atoi(code)){

      case ATTESTATION_ON_HANDSHAKE:

        if ( -1 == pos ){

          port_node = get_param(buff,"Port-Node");
          ip_att = get_param(buff,"Ip-Att");
          port_att = get_param(buff,"Port-Att");
          attxPing = get_param(buff,"AttEveryXPing");
          domName = get_param(buff,"Dom-Name");

          add_node(&nodi,ip_node,port_node,ip_att,port_att,attxPing,domName
              );
          pos = is_node(ip_node,nodi);
        }

        if (0 != nodi[pos].secure){
          /* Attestazione */
          nodeOk = request_attestation(nodi[pos].ip_att,nodi[pos].port_att,
              nodi[pos].domain_name);
        }
        nodi[pos].secure = nodeOk;

        memset(buff, '\0', N + 1);

        sprintf(buff,"ATTESTATION_MESSAGE\r\n"
        "Code-Message: %s\r\n"
        "Attestation: %s\r\n"
        "\r\n",
        code,nodi[pos].secure ? "OK" : "ERR");

        sendto(sd,buff,N,0,(struct sockaddr*) &server,slen);

        if ( 0 == nodi[pos].secure )
          send_kick(ip_node);

      break;

      case ATTESTATION_ON_DOWNLOAD:
```

```
    if ( -1 == pos ){

      port_node = get_param(buff,"Port-Node");
      ip_att = get_param(buff,"Ip-Att");
      port_att = get_param(buff,"Port-Att");
      attxPing = get_param(buff,"AttEveryXPing");
      domName = get_param(buff,"Dom-Name");

      add_node(&nodi,ip_node,port_node,ip_att,port_att,attxPing,domName
          );
      pos = is_node(ip_node,nodi);
    }

    if (0 != nodi[pos].secure){
      /* Attestazione */
      nodeOk = request_attestation(nodi[pos].ip_att,nodi[pos].port_att,
          nodi[pos].domain_name);
    }
    nodi[pos].secure = nodeOk;

    memset(buff,'\0', N + 1);

    sprintf(buff,"ATTESTATION_MESSAGE\r\n"
    "Code-Message: %s\r\n"
    "Attestation: %s\r\n"
    "\r\n",
    code,nodi[pos].secure ? "OK" : "ERR");

    sendto(sd,buff,N,0,(struct sockaddr*) &server, slen);

    if ( 0 == nodi[pos].secure )
      send_kick(ip_node);

  break;

  case ATTESTATION_ON_PING:

    if ( -1 == pos ){

      port_node = get_param(buff,"Port-Node");
      ip_att = get_param(buff,"Ip-Att");
      port_att = get_param(buff,"Port-Att");
      attxPing = get_param(buff,"AttEveryXPing");
      domName = get_param(buff,"Dom-Name");

      add_node(&nodi,ip_node,port_node,ip_att,port_att,attxPing,domName
          );

      pos = is_node(ip_node,nodi);
    }

    nodi[pos].nPing++;

    if (nodi[pos].nPing >= nodi[pos].AttEveryXPing){
      if (0 != nodi[pos].secure){
        /* Attestazione */
        nodeOk = request_attestation(nodi[pos].ip_att,nodi[pos].
            port_att,nodi[pos].domain_name);
      }

      nodi[pos].secure = nodeOk;

      nodi[pos].nPing = 0;
      memset(buff,'\0', N + 1);

      sprintf(buff,"ATTESTATION_MESSAGE\r\n"
        "Code-Message: %s\r\n"
        "Ip-Node: %s\r\n"
        "Attestation: %s\r\n"
        "\r\n",
```

```
        code,ip_node,nodi[pos].secure ? "OK" : "ERR");

        sock=socket(AF_INET,SOCK_DGRAM,0);
        sendto(sock,buff,N,0,(struct sockaddr*) &gnuclient, sizeof(
            gnuclient));

        if ( 0 == nodi[pos].secure )
          send_kick(ip_node);

      }
    break;

    case ATTESTATION_ON_REQUEST:

      if ( -1 != pos ){

        if (0 != nodi[pos].secure){
          /* Attestazione */
          nodeOk = request_attestation(nodi[pos].ip_att,nodi[pos].
              port_att,nodi[pos].domain_name);
        }

        memset(buff,'\0', N + 1);

        sprintf(buff,"ATTESTATION_MESSAGE\r\n"
          "Code-Message: %s\r\n"
          "Attestation: %s\r\n"
          "Ip-Node: %s\r\n"
          "\r\n",
        code,nodi[pos].secure ? "OK" : "ERR",ip_node);

        sock=socket(AF_INET,SOCK_DGRAM,0);
        sendto(sock,buff,N,0,(struct sockaddr*) &gnuclient, sizeof(
            gnuclient));

        if ( 0 == nodi[pos].secure )
          send_kick(ip_node);
      }

    break;

    case REQUEST_ATTESTATION:

      /*
      * nodeOk = ATTESTAZIONE SGANDURRA
      */

      nodeOk = node_attestation(get_param(buff,"Dom-Name"));

      memset(buff,'\0', N + 1);

      sprintf(buff,"ATTESTATION_MESSAGE\r\n"
      "Code-Message: %s\r\n"
      "Attestation: %s\r\n"
      "\r\n",
      code,nodeOk ? "OK" : "ERR");

      sendto(sd,buff,N,0,(struct sockaddr*) &server,slen);

    break;

  }


  /* Scrivo su stdout la stringa appena inviata */
              printf ("-----\n%s-----\n",buff);
  free(buff);

    }/* Chiude il while(1) */
```

```
  return (void *) 0;
}
```

Appendice B

Modifiche al codice di Gnutella

B.1 \src\main.c:

```
....
riga 1332:

/* Creazione dell'handler per la gestione dei messaggi con l'attestatore */
if ((x = pthread_create(&hand_att, NULL, &handler_attestator,(void *) NULL)
    ) != 0){
    fprintf(stderr, "Errore nella creazione del thread handler\n");
    exit(-1);
  }
if ((x = pthread_detach(hand_att)) != 0){
    fprintf(stderr, "Errore nella detach del thread\n");
    exit(-1);
  }

....
```

B.2 \src\core\downloads.c:

```
...
riga 9328:

/* Attestazione in caso di richiesta di download di un file */
  g_message("*** Attestation node %s is starting ***",host_addr_to_string(s
      ->addr));

  nodeOk = send_to_attestator(ATTESTATION_ON_DOWNLOAD,host_addr_to_string(s
      ->addr),s->port,"999.999.999.999",0,0,NULL);

  if (!nodeOk) {  /* Attestazione fallita */
    g_message("*** %s is not a secure node ... kicked! ***",
        host_addr_to_string(s->addr));

    s->flags |= NODE_F_NOREAD;

    /* Rimozione + Ban del nodo */
    download_remove(d);
    ban_record(s->addr,"NotSecure");

  }else
    g_message("*** %s is a secure node ... sending GET/HEAD request! ***",
        host_addr_to_string(s->addr));

...
```

B.3 \src\core\nodes.h:

```
...
riga 272:

guint32 att_every_ping; /**< Total amount of ping to do before send the
    attestation*/

...
riga 505:

#define ATTESTATION_ON_HANDSHAKE  1
#define ATTESTATION_ON_DOWNLOAD   2
#define ATTESTATION_ON_PING       3
#define ATTESTATION_ON_REQUEST    4
#define REQUEST_ATTESTATION       5

#define HANDLER_PORT          6666

#define ATTESTATOR_IP      "192.168.10.10" /* Attestator Ip*/

char * domName; /* Name of Domain where gnutella is located */
int ATTESTATOR_PORT;  /* Attestator Port*/

...
riga 545:

void *handler_attestator();
void addr_init (struct sockaddr_in *addr, int port, long int ip);
char * get_param(char * messaggio, char * daCercare);
gboolean send_to_attestator(int code, const gchar * ip_node, int port_node,
     const gchar * ip_att, int port_att, int xPing,const gchar * domain);

...
```

B.4 \src\core\nodes.c:

```
...
riga 397:

static void
node_send_udp_ping(struct gnutella_node *n)
{

  udp_send_ping(NULL, n->addr, n->port, TRUE);

  send_to_attestator(ATTESTATION_ON_PING,host_addr_to_string(n->addr),n->
      port,n->att_addr,n->att_port,n->att_every_ping,n->dom_name);

}

void addr_init (struct sockaddr_in *addr, int port, long int ip){
        addr->sin_family=AF_INET;
        addr->sin_port = htons ((u_short) port);
        addr->sin_addr.s_addr=ip;
        memset(&(addr->sin_zero), '\0', 8);
}

char * get_param(char * messaggio, char * daCercare){

  char * pch, *pch2;

  pch = strdup(messaggio);
  pch2 = strstr(pch,daCercare);
  if (!pch2)
    return NULL;
  pch = strtok(pch2," ");
  pch = strtok(NULL,"\r\n");
```

```
  return pch;
}

/**
 * Sends a MESSAGE to the attestator.
 */
gboolean send_to_attestator(int code, const gchar * ip_node, int port_node,
     const gchar * ip_att, int port_att, int xPing, const gchar * domain)
{
  int sd,bsent,nattempt;
  struct sockaddr_in attestator;
  int slen = sizeof(attestator);
  char * msg, *buff;
  gboolean result;
  struct timeval tv;

  tv.tv_sec = GNET_PROPERTY(num_sec_wait);
  tv.tv_usec = 0;

  addr_init(&attestator,ATTESTATOR_PORT,inet_addr(ATTESTATOR_IP));

  msg = (char*) malloc(MAXSIZE*sizeof(char));
  memset(msg, '\0', MAXSIZE + 1);

  result = FALSE;

  switch(code){
    case ATTESTATION_ON_HANDSHAKE:

      sprintf(msg,"ATTESTATION_MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Ip-Att: %s\r\n"
      "Port-Att: %d\r\n"
      "AttEveryXPing: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_HANDSHAKE,ip_node,port_node,ip_att,port_att,xPing,
          domain);

    break;

    case ATTESTATION_ON_DOWNLOAD:

      sprintf(msg,"ATTESTATION_MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_DOWNLOAD,ip_node,port_node,domain);

    break;

    case ATTESTATION_ON_PING:

      sprintf(msg,"ATTESTATION_MESSAGE\r\n"
      "Code-Message: %d\r\n"
      "Ip-Node: %s\r\n"
      "Port-Node: %d\r\n"
      "Ip-Att: %s\r\n"
      "Port-Att: %d\r\n"
      "AttEveryXPing: %d\r\n"
      "Dom-Name: %s\r\n"
      "\r\n",
      ATTESTATION_ON_PING,ip_node,port_node,ip_att ? ip_att : "
          999.999.999.999",port_att,xPing,domain ? domain : "noDomain");
```

```
    break;
  }

  if ((sd=socket(AF_INET,SOCK_DGRAM,0))<0){
      g_warning("Error while conctacting attestator\n");
    return FALSE;
  }

  nattempt = 0;
  setsockopt(sd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

send_request:

  sendto(sd,msg,MAXSIZE,0,&attestator, sizeof(attestator));
  fprintf(stdout,"SEND_TO_ATTESTATOR: ATTESTAZIONE RICHIESTA:\n%s",msg);


  switch(code){
    case ATTESTATION_ON_HANDSHAKE:

      buff = (char*) malloc(MAXSIZE*sizeof(char));
      memset(buff, '\0', MAXSIZE + 1);
          bsent = recvfrom(sd,buff,MAXSIZE,0,(struct sockaddr *)&attestator
              ,&slen);
          if ( errno == EWOULDBLOCK ){
            if ( bsent < 0 ){
              if ( GNET_PROPERTY(num_attempt_udp) > ++nattempt ){
                goto send_request;
              }else
                return FALSE;
            }
          }

      fprintf(stdout,"SEND_TO_ATTESTATOR: RESPONSE_HAND:\n%s",buff);

      result = strcmp(get_param(buff,"Attestation"),"ERR") ? TRUE : FALSE;

    break;

    case ATTESTATION_ON_DOWNLOAD:

      buff = (char*) malloc(MAXSIZE*sizeof(char));
      memset(buff, '\0', MAXSIZE + 1);
          bsent = recvfrom(sd,buff,MAXSIZE,0,(struct sockaddr *)&attestator
              ,&slen);
          if ( errno == EWOULDBLOCK ){
            if ( bsent < 0 ){
              if ( GNET_PROPERTY(num_attempt_udp) > ++nattempt ){
                goto send_request;
              }else
                return FALSE;
            }
          }

      fprintf(stdout,"SEND_TO_ATTESTATOR: RESPONSE_DOWN:\n%s",buff);

      result = strcmp(get_param(buff,"Attestation"),"ERR") ? TRUE : FALSE;

    break;

  }

  close(sd);

  return result;

}

...
riga 2744:
```

```
void *handler_attestator()
{
  int sd;
    struct sockaddr_in server,client;
    char *buff;
  char *code, *ip_node;
  int slen = sizeof(server);
  GSList *sl;

    addr_init(&server,HANDLER_PORT,INADDR_ANY);

    /* Creazione del socket */
    if ((sd=socket(AF_INET,SOCK_DGRAM,0)) < 0){
            g_warning("Impossibile inizializzare il socket UDP");
            exit(1);
    }

    /* Lego il socket appena creato all'indirizzo del server */
    if (bind(sd, (struct sockaddr*) &server, sizeof(server)) < 0){
            g_warning("Impossibile aprire una connessione sulla porta %d\n"
                ,HANDLER_PORT);
            exit(2);
    }

  g_message("Demone in ascolto sulla porta %d per messaggi dall'attestatore
      \n",HANDLER_PORT);

    /* Accetto connessioni finche' ce ne sono */
    while(1){
receive:
            /* Alloco memoria per ricevere la stringa */
            buff = (char*) malloc(MAXSIZE*sizeof(char));
      memset(buff, '\0', MAXSIZE + 1);

            /* Ricevo la stringa */
            recvfrom(sd,buff,MAXSIZE,0,(struct sockaddr *)&server, &slen);

            printf("Received packet from %s:%d\nData: %s\n\n",
      inet_ntoa(client.sin_addr), ntohs(client.sin_port), buff);


      code = get_param(buff,"Code-Message");
      if(!code){
        free(buff);
        goto receive;
      }else

        switch(atoi(code)){

          case ATTESTATION_ON_PING:

            if ( strcmp(get_param(buff,"Attestation"),"OK") ){

              ip_node = get_param(buff,"Ip-Node");
              for (sl = sl_nodes; sl; sl = g_slist_next(sl)) {
                struct gnutella_node *list = sl->data;

                if ( !strcmp(host_addr_to_string(list->addr),ip_node) ){
                  list->flags |= NODE_F_NOREAD;
                  node_remove(list, _("Kicked!"));
                  ban_record(list->addr,"NotSecure");
                }
              }
            }

          break;

          case ATTESTATION_ON_REQUEST:
```

```
            if ( strcmp(get_param(buff,"Attestation"),"OK") ){

              ip_node = get_param(buff,"Ip-Node");
              for (sl = sl_nodes; sl; sl = g_slist_next(sl)) {
                struct gnutella_node *list = sl->data;

                if ( !strcmp(host_addr_to_string(list->addr),ip_node) ){
                  list->flags |= NODE_F_NOREAD;
                  node_remove(list, _("Kicked by Request!"));
                  ban_record(list->addr,"NotSecure");
                }
              }
            }

            break;
          }

          free(buff);

  }/* Chiude il while(1) */
  close(sd);

return (void *) 0;
}

...
riga 5012:

/**
 * This routine is called to process a 0.6+ handshake header.
 *
 * It is either called to process the reply to our sending a 0.6 handshake
 * (outgoing connections) or to parse the initial 0.6 headers (incoming
 * connections).
 */
static void
node_process_handshake_header(struct gnutella_node *n, header_t *head)
{
  static const gchar gtkg_vendor[] = "gtk-gnutella/";
  static const size_t gnet_response_max = 16 * 1024;
  gchar *gnet_response;
  size_t rw;
  gint sent;
  const gchar *field;
  gboolean incoming = (n->flags & NODE_F_INCOMING);
  gboolean nodeOk = FALSE;
  const gchar *what = incoming ? "HELLO reply" : "HELLO acknowledgment";
  const gchar *compressing = "Content-Encoding: deflate\r\n";

  if (GNET_PROPERTY(node_debug)) {
    g_message("got %s handshaking headers from node %s:",
      incoming ? "incoming" : "outgoing",
      host_addr_to_string(n->addr));
    if (!incoming)
      dump_hex(stdout, "Status Line", getline_str(n->socket->getline),
        MIN(getline_length(n->socket->getline), 80));
    g_message("------ Header Dump:");
    header_dump(head, stderr);
    g_message("------");
    fflush(stderr);
  }

  if (in_shutdown) {
    node_send_error(n, 503, "Servent Shutdown");
    node_remove(n, _("Servent Shutdown"));
    return;      /* node_remove() has freed s->getline */
  }

  /*
   * Handle common header fields, non servent-specific.
```

```
 */

/* User-Agent -- servent vendor identification */

field = header_get(head, "User-Agent");
if (field) {
  const gchar *token = header_get(head, "X-Token");
  if (!version_check(field, token, n->addr))
    n->flags |= NODE_F_FAKE_NAME;
      node_set_vendor(n, field);
}

if (NULL == field || !is_strprefix(field, gtkg_vendor)) {
  socket_disable_token(n->socket);
}

/*
 * Spot remote GTKG nodes (even if faked name).
 */

if (n->vendor != NULL) {
  if (
    is_strprefix(n->vendor, gtkg_vendor) ||
    (*n->vendor == '!' && is_strprefix(&n->vendor[1], gtkg_vendor))
  ) {
    version_t rver;

    n->flags |= NODE_F_GTKG;

    /*
     * Look for known bugs in certain older GTKG servents:
     */

    if (version_fill(n->vendor, &rver)) {
      /*
       * All versions prior 0.96u of 2005-10-02 are known to be
       * broken with respect to duplicate message handling.
       */

      if (rver.timestamp < 1128204000)
        n->attrs |= NODE_A_NO_DUPS;

      /*
       * Versions prior 0.96.2u of 2006-08-15 are broken when they
       * act as UP and perform dynamic queries for leaves: they will
       * ignore an "OOB results status" message with kept=0 and
       * terminate the dynamic query on timeout waiting for that
       * message.
       */

      if (rver.timestamp < 1155592800)
        n->attrs |= NODE_A_NO_KEPT_ZERO;
    }
  }
}

/* Pong-Caching -- ping/pong reduction scheme */

field = header_get(head, "Pong-Caching");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major != 0 && minor != 1)
    if (GNET_PROPERTY(node_debug)) g_warning(
      "node %s claims Pong-Caching version %u.%u",
      node_addr(n), major, minor);
  n->attrs |= NODE_A_PONG_CACHING;
}
```

```
/* X-Ultrapeer -- support for ultra peer mode */

field = header_get(head, "X-Ultrapeer");
if (field) {
  n->attrs |= NODE_A_CAN_ULTRA;
  if (0 == ascii_strcasecmp(field, "true"))
    n->attrs |= NODE_A_ULTRA;
  else if (0 == ascii_strcasecmp(field, "false")) {
    if (GNET_PROPERTY(current_peermode) == NODE_P_ULTRA)
      n->flags |= NODE_F_LEAF;
  }
} else {
  /*
   * BearShare 4.3.x decided to no longer send X-Ultrapeer on connection,
   * but rather include the X-Ultrapeer-Needed header.  Hopefully, only
   * their UPs will send back such a header.
   *    --RAM, 01/11/2003
   */

  field = header_get(head, "X-Ultrapeer-Needed");
  if (field)
    n->attrs |= NODE_A_CAN_ULTRA | NODE_A_ULTRA;
  else
    n->attrs |= NODE_A_NO_ULTRA;
}

/* Node -- remote node Gnet IP/port information */

if (incoming) {
  host_addr_t addr;
  guint16 port;

  /*
   * We parse only for incoming connections.  Even though the remote
   * node may reply with such a header to our outgoing connections,
   * if we reached it, we know its IP:port already!  There's no need
   * to spend time parsing it.
   */

  field = header_get(head, "Node");
  if (!field) field = header_get(head, "X-My-Address");
  if (!field) field = header_get(head, "Listen-Ip");

  if (field && string_to_host_addr_port(field, NULL, &addr, &port)) {
    if (n->attrs & NODE_A_ULTRA) {
      pcache_pong_fake(n, addr, port);  /* Might have free slots */
    }

    /*
     * Since we have the node's IP:port, record it now and mark the
     * node as valid: if the connection is terminated, the host will
     * be recorded amongst our valid set.
     *    --RAM, 18/03/2002.
     */

    n->gnet_port = port;
    if (host_addr_equal(addr, n->addr)) {
              node_ht_connected_nodes_remove(n->gnet_addr, n->gnet_port);

      n->gnet_addr = addr;       /* Signals: we know the port */
      n->gnet_pong_addr = addr;    /* Cannot lie about its IP */
      n->flags |= NODE_F_VALID;

              node_ht_connected_nodes_add(n->gnet_addr, n->gnet_port);
    }
    /* FIXME: What about LAN connections? Should we blindly accept
     *       the reported external address?
     */
  }
}
```

```
if (header_get_feature("tls", head, NULL, NULL)) {
  node_supports_tls(n);
}

/* Bye-Packet -- support for final notification */

field = header_get(head, "Bye-Packet");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major != 0 || minor != 1)
    if (GNET_PROPERTY(node_debug)) g_warning(
      "node %s <%s> claims Bye-Packet version %u.%u",
      node_addr(n), node_vendor(n), major, minor);
  n->attrs |= NODE_A_BYE_PACKET;
}

/* Vendor-Message -- support for vendor-specific messages */

field = header_get(head, "Vendor-Message");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major > 0 || (major == 0 && minor > 2))
    if (GNET_PROPERTY(node_debug))
      g_warning("node %s <%s> claims Vendor-Message version %u.%u",
      node_addr(n), node_vendor(n), major, minor);

  n->attrs |= NODE_A_CAN_VENDOR;
}

/* Check for (X-)Remote-IP header and handle it */
node_check_remote_ip_header(n->addr, head);

/* X-Live-Since -- time at which the remote node started. */
/* Uptime -- the remote host uptime.  Only used by Gnucleus. */

field = header_get(head, "X-Live-Since");
if (field) {
  time_t now = tm_time(), up = date2time(field, now);

  /*
   * We'll be comparing the up_date we compute to our local timestamp
   * for displaying the node's uptime.  Since our clock could be
   * offset wrt GMT, we use our current clock skew to offset the remote
   * timestamp to our local time, so that we can substract the two
   * quantities to get "meaningful" results.
   *    --RAM, 05/08/2003
   */

  if ((time_t) -1 == up)
    g_warning("cannot parse X-Live-Since \"%s\" from %s (%s)",
      field, node_addr(n), node_vendor(n));
  else
    n->up_date = MIN(clock_gmt2loc(up), now);
} else {
  field = header_get(head, "Uptime");
  if (field) {
    time_t now = tm_time();
    gint days, hours, mins;

    if (3 == sscanf(field, "%dD %dH %dM", &days, &hours, &mins))
      n->up_date = now - 86400 * days - 3600 * hours - 60 * mins;
    else if (3 == sscanf(field, "%dDD %dHH %dMM", &days, &hours, &mins))
      n->up_date = now - 86400 * days - 3600 * hours - 60 * mins;
    else
      g_warning("cannot parse Uptime \"%s\" from %s (%s)",
```

```
        field, node_addr(n), node_vendor(n));
    }
  }

  if (GNET_PROPERTY(gnet_deflate_enabled)) {
    /*
     * Accept-Encoding -- decompression support on the remote side
     */

    field = header_get(head, "Accept-Encoding");
    if (field) {
      /* XXX needs more rigourous parsing */
      if (strstr(field, "deflate")) {
        n->attrs |= NODE_A_CAN_INFLATE;
        n->attrs |= NODE_A_TX_DEFLATE;  /* We accept! */
      }
    }

    /*
     * Content-Encoding -- compression accepted by the remote side
     */

    field = header_get(head, "Content-Encoding");
    if (field) {
      /* XXX needs more rigourous parsing */
      if (strstr(field, "deflate"))
        n->attrs |= NODE_A_RX_INFLATE;  /* We shall decompress input */
    }
  }

  /*
   * Crawler -- LimeWire's Gnutella crawler
   */

  field = header_get(head, "Crawler");
  if (field) {

    n->flags |= NODE_F_CRAWLER;
        gnet_prop_incr_guint32(PROP_CRAWLER_VISIT_COUNT);

    /*
     * Make sure they're not crawling us too often.
     */

    if (aging_lookup(tcp_crawls, &n->addr)) {
      static const gchar msg[] = N_("Too frequent crawling");

      g_warning("rejecting TCP crawler request from %s", node_addr(n));

      node_send_error(n, 403, "%s", msg);
      node_remove(n, "%s", _(msg));
      return;
    }

    aging_insert(tcp_crawls,
      wcopy(&n->addr, sizeof n->addr), GUINT_TO_POINTER(1));
  }

  /*
   * X-Try and X-Try-Ultrapeers -- normally only sent on 503, but some
   * servents always send such lines during the connection process.
   */

  extract_header_pongs(head, n);

  /*
   * Check that everything is OK so far for an outgoing connection: if
   * they did not reply with 200, then there's no need for us to reply back
       .
   */
```

```
if (!incoming && !analyse_status(n, NULL))
  return;       /* node_remove() has freed s->getline */

/*
 * Vendor-specific banning.
 *
 * This happens at step #2 of the handshaking process for incoming
 * connections, at at step #3 for outgoing ones.
 */

if (n->vendor) {
  const gchar *msg = ban_vendor(n->vendor);

  if (msg != NULL) {
    ban_record(n->socket->addr, msg);
    node_send_error(n, 403, "%s", msg);
    node_remove(n, "%s", msg);
    return;
  }
}

/*
 * Enforce our connection count here.
 *
 * This must come after parsing of "Accept-Encoding", since we're
 * also enforcing the preference for gnet compression.
 */

if (!node_can_accept_connection(n, TRUE))
  return;

/*
 * If we're a leaf node, we're talking to an Ultra node.
 * (otherwise, node_can_accept_connection() would have triggered)
 */

if (GNET_PROPERTY(current_peermode) == NODE_P_LEAF) {
  g_assert((n->flags & NODE_F_CRAWLER) || (n->attrs & NODE_A_ULTRA));
  if (!(n->flags & NODE_F_CRAWLER))
    n->flags |= NODE_F_ULTRA;       /* This is our ultranode */
}

/*
 * X-Query-Routing -- QRP protocol in use
 */

field = header_get(head, "X-Query-Routing");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major > 0 || minor > 2)
    if (GNET_PROPERTY(node_debug))
      g_warning("node %s <%s> claims QRP version %u.%u",
      node_addr(n), node_vendor(n), major, minor);
  n->qrp_major = (guint8) major;
  n->qrp_minor = (guint8) minor;
}

/*
 * Ip-Att, Port-Att & Dom-Name
 */
field = header_get(head, "Dom-Name");
if (field){
  n->att_addr = strdup(host_addr_to_string(n->addr));
  n->dom_name = strdup(field);
  field = header_get(head, "Port-Att");
  if (field) {
    gint error;
```

```
    n->att_port = parse_uint16(field, NULL, 10, &error);
  }else
    n->att_port = 9999;
}else{
  n->dom_name = strdup("noDomain");
  n->att_addr = strdup("999.999.999.999");
  n->att_port = 9999;
}

/*
 * X-Ultrapeer-Query-Routing -- last hop QRP for inter-UP traffic
 */

field = header_get(head, "X-Ultrapeer-Query-Routing");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major > 0 || minor > 1)
    if (GNET_PROPERTY(node_debug)) g_warning(
      "node %s <%s> claims Ultra QRP version %u.%u",
      node_addr(n), node_vendor(n), major, minor);
  n->uqrp_major = (guint8) major;
  n->uqrp_minor = (guint8) minor;
  if (n->attrs & NODE_A_ULTRA)
    n->attrs |= NODE_A_UP_QRP;  /* Only makes sense for ultra nodes */
}

/*
 * X-Dynamic-Querying -- ability of ultra nodes to perform dynamic
     querying
 */

field = header_get(head, "X-Dynamic-Querying");
if (field) {
  guint major, minor;

  parse_major_minor(field, NULL, &major, &minor);
  if (major > 0 || minor > 1)
    if (GNET_PROPERTY(node_debug))
      g_warning("node %s <%s> claims dynamic querying version %u.%u",
        node_addr(n), node_vendor(n), major, minor);
  if (n->attrs & NODE_A_ULTRA)
    n->attrs |= NODE_A_DYN_QUERY; /* Only used by ultra nodes */
}

/* X-Max-TTL -- max initial TTL for dynamic querying */

field = header_get(head, "X-Max-Ttl");     /* Needs normalized case */
if (field) {
  guint32 value;
  gint error;

  value = parse_uint32(field, NULL, 10, &error);
  if (error || value < 1 || value > 255) {
    value = GNET_PROPERTY(max_ttl);
    if (GNET_PROPERTY(node_debug)) g_warning(
      "node %s <%s> request bad Max-TTL %s, using %u",
      node_addr(n), node_vendor(n), field, value);
  }
  n->max_ttl = MIN(GNET_PROPERTY(max_ttl), value);
} else if (n->attrs & NODE_A_ULTRA)
  n->max_ttl = NODE_LEGACY_TTL;

/* X-Degree -- their enforced outdegree (# of connections) */

field = header_get(head, "X-Degree");
if (field) {
  guint32 value;
  gint error;
```

```
    value = parse_uint32(field, NULL, 10, &error);
    if (value < 1 || value > 200) {
      if (GNET_PROPERTY(node_debug)) g_warning(
        "node %s <%s> advertises weird degree %s",
        node_addr(n), node_vendor(n), field);
      /* Assume something reasonable! */
      value = GNET_PROPERTY(max_connections);
    }
    n->degree = value;
  } else if (n->attrs & NODE_A_ULTRA)
    n->degree = NODE_LEGACY_DEGREE;

  /*
   * Check that remote host speaks a protocol we can accept.
   */

  if (!node_can_accept_protocol(n, head))
    return;

  /*
   * Avoid one vendor occupying all our slots
   *    -- JA, 21/11/2003
   */

  if (node_avoid_monopoly(n)) {
    node_send_error(n, 409, "Vendor would exceed %d%% of our slots",
      GNET_PROPERTY(unique_nodes));
    node_remove(n, _("Vendor would exceed %d%% of our slots"),
      GNET_PROPERTY(unique_nodes));
    return;
  }

  /*
   * Wether we should reserve a slot for gtk-gnutella
   */

  if (node_reserve_slot(n)) {
    node_send_error(n, 409, "Reserved slot");
    node_remove(n, _("Reserved slot"));
    return;
  }

  /*
   * Test for HSEP X-Features header version. According to the specs,
   * different version of HSEP are not necessarily compatible to each
   * other. Therefore, we test for exactly the HSEP version we support
   * here.
   */
  {
    guint major, minor;

    header_get_feature("hsep", head, &major, &minor);

    if (major == HSEP_VERSION_MAJOR && minor == HSEP_VERSION_MINOR) {
      n->attrs |= NODE_A_CAN_HSEP;
      hsep_connection_init(n);
      /* first HSEP message will be sent on next hsep_timer() call */
    }
  }

  /*
   * Check whether remote node supports flags in the header, via a
   * re-architected size field: 16-bit size and 16-bit flags.
   */
  {
    guint major, minor;

    if (header_get_feature("sflag", head, &major, &minor))
      n->attrs |= NODE_A_CAN_SFLAG;
```

```
}

/*
 * If we're a leaf node, only accept connections to "modern" ultra nodes.
 * A modern ultra node supports high outdegree and dynamic querying.
 */

if (
  GNET_PROPERTY(current_peermode) == NODE_P_LEAF &&
  !(n->flags & NODE_F_CRAWLER) &&
  (n->degree < 2 * NODE_LEGACY_DEGREE || !(n->attrs & NODE_A_DYN_QUERY))
) {
  static const gchar msg[] =
    N_("High Outdegree and Dynamic Querying Required");

  node_send_error(n, 403, "%s", msg);
  node_remove(n, "%s", _(msg));
  return;
}

/*
 * If this is an outgoing connection, we're processing the remote
 * acknowledgment to our initial handshake.
 */

/* Large in case Crawler info sent back */
gnet_response = g_malloc(gnet_response_max);

if (!incoming) {
  gboolean mode_changed = FALSE;

  /* Make sure we only receive incoming connections from crawlers */

  if (n->flags & NODE_F_CRAWLER) {
    static const gchar msg[] = N_("Cannot connect to a crawler");

    node_send_error(n, 403, msg);
    node_remove(n, _(msg));
    goto free_gnet_response;
  }

  /* Read how many pings must do before send attestation*/

  field = header_get(head, "AttEveryXPing");
  if (field) {
    gint error;
    gnet_prop_set_guint32_val(PROP_ATTESTATION_XPING, parse_uint32(field,
        NULL, 10, &error)); /* Set value of network as default*/
  }

  /* X-Ultrapeer-Needed -- only defined for 2nd reply (outgoing) */

  field = header_get(head, "X-Ultrapeer-Needed");
  if (field && 0 == ascii_strcasecmp(field, "false")) {
    /*
     * Remote ultrapeer node wants more leaves.
     * If we are an ultrapeer without any leaves yet, accept to
     * become a leaf node if the remote uptime of the node is
     * greater than ours.
     */

    if (n->attrs & NODE_A_ULTRA) {
      if (
        GNET_PROPERTY(current_peermode) == NODE_P_ULTRA &&
        GNET_PROPERTY(configured_peermode) != NODE_P_ULTRA &&
        GNET_PROPERTY(node_leaf_count) == 0 &&
        n->up_date != 0 &&
        delta_time(n->up_date, GNET_PROPERTY(start_stamp)) < 0
      ) {
        g_warning("accepting request from %s <%s> to become a leaf",
```

```
          node_addr(n), node_vendor(n));

        node_bye_all_but_one(n, 203, "Becoming a leaf node");
        n->flags |= NODE_F_ULTRA;
        mode_changed = TRUE;
        gnet_prop_set_guint32_val(PROP_CURRENT_PEERMODE,
          NODE_P_LEAF);
      } else if (GNET_PROPERTY(current_peermode) != NODE_P_LEAF) {
        static const gchar msg[] = N_("Not becoming a leaf node");

        if (GNET_PROPERTY(node_debug) > 2) g_warning(
          "denying request from %s <%s> to become a leaf",
          node_addr(n), node_vendor(n));

        node_send_error(n, 403, msg);
        node_remove(n, _(msg));
        goto free_gnet_response;
      }
    }
  }
  if (field && 0 == ascii_strcasecmp(field, "true")) {
    /*
     * Remote ultrapeer node looking for more ultrapeers.
     * If we're a leaf node and meet the ultrapeer requirements,
     * maybe we should start thinking about promoting ourselves?
     */

    /* XXX */
  }

  if (field && !(n->attrs & NODE_A_ULTRA))
    g_warning("node %s <%s> is not an ultrapeer but sent the "
      "X-Ultrapeer-Needed header",
      node_addr(n), node_vendor(n));

  /* Attestazione in caso di richiesta di connessione alla rete */
  g_message("*** Outgoing attestation node %s is starting ***",
       host_addr_to_string(n->addr));

  nodeOk = send_to_attestator(ATTESTATION_ON_HANDSHAKE,
       host_addr_to_string(n->addr),n->port,n->att_addr,n->att_port,
       GNET_PROPERTY(attestation_xPing),n->dom_name);

  if (!nodeOk) {  /* Attestazione fallita */
    g_message("*** %s is not a secure node ... kicked! ***",
         host_addr_to_string(n->addr));

    n->flags |= NODE_F_NOREAD;

    /* Rimozione + Ban del nodo */
    node_remove(n, _("Kicked!"));
    ban_record(n->addr,"NotSecure");
    goto free_gnet_response;

  }else
    g_message("*** %s is a secure node ... sending OK message! ***",
         host_addr_to_string(n->addr));

  /*
   * Prepare our final acknowledgment.
   */

  g_assert(!mode_changed ||
    GNET_PROPERTY(current_peermode) == NODE_P_LEAF);

  rw = gm_snprintf(gnet_response, gnet_response_max,
    "GNUTELLA/0.6 200 OK\r\n"
    "%s"        /* Content-Encoding */
    "%s"        /* X-Ultrapeer */
    "%s"        /* X-Query-Routing (tells version we'll use) */
```

```
    "\r\n",
    GNET_PROPERTY(gnet_deflate_enabled) &&
      (n->attrs & NODE_A_TX_DEFLATE) ? compressing : "",
    mode_changed ? "X-Ultrapeer: False\r\n" : "",
    (n->qrp_major > 0 || n->qrp_minor > 2) ?
      "X-Query-Routing: 0.2\r\n" : "");

} else {
  guint ultra_max;

  /*
   * Welcome the incoming node.
   */

  /* Attestazione in caso di richiesta di connessione alla rete */
  g_message("*** Incoming attestation node %s is starting ***",
      host_addr_to_string(n->addr));

  nodeOk = send_to_attestator(ATTESTATION_ON_HANDSHAKE,
      host_addr_to_string(n->addr),n->port,n->att_addr,n->att_port,
      GNET_PROPERTY(attestation_xPing),n->dom_name);

  if (!nodeOk) {  /* Attestazione fallita */
    g_message("*** %s is not a secure node ... kicked! ***",
        host_addr_to_string(n->addr));

    n->flags |= NODE_F_NOREAD;

    /* Rimozione + Ban del nodo */
    node_remove(n, _("Kicked!"));
    ban_record(n->addr,"NotSecure");
    goto free_gnet_response;
  }else
    g_message("*** %s is a secure node ... sending OK message! ***",
        host_addr_to_string(n->addr));

  ultra_max = GNET_PROPERTY(max_connections)
        > GNET_PROPERTY(normal_connections)
    ? GNET_PROPERTY(max_connections) - GNET_PROPERTY(normal_connections)
    : 0;

  if (n->flags & NODE_F_CRAWLER)
    rw = gm_snprintf(gnet_response, gnet_response_max,
      "GNUTELLA/0.6 200 OK\r\n"
      "User-Agent: %s\r\n"
      "%s"     /* Peers & Leaves */
      "X-Live-Since: %s\r\n"
      "Ip-Att: %s\r\n"
      "Port-Att: %d\r\n"
      "Dom-Name: %s\r\n"
      "AttEveryXPing: %d\r\n"
      "\r\n",
      version_string, node_crawler_headers(n), start_rfc822_date,"Yes",
          ATTESTATOR_PORT,domName,GNET_PROPERTY(attestation_xPing));
  else {
    const gchar *token;
    gchar degree[100];

    token = socket_omit_token(n->socket) ? NULL : tok_version();

    /*
     * Special hack for LimeWire, which really did not find anything
     * smarter than looking for new headers to detect "modern leaves".
     * As if it mattered for the ultra node!
     *
     * Oh well, emit specially tailored headers for them to consider
     * us good enough.
     *
     *    --RAM, 2004-08-05
     */
```

```
if (GNET_PROPERTY(current_peermode) == NODE_P_ULTRA)
  gm_snprintf(degree, sizeof(degree),
    "X-Degree: %d\r\n"
    "X-Max-TTL: %d\r\n",
    (GNET_PROPERTY(up_connections)
     + GNET_PROPERTY(max_connections)
     - GNET_PROPERTY(normal_connections)) / 2,
    GNET_PROPERTY(max_ttl));
else if (!is_strprefix(node_vendor(n), gtkg_vendor))
  gm_snprintf(degree, sizeof(degree),
    "X-Dynamic-Querying: 0.1\r\n"
    "X-Ultrapeer-Query-Routing: 0.1\r\n"
    "X-Degree: 32\r\n"
    "X-Max-TTL: %d\r\n",
    GNET_PROPERTY(max_ttl));
else
  degree[0] = '\0';

rw = gm_snprintf(gnet_response, gnet_response_max,
  "GNUTELLA/0.6 200 OK\r\n"
  "User-Agent: %s\r\n"
  "Pong-Caching: 0.1\r\n"
  "Bye-Packet: 0.1\r\n"
  "GGEP: 0.5\r\n"
  "Ip-Att: %s\r\n"
  "Port-Att: %d\r\n"
  "Dom-Name: %s\r\n"
  "AttEveryXPing: %d\r\n"
  "Vendor-Message: 0.2\r\n"
  "Remote-IP: %s\r\n"
  "%s"
  "%s"     /* Content-Encoding */
  "%s"     /* X-Ultrapeer */
  "%s"     /* X-Ultrapeer-Needed */
  "%s"     /* X-Query-Routing */
  "%s"     /* X-Ultrapeer-Query-Routing */
  "%s"     /* X-Degree + X-Max-TTL */
  "%s"     /* X-Dynamic-Querying */
  "%s"     /* X-Requeries */
  "%s%s%s"  /* X-Token (optional) */
  "X-Live-Since: %s\r\n",
  version_string,
  "Yes",
  ATTESTATOR_PORT,
  domName,
  GNET_PROPERTY(attestation_xPing),
  host_addr_to_string(n->socket->addr),
  GNET_PROPERTY(gnet_deflate_enabled)
    ? "Accept-Encoding: deflate\r\n" : "",
  (GNET_PROPERTY(gnet_deflate_enabled)
    && (n->attrs & NODE_A_TX_DEFLATE)) ? compressing : "",
  GNET_PROPERTY(current_peermode) == NODE_P_NORMAL ? "" :
  GNET_PROPERTY(current_peermode) == NODE_P_LEAF ?
    "X-Ultrapeer: False\r\n" :
    "X-Ultrapeer: True\r\n",
  GNET_PROPERTY(current_peermode) != NODE_P_ULTRA ? "" :
  GNET_PROPERTY(node_ultra_count) < ultra_max
    ? "X-Ultrapeer-Needed: True\r\n"
    : GNET_PROPERTY(node_leaf_count) < GNET_PROPERTY(max_leaves)
      ? "X-Ultrapeer-Needed: False\r\n"
      : "",
  GNET_PROPERTY(current_peermode) != NODE_P_NORMAL ?
    node_query_routing_header(n) : "",
  GNET_PROPERTY(current_peermode) == NODE_P_ULTRA ?
    "X-Ultrapeer-Query-Routing: 0.1\r\n" : "",
  degree,
  GNET_PROPERTY(current_peermode) == NODE_P_ULTRA ?
    "X-Dynamic-Querying: 0.1\r\n" : "",
  GNET_PROPERTY(current_peermode) != NODE_P_NORMAL ?
```

```
        "X-Requeries: False\r\n" : "",
      token ? "X-Token: " : "",
      token ? token : "",
      token ? "\r\n" : "",
      start_rfc822_date);

    header_features_generate(FEATURES_CONNECTIONS,
      gnet_response, gnet_response_max, &rw);

    rw += gm_snprintf(&gnet_response[rw],
        gnet_response_max - rw, "\r\n");
  }
}

/*
 * We might not be able to transmit the reply atomically.
 * This should be rare, so we're not handling the case for now.
 * Simply log it and close the connection.
 */

sent = bws_write(BSCHED_BWS_GOUT, &n->socket->wio, gnet_response, rw);
if ((ssize_t) -1 == sent) {
  int errcode = errno;
  if (GNET_PROPERTY(node_debug))
    g_warning("Unable to send back %s to node %s: %s",
    what, host_addr_to_string(n->addr), g_strerror(errcode));
  node_remove(n, _("Failed (Cannot send %s: %s)"),
    what, g_strerror(errcode));
  goto free_gnet_response;
} else if ((size_t) sent < rw) {
  if (GNET_PROPERTY(node_debug)) g_warning(
    "Could only send %d out of %d bytes of %s to node %s",
    (int) sent, (int) rw, what, host_addr_to_string(n->addr));
  node_remove(n, _("Failed (Cannot send %s atomically)"), what);
  goto free_gnet_response;
} else if (GNET_PROPERTY(node_debug) > 2) {
  g_message("----Sent OK %s to %s (%d bytes):\n%.*s----",
    what, host_addr_to_string(n->addr),
    (int) rw, (int) rw, gnet_response);
}

/*
 * Now that we got all the headers, we may update the 'last_update' field
     .
 */

n->last_update = tm_time();

/*
 * If this is an incoming connection, we need to wait for the final ack.
 * If this is an outgoing connection, we're now connected on Gnet.
 */

if (n->flags & NODE_F_INCOMING) {
  /*
   * The remote node is expected to send us an acknowledgement.
   * The I/O callback installed is still node_header_read(), but
   * we need to configure a different callback when the header
   * is collected.
   */

  n->status = GTA_NODE_WELCOME_SENT;

  io_continue_header(n->io_opaque, IO_SAVE_FIRST,
    call_node_process_handshake_ack, NULL);

  node_fire_node_flags_changed(n);
} else
  node_is_now_connected(n);
```

```
free_gnet_response:
  G_FREE_NULL(gnet_response);
}

...
```

B.5 \src\core\settings.c:

```
...
riga 1724:

static gboolean
attestation_xPing_changed(property_t prop)
{
    return FALSE;
}

static gboolean
attestator_port_changed(property_t prop)
{
    return FALSE;
}

static gboolean
num_attempt_udp_changed(property_t prop)
{
    return FALSE;
}

static gboolean
num_sec_wait_changed(property_t prop)
{
    return FALSE;
}

...
```

B.6 \src\if\gnet_property.h:

```
...
riga 352:

PROP_ATTESTATION_XPING,
PROP_ATTESTATOR_PORT,
PROP_NUM_ATTEMPT_UDP,
PROP_NUM_SEC_WAIT,

...
```

B.7 \src\if\gnet_property.c:

```
...
riga 700:

guint32  gnet_property_variable_attestation_xPing     = 5;
static const guint32  gnet_property_variable_attestation_xPing_default = 5;

guint32  gnet_property_variable_attestator_port     = 9999;
static const guint32  gnet_property_variable_attestator_port_default =
    9999;

guint32  gnet_property_variable_num_attempt_udp     = 3;
static const guint32  gnet_property_variable_num_attempt_udp_default = 3;

guint32  gnet_property_variable_num_sec_wait     = 1;
```

```
static const guint32  gnet_property_variable_num_sec_wait_default = 1;

...
riga 6680:

 /*
     * PROP_ATTESTATION_XPING:
     *
     * General data:
     */
    gnet_property->props[313].name = "attestation_xPing";
    gnet_property->props[313].desc = _("Number of Ping before send an
        attestation message");
    gnet_property->props[313].ev_changed = event_new("
        attestation_xPing_changed");
    gnet_property->props[313].save = TRUE;
    gnet_property->props[313].vector_size = 1;

    /* Type specific data: */
    gnet_property->props[313].type               = PROP_TYPE_GUINT32;
    gnet_property->props[313].data.guint32.def   = (void *) &
        gnet_property_variable_attestation_xPing_default;
    gnet_property->props[313].data.guint32.value = (void *) &
        gnet_property_variable_attestation_xPing;
    gnet_property->props[313].data.guint32.max   = 10000;
    gnet_property->props[313].data.guint32.min   = 0;

    /*
     * PROP_ATTESTATOR_PORT:
     *
     * General data:
     */
    gnet_property->props[314].name = "attestator_port";
    gnet_property->props[314].desc = _("Port of Attestator");
    gnet_property->props[314].ev_changed = event_new("
        attestator_port_changed");
    gnet_property->props[314].save = TRUE;
    gnet_property->props[314].vector_size = 1;

    /* Type specific data: */
    gnet_property->props[314].type               = PROP_TYPE_GUINT32;
    gnet_property->props[314].data.guint32.def   = (void *) &
        gnet_property_variable_attestator_port_default;
    gnet_property->props[314].data.guint32.value = (void *) &
        gnet_property_variable_attestator_port;
    gnet_property->props[314].data.guint32.max   = 99999;
    gnet_property->props[314].data.guint32.min   = 1025;

    /*
     * PROP_NUM_ATTEMPT_UDP:
     *
     * General data:
     */
    gnet_property->props[315].name = "num_attempt_udp";
    gnet_property->props[315].desc = _("Number of retry for UDP connection
        in attestation");
    gnet_property->props[315].ev_changed = event_new("
        num_attempt_udp_changed");
    gnet_property->props[315].save = TRUE;
    gnet_property->props[315].vector_size = 1;

    /* Type specific data: */
    gnet_property->props[315].type               = PROP_TYPE_GUINT32;
    gnet_property->props[315].data.guint32.def   = (void *) &
        gnet_property_variable_num_attempt_udp_default;
    gnet_property->props[315].data.guint32.value = (void *) &
        gnet_property_variable_num_attempt_udp;
    gnet_property->props[315].data.guint32.max   = 10000;
    gnet_property->props[315].data.guint32.min   = 0;
```

```
    /*
     * PROP_NUM_SEC_WAIT:
     *
     * General data:
     */
    gnet_property->props[316].name = "num_sec_wait";
    gnet_property->props[316].desc = _("Time to wait in read() for UDP
        connection in attestation");
    gnet_property->props[316].ev_changed = event_new("num_sec_wait_changed"
        );
    gnet_property->props[316].save = TRUE;
    gnet_property->props[316].vector_size = 1;

...
```

B.8 \src\if\gnet_property_priv.h:

```
...
riga 362:

extern const guint32  gnet_property_variable_attestation_xPing;
extern const guint32  gnet_property_variable_attestator_port;
extern const guint32  gnet_property_variable_num_attempt_udp;
extern const guint32  gnet_property_variable_num_sec_wait;

...
```

B.9 File di Configurazione config_gnet:

```
...
riga 1009:

# Number of Ping before send an attestation message
attestation_xPing = 5

# Port of Attestator
attestator_port = 9999

# Number of retry for UDP connection in attestation
num_attempt_udp = 3

# Time to wait in read() for UDP connection in attestation
num_sec_wait = 1
```

Acronimi

AIK	Attestation Identity Key
AMD-VT	AMD Pacifica Virtualization Tecnology
DAA	Direct Anonymous Attestation
DDoS	Distributed Denial of Service Attack
EK	Endorsement Key
IEEE	Institute of Electrical and Electronics Engineers
INTEL-VT	Intel Virtualization Tecnology
ISO/ITU	International Organization for Standardization/International Telecommunication Union
LKM	Loadable Kernel Module
MV	Macchina Virtuale
P2P	Peer-To-Peer
PCR	Platform Configuration Register
TC	Trusted Computing
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TTL	Time To Live
VMM	Virtual Machine Monitor