POLITECNICO DI MILANO





Sistemi Distribuiti

Secure p2p file sharing

Eros Lever - 766022
Alessandro Sisto - 770851

http://code.google.com/p/secure-p2p-polimi-distsys/

Descrizione progetto

Design and develop a peer-to-peer system structured as a **two-level node hierarchy**: supernodes and standard nodes; the former are fixed (both in number and address), while the latter dinamically connect and disconnect from the network. **Supernodes are all interconnected** with each other (choose your own topology). The community is private, so each node has access credentials (obtained out-of-band).
Each node executes the following operations:

- **join**: it connects to a known supernode, sending its credentials; the supernode checks these informations, possibly forwarding them to the other supernodes, until it is able to determine if the node can be authenticated; on a positive answer, the node will send its shared document list to the referring supernode
- **leave**: leave the network; closing existing connections and removing the set of shared files
- **publish**: publish a new file, informing the supernode
- **unpublish**: opposite as above
- **search**: the node sends the search request to its supernode, who will check in its internal list and possibly forward the query to the other supernodes, sending back the (possibly empty) list of nodes having the corresponding documents
- **fetch**: the node fetches the documents directly from the nodes who have it; if more than one node has the file, the node should be able to download different chunks from different peers.

It is required to introduce secure connections among all peers in the network (node to node, node to supernode, supernode to supernode), with the following objectives:

- avoid interception of any communication passing in the network
- verify the identity of nodes before fetching documents
- avoid malicious supernodes to appear as good ones to a connecting node

Build your own authentication and secure connection protocol (i.e., **do not use pre-defined ones like ssl**). May use the basic facilities provided into java.security and javax.crypto for creating simmetric and asymmetric keys, encrypting and signing data, and creating secure streams.

Assunzioni preliminari - 1

Supernodes are fixed (both in number and address):

Abbiamo assunto che tutti i nodi conoscano tre informazioni riguardo ai supernodi
<indirizzo ip, porta, public key>. Esse (per ogni supernodo del network) sono salvate all'interno del file supernodes.list.

Ogni nodo (sia supernodo che nodo semplice) conosce in qualunque momento le informazioni necessarie per connettersi ad un supernodo.

The community is private, so each node has access credentials (obtained out-of-band):

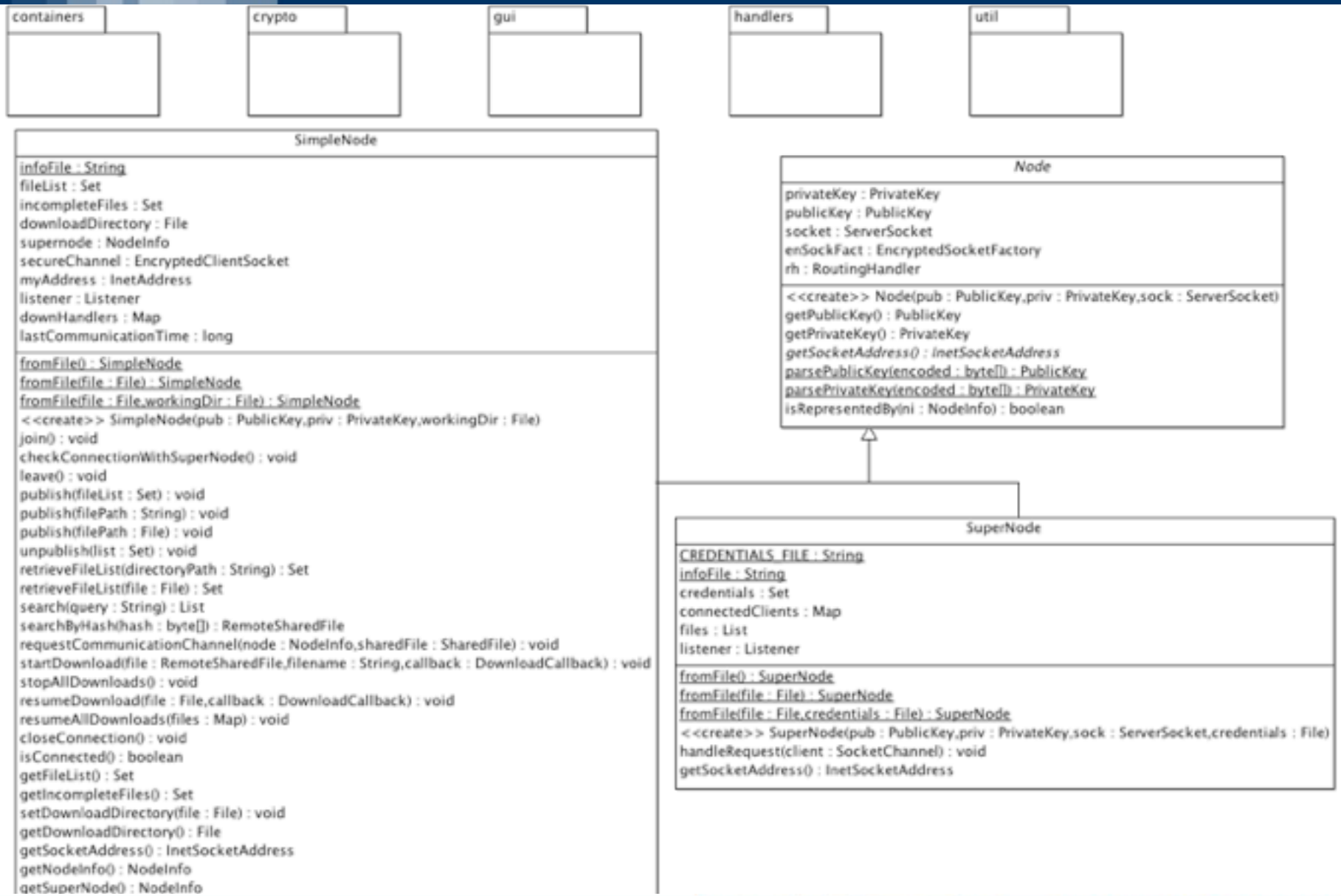
per l'autenticazione non usiamo la classica configurazione user, password. *("something i know")*, i supernodi posseggono una **lista di credenziali** (nel file credentials.list) che contiene le chiavi pubbliche dei nodi autorizzati*("something i have")*. Ogni nodo è quindi identificato da una chiave pubblica. Per i dettagli sul funzionamento dell'autenticazione e del protocollo di sicurezza rimandiamo alle slide successive.

Assunzioni preliminari - 2

Supernodes are all interconnected with each other (choose your own topology):

I supernodi sono tutti connessi tra loro (ogni nodo possiede la lista dei supernodi del network, accettabile data l'assunzione di numero e indirizzo fissato), inoltre ogni supernodo tiene traccia dei nodi collegati a se.
Le richieste all'interno del network che coinvolgono più supernodi (eg. search) vengono mandate in broadcast tra di essi.
Questo ovviamente pone un limite alla scalabilità del sistema (eccessivo numero di messaggi scambiati all'interno della rete per gestire le richieste)

Architettura: Overview



Gestione Persistenza

supernode.info: file che mantiene le informazioni locali del supernodo
<pub key: priv key: porta>

simplenode.info: file che mantiene le informazioni locali di un nodo semplice
<pub key: priv key>

credentials.list: lista dei nodi autorizzati alla comunicazione
lista di <pub key>

supernode.list: dati necessari per connettersi con i supernodi della rete
lista di <host: porta: pub key>

<nomefile>.tmp: file temporaneo usato per la gestione dei file incompleti. Esso contiene l'hash del file e un indice delle parti del file scaricate (usiamo un array di bit, dove l'indice indica il i-esimo blocco del file)

Gestione Concorrenza

Sia i nodi che i supernodi quando vengono istanziati lanciano un oggetto Listener che opera su un thread in parallelo e si mette in ascolto di richieste in ingresso.
Ogni volta che viene ricevuta una richiesta viene aperto un altro thread per gestirla.

I nodi gestiscono in parallelo anche i download in corso. (descrizione nel dettaglio in seguito)

La scelta da noi fatta è adeguata per il progetto in questione, in quanto le dimensioni dell'ambiente distribuito sono ridotte, quindi l'overhead per aprire e chiudere i thread ad ogni richiesta è accettabile.

Inoltre una soluzione di questo tipo non è sicura. Non c'e un limite massimo al numero di richieste gestibili ==> DoS

Protocollo di Sicurezza - 1

Algoritmi utilizzati:

Crittografia Asimmetrica: RSA, chiave: 1028 bit
- utilizzato per la fase di handshake

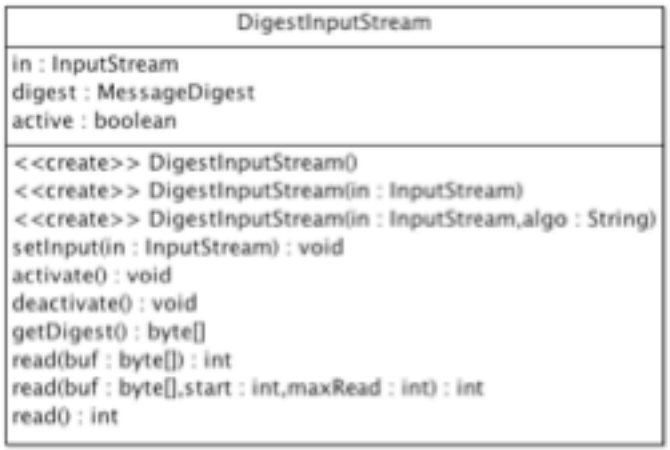
Crittografia Simmetrica: AES, chiave: 128 bit
- usato per le comunicazioni successive

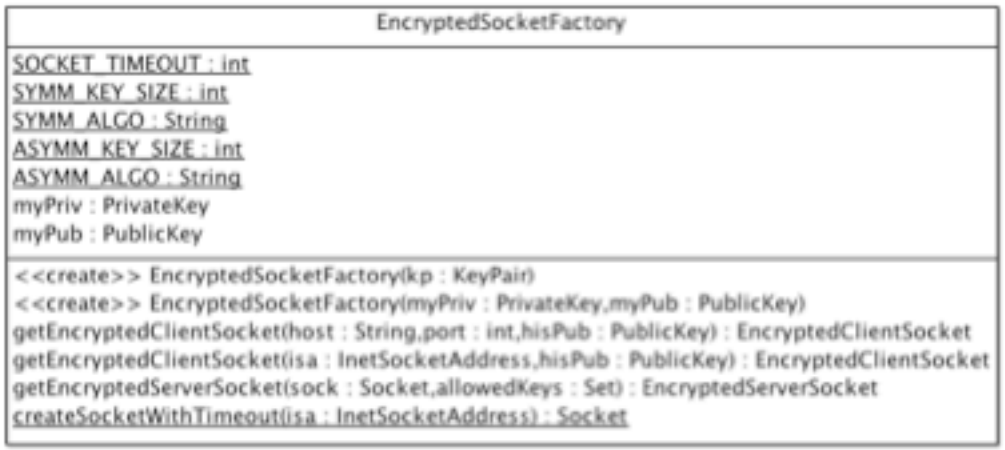
Hashing: SHA-1

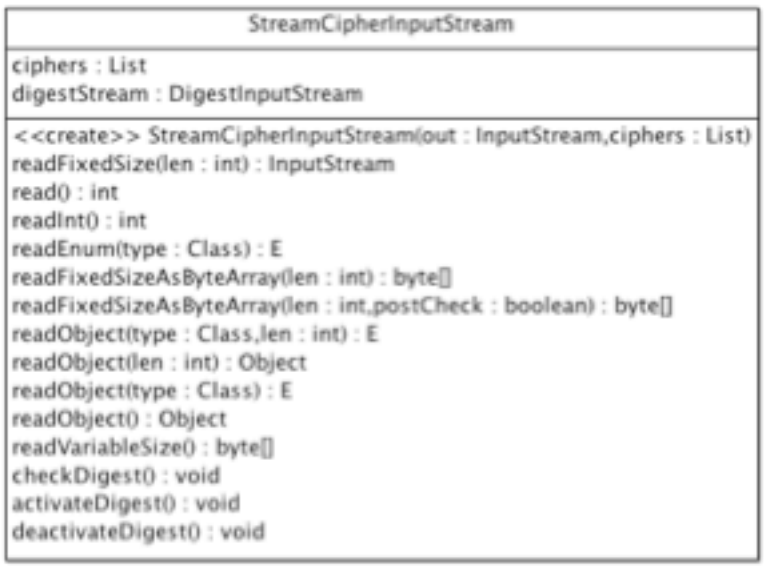
Per gestire in maniera il più possibile trasparente gli aspetti di sicurezza, abbiamo utilizzato una classe ad Hoc **EncryptedSocket** che oltre ad offrire le classiche primitive di comunicazione, garantisce che gli stream in input e output siano cifrati e decifrati automaticamente.
Per garantire l'integrità dei messaggi scambiati inoltre ogni comunicazione nel network è accompagnata da un digest, questa funzionalità è dal encryptedSocket (il digest viene creato in automatico, ma il controllo deve essere chiamato esplicitamente: primitive sendDigest() e checkDigest())

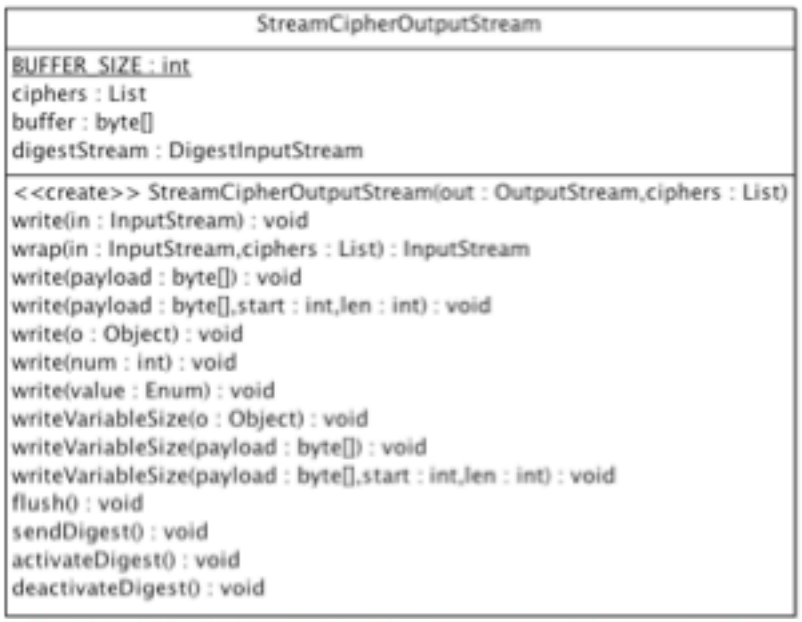
In caso di problemi di sicurezza viene lanciata una **GeneralSecurityException**

Architettura: Crypto



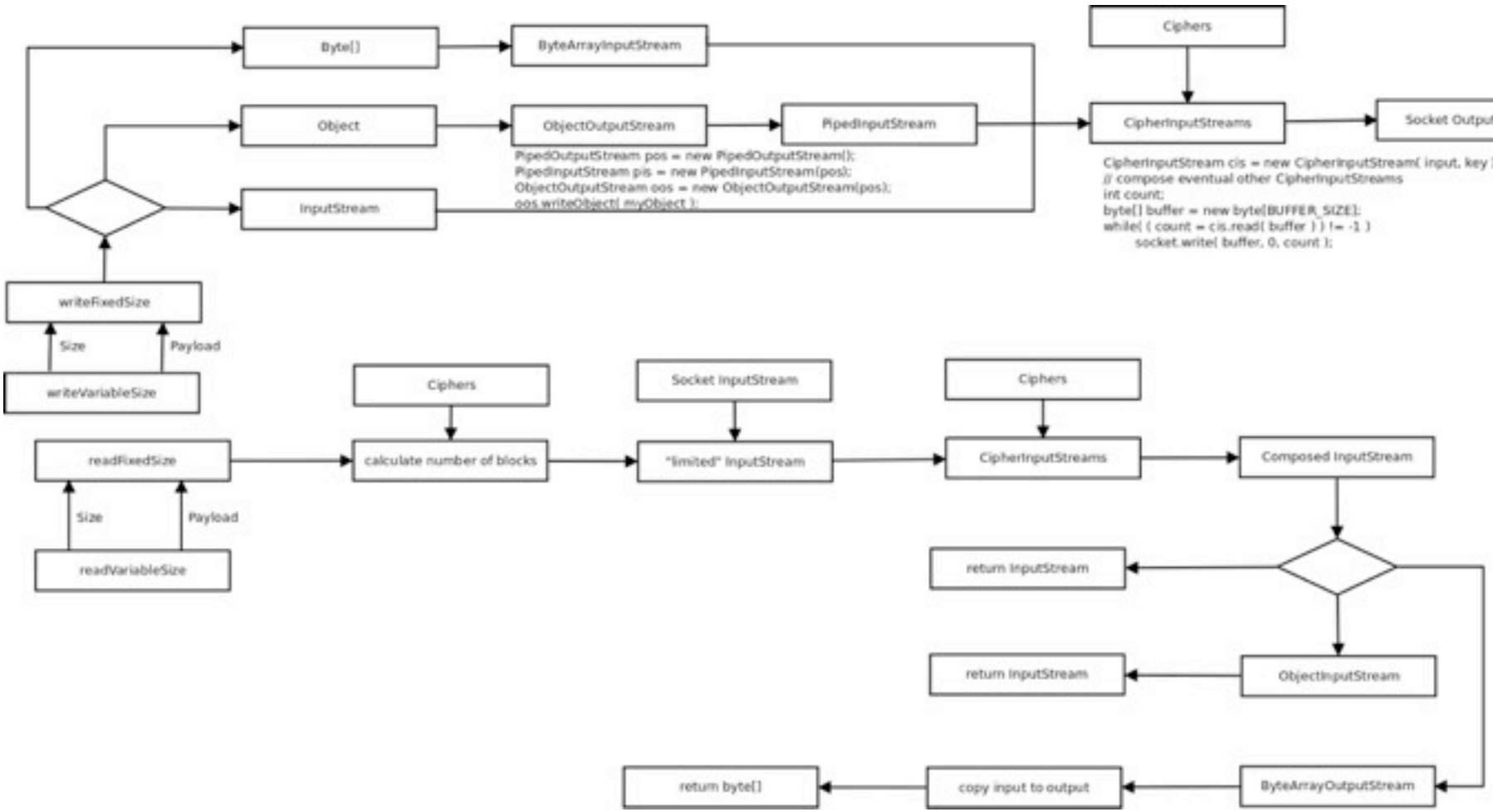






Protocollo di Sicurezza - 2

Encrypted Socket Layer: funzionamento



Protocollo di Sicurezza - 3

Handshake

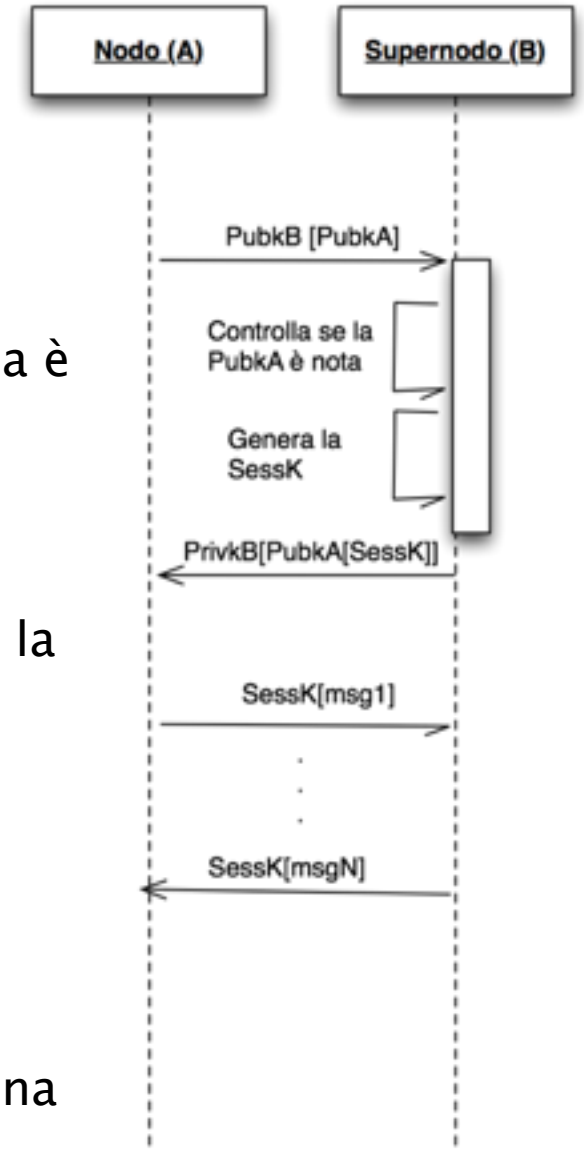
1) A cifra la propria publicKey con la chiave pubblica di B e la spedisce

2) B decifra il messaggio con la propria chiave privata e controlla se la chiave pubblica ricevuta è presente tra quelle autorizzate alla comunicazione
(assunzione credenziali out-of-band)

3) B genera una chiave segreta per la sessione, la cifra prima con la propria privata,poi con la pubblica A e la invia.

4) A decifra prima con la sua chiave privata poi con la chiave pubblica di B.

- le comunicazioni successive avvengono con crittografia simmetrica, usando il segreto appena condiviso



Protocollo di Comunicazione

La comunicazione tra nodi diversi avviene attraverso lo scambio di messaggi. Le possibili richieste e risposte sono qui elencate.

Primitive di comunicazione:

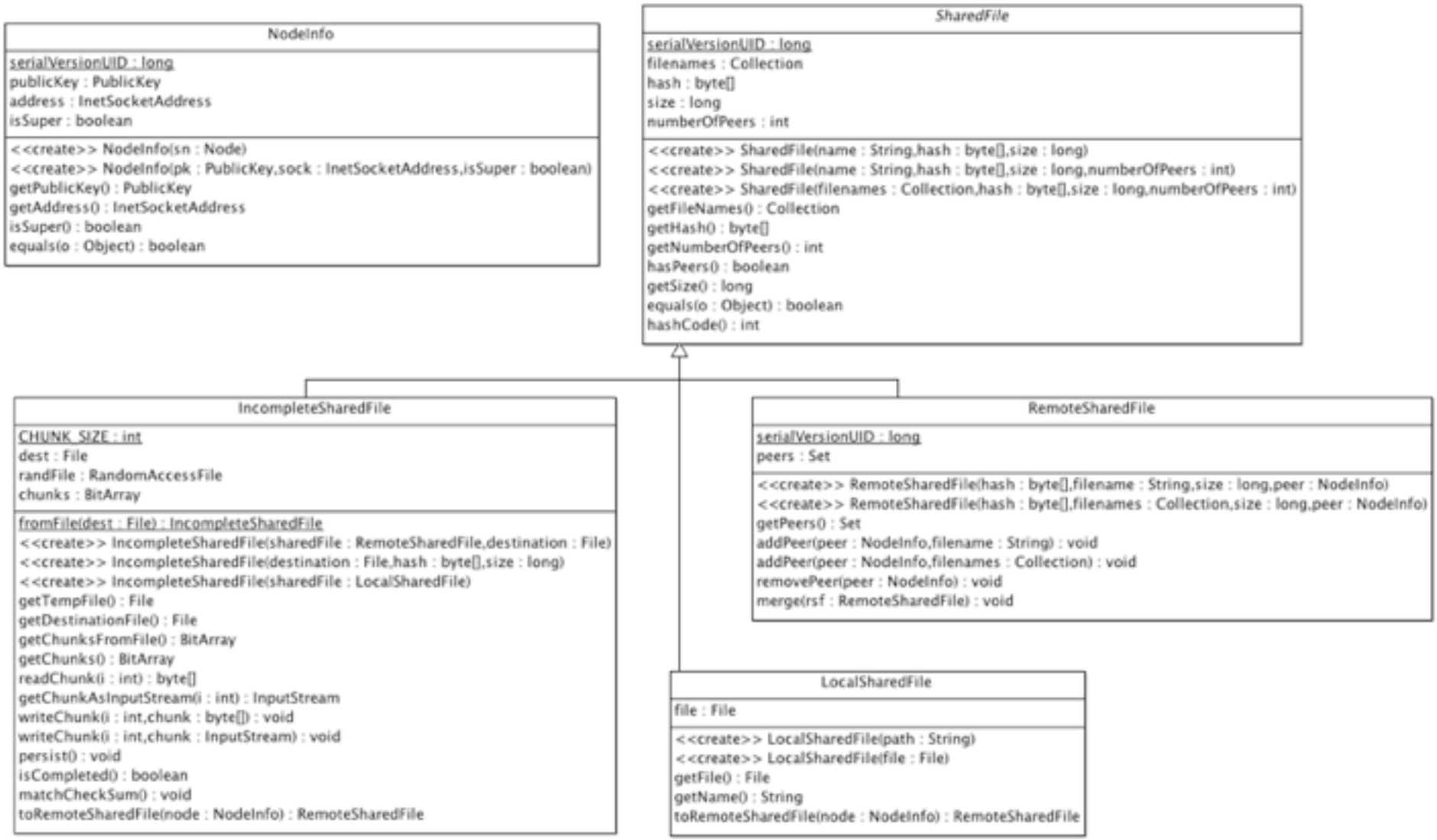
Richieste:

LOGIN,
PUBLISH,
UNPUBLISH,
SEARCH,
LEAVE,
CLOSE_CONN,
FORWARD_SEARCH,
LIST_AVAILABLE_CHUNKS,
FETCH_CHUNK,
SEARCH_BY_HASH,
FORWARD_SEARCH_BY_HASH,
ADD_TRUSTED_DOWNLOAD,
OPEN_COMMUNICATION,
PING

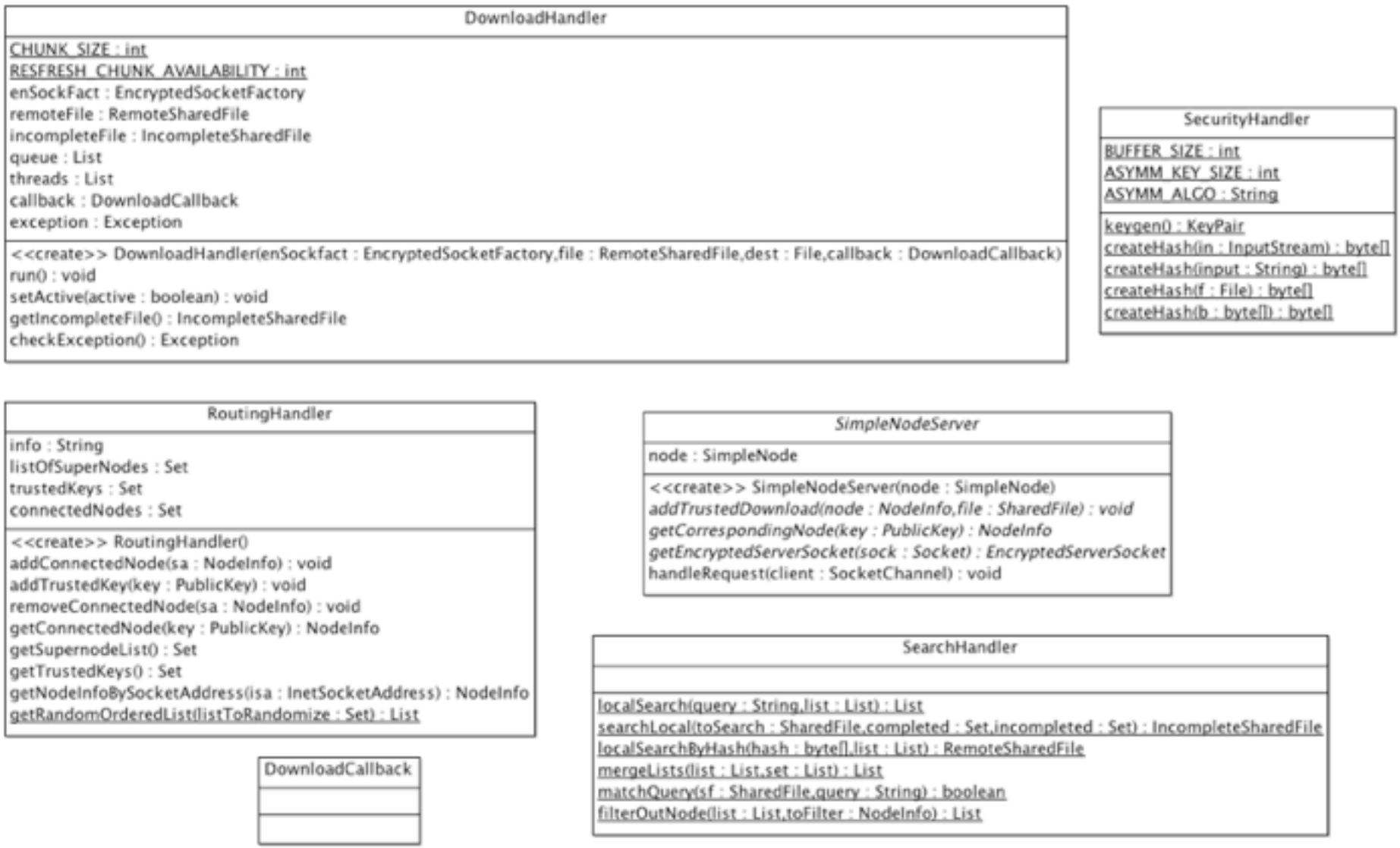
Risposte:

OK,
FAIL,
ALREADY_CONNECTED,
NOT_CONNECTED,
PONG

Architettura: Containers



Architettura: Handlers



Protocollo di comunicazione - Join

L'autenticazione di un nodo con il supernodo avviene in maniera intrinseca per come abbiamo concepito il protocollo di sicurezza.

1) Il nodo invia al supernodo la propria pub key
2) il supernodo controlla se la pub key del nodo è nella lista di quelli autorizzati, solo in quel caso continua la comunicazione

Primitiva di richiesta: Join

Payload atteso: la porta del nodo su cui si è messo in listen

Risposte: OK, in caso di successo, nessuna risposta in caso di insuccesso [viene considerato un timeout]

Il nodo quando prova a connettersi al network contatta a caso uno dei supernodi fino all'ottenimento di una risposta OK, le successive comunicazioni avverranno tutte con quel supernodo.

Protocollo di comunicazione - Publish

- Ogni supernodo tiene traccia localmente di tutti i file condivisi dai nodi semplici a lui connessi, siano essi completi o meno.

Due primitive per gestire la publish:

PUBLISH: il nodo invia al supernodo di riferimento una lista di file da condividere (eventualmente una lista con un solo elemento)
payload atteso: lista di <? extends SharedFile>
Risposte: OK, in caso di successo, nessuna risposta in caso di insuccesso [viene considerato un timeout]

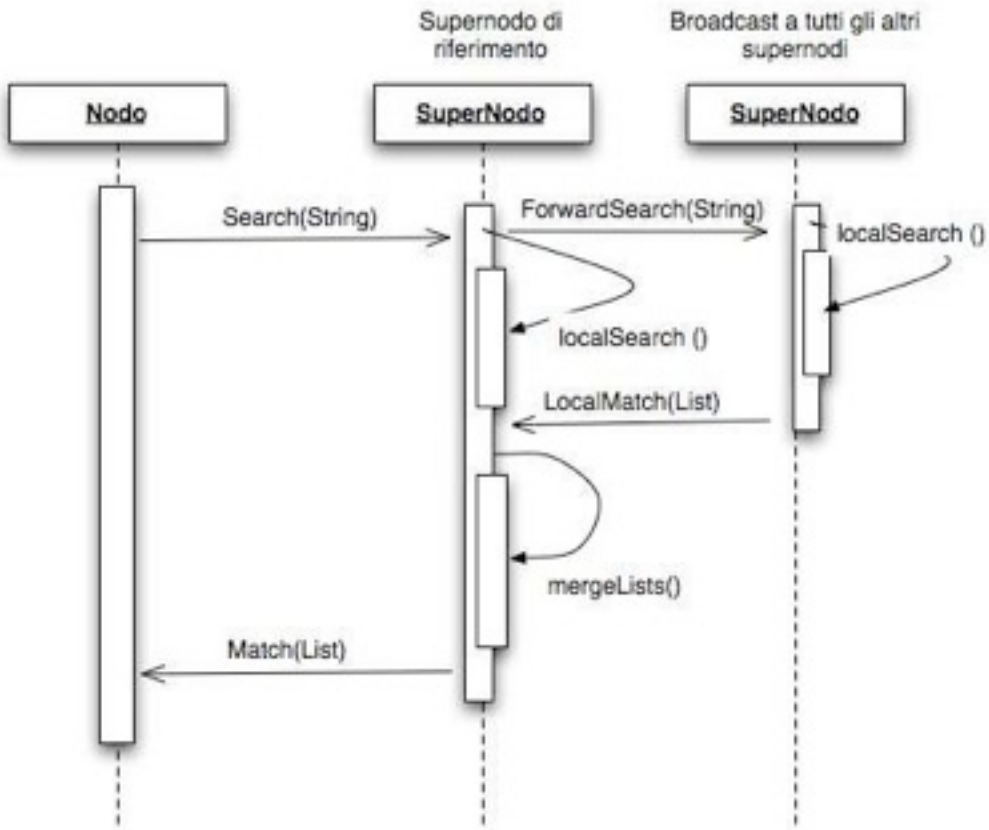
UNPUBLISH: il nodo invia al supernodo una lista di file (anche in questo caso il caso limite è una lista con un unico elemento) che desidera rimuovere dalla condivisione
payload atteso: lista di <? extends SharedFile>
Risposte: OK, in caso di successo, nessuna risposta in caso di insuccesso [viene considerato un timeout]

NOTA: i file nei supernodi sono raggruppati per hash!

Protocollo di comunicazione - Search 1

Ogni nodo può effettuare una ricerca per trovare i file condivisi da altri nodi nel network.

La ricerca avviene in 4 passaggi:
1) il nodo invia la richiesta al supernodo(B) di riferimento
2) B inoltra la richiesta in broadcast agli altri supernodi e cerca nella sua lista locale
3) una volta ricevuta la richiesta i supernodi preparano cercano nelle proprie liste locali e restituiscono a B il risultato
4) B unisce i risultati locali ottenuti e invia al nodo richiedente la risposta alla sua query



Protocollo di comunicazione - Search 2

SEARCH: richiesta effettuata nodo -> supernodo
payload atteso: Stringa (query)
Risposte: in caso di risposta OK, viene inviata una lista di RemoteSharedFile

FORWARD_SEARCH: primitiva per inoltrare la richiesta agli altri supernodi
payload atteso: Stringa (query)
Risposte: in caso di risposta OK, viene inviata una lista di RemoteSharedFile

Il matching della query è gestito a livello di parole. Se almeno una parola della query è uguale ad una di quelle presenti nel nome del file, allora il file viene restituito all'utente. Per semplicità non abbiamo considerato possibili misure di similarità tra le parole.

Protocollo di comunicazione - Search 3

Caso limite: dato che la ricerca viene gestita a livello di nome, può succedere, nel caso di file uguali (lo stesso hash) con nomi diversi, che vengano restituite meno fonti di quelle che ci sono realmente nella rete.

Per ovviare a ciò abbiamo adottato 2 strategie:

1) Localmente (nelle liste dei supernodi) raggruppiamo i file per valore di Hash

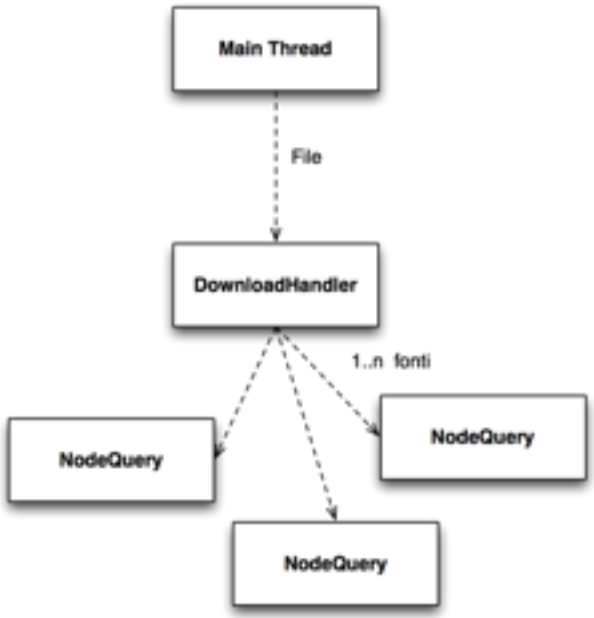
2) Nel momento che un utente chiede il download di un file viene lanciata una primitiva di ricerca a livello di Hash, in modo da ottenere con certezza tutte le fonti per quel determinato file.

Questa seconda operazione funziona in maniera del tutto analoga a quanto descritto per la search, usando le primitive **SEARCH_BY_HASH** e **FORWARD_SEARCH_BY_HASH**

Protocollo di comunicazione - Download 1

Una volta svolta la ricerca, il nodo può scaricare il file dagli altri nodi che lo condividono.

L'operazione è svolta su diversi threads:
ad ogni richiesta viene creato un oggetto DownloadHandler che aprirà un thread apposta per il download del file specificato.
A sua volta aprirà un thread (NodeQuerySender) per ognuna delle fonti disponibili per quel file, che si occuperà della comunicazione nodo->nodo



Per permettere il download contemporaneo da più fonti, abbiamo deciso di suddividere ogni file in blocchi di dimensione fissata [256Kb]

Protocollo di comunicazione - Download 2

La prima operazione necessaria per effettuerare il download è l'autenticazione nodo --> nodo.

OPEN_COMUNICATION: ogni thread NodeQuerySender (A) manda una richiesta al supernodo di riferimento chiedendo di aprire la connessione con un altro nodo (B) appositamente per il passaggio di uno specifico file X
payload atteso: NodeInfo (B), SharedFile (X)
Risposte: OK, se il supernodo ha svolto con successo una richiesta ADD_TRUSTED DOWNLOAD

ADD_TRUSTED DOWNLOAD: il supernodo contatta il nodo (B) della rete e gli comunica di accettare le richieste del nodo (A) per il file (X)
payload atteso: NodeInfo (A), SharedFile (X)
Risposte: OK, in caso di successo

In pratica il supernodo (componente trusted) comunica la publicKey (contenuta nel NodeInfo) del richiedente al nodo che possiede il file. A questo punto saran i due nodi con le publicKey corrispondenti a svolgere l'handshake e ad instaurare una comunicazione sicura.

Protocollo di comunicazione - Download 3

Una volta aperta la comunicazione tra due nodi i messaggi seguenti vengono scambiati direttamente nodo->nodo senza interventi dei supernodi.

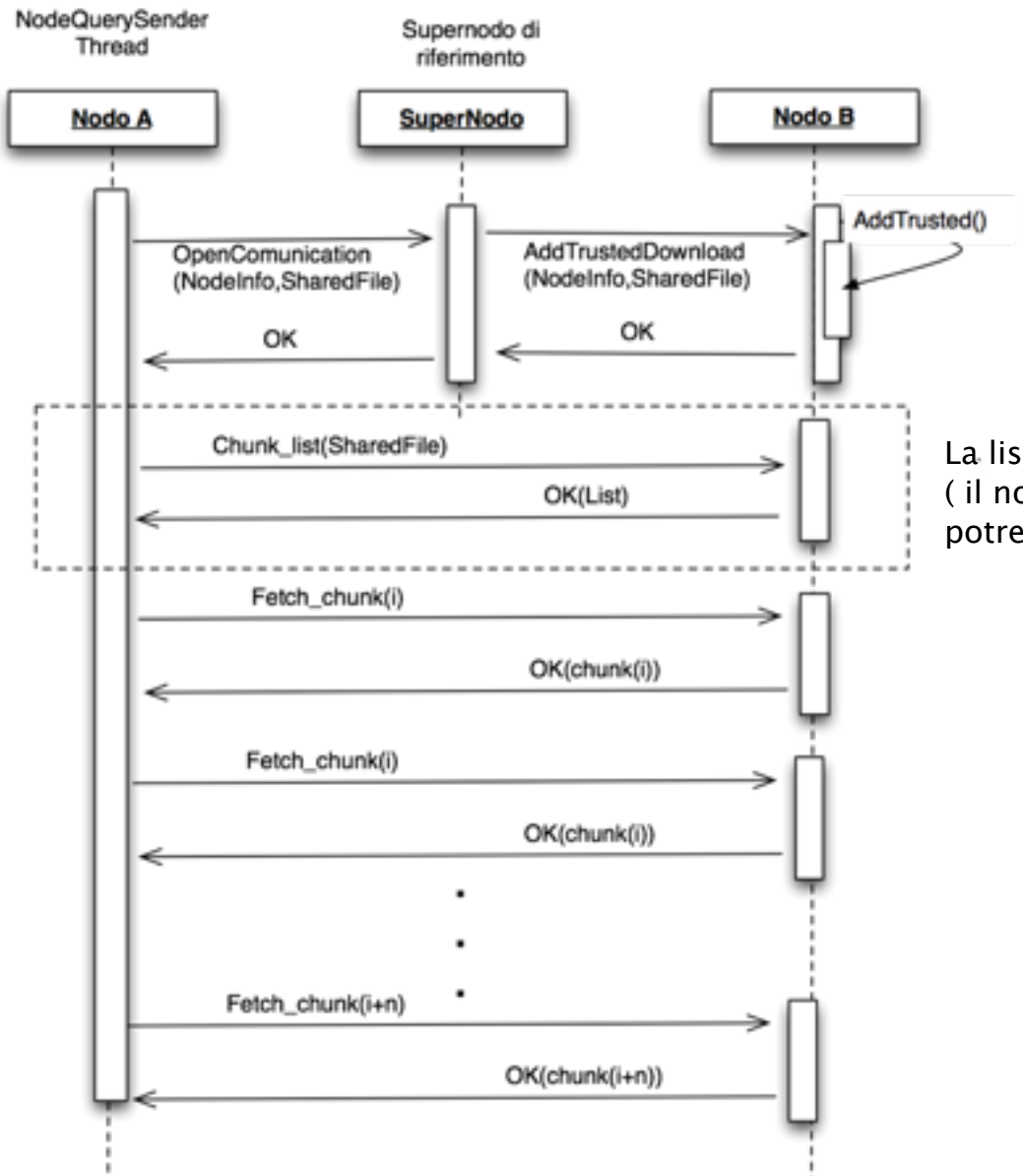
LIST_AVAILABLE_CHUNKS: il nodo A chiede al nodo B quali parti del file ha disponibili (necessario per poter garantire condivisioni di file parziali)
payload atteso: RemoteSharedFile
Risposta: BitArray (lista dei blocchi disponibili)

FETCH_CHUNK: il Nodo A richiede al nodo B l'i-esimo blocco del file
payload atteso: indice del blocco, RemoteSharedFile
Risposta: in caso di OK, il blocco del file

Ogni volta che un blocco viene scaricato con successo, si aggiorna:
- il file destinazione (viene scritto il blocco nella posizione corretta)
- il file temporaneo che tiene traccia dei blocchi già scaricati

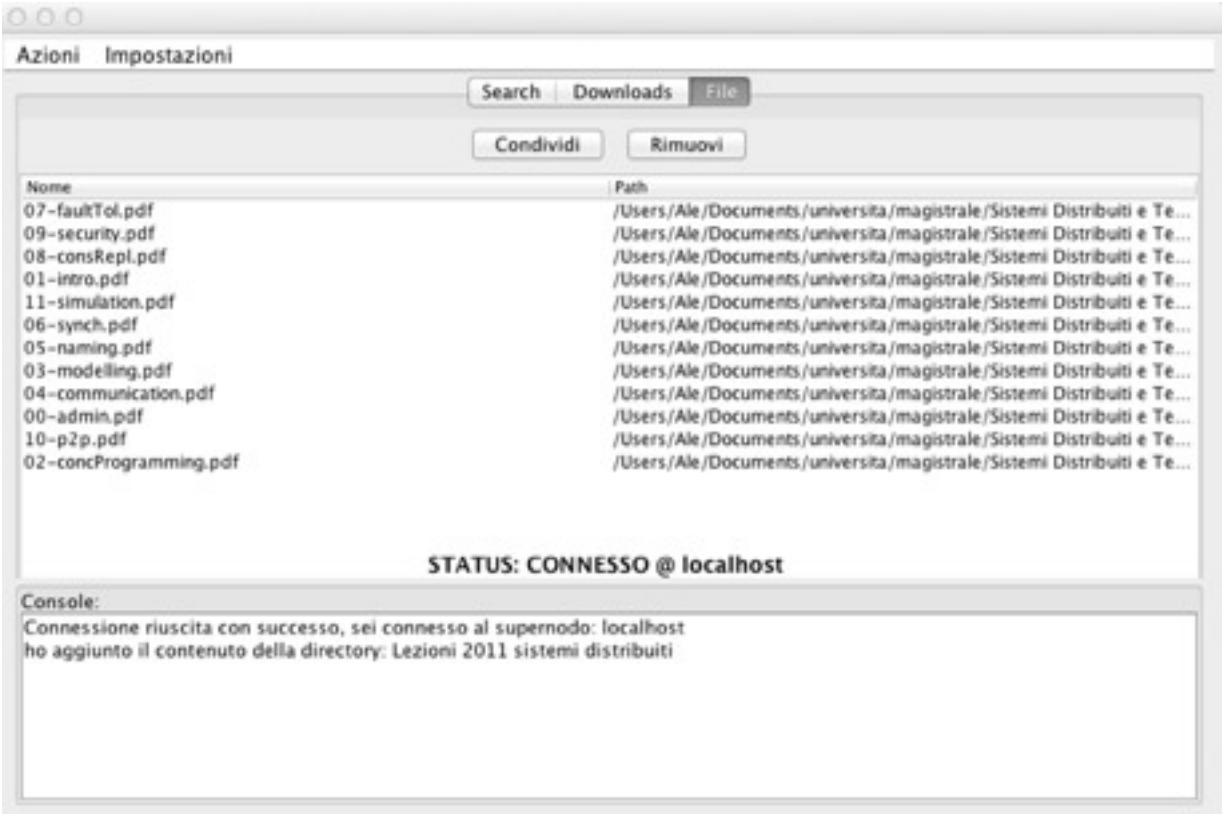
Dato che vengono scaricati più blocchi del file in parallelo da diverse fonti, usiamo una Lista sincronizzata per tracciare i download dei blocchi in corso, per evitare di richiedere lo stesso blocco più volte.

Protocollo di comunicazione - Download 4



La lista viene richiesta ogni N secondi (il nodo b se non ha il file completo potrebbe aver ottenuto nuove parti)

Graphical User Interface



Per facilitare il testing e la presentazione abbiamo inoltre realizzato una piccola interfaccia grafica per i nodi semplici