

Università degli Studi di Napoli “Parthenope” www.uniparthenope.it

Dipartimento di Scienze e Tecnologie

Centro Direzionale Isola C4

80143 Napoli - Italy

**Reti di Calcolatori e Laboratorio di Reti di Calcolatori**

A.A. 2018-2019

**Progetto** **BlockExplorer**

**Studenti:**

|  |  |  |
| --- | --- | --- |
| COGNOME | NOME | MATRICOLA |
| Bevilacqua | Vincenzo | 0124001490 |
| Di Marino | Antonio | 0124001344 |

**Docente:** Alessio Ferone

INDICE

[Descrizione del progetto 3](#_Toc3843705)

[Descrizione e schemi dell’architettura 4](#_Toc3843706)

[Schema della rete 4](#_Toc3843707)

[Descrizione della rete 4](#_Toc3843708)

[Protocollo di livello trasporto 5](#_Toc3843709)

[Descrizione e schemi del protocollo applicazione 6](#_Toc3843710)

[Schema generale 6](#_Toc3843711)

[Descrizione schema generale 7](#_Toc3843712)

[Schema Generazione Blocchi 8](#_Toc3843713)

[Descriziome Generazione Blocchi 8](#_Toc3843714)

[Schema Trasferimento Blocchi 10](#_Toc3843715)

[Descrizione Trasferimento Blocchi 10](#_Toc3843716)

# **Descrizione del progetto**

Il progetto si pone l’obiettivo di analizzare una *blockchain*, essa è una sequenza di blocchi in cui ogni blocco contiene una transazione.  
La rete si compone di tre entità fondamentali, un *NodoN*, un *BlockServer* e un *BlockClient*.

Un *NodoN* è un ` entità che simula una blockchain e ne salva la copia su di un file, il *BlockServer*  ha il compito di connettersi con il *NodoN* per ricevere una copia della blockchain e fornire informazioni su di essa ad uno o più *BlockClient* , il *BlockClient* a sua volta si connette al *BlockServer* per poter analizzare le transazioni contenute nella blockchain.

# **Descrizione e schemi dell’architettura**

## Schema della rete

## Descrizione della rete

La rete si compone di un *NodoN* che genera blocchi casuali in una blockchain per poi salvarli in un suo file locale “blocchi\_nodon.txt”.  
In una visione più ampia della rete il *NodoN* si collega ad una rete di peer dove la funzione principale è quella di scambiarsi i blocchi di una blockchain, in questo progetto invece si presuppone che tutto ciò già avvenga, per cui il trasferimento dei blocchi viene simulato generando i blocchi casualmente, quindi il *NodoN* verrà utilizzato solo come **server**. La finalità di questo progetto consiste nell’analisi dei blocchi e per tal fine si necessita di un solo *NodoN*.

Il *BlockServer* è l’entità che ha una doppia funzionalità, esso si occupa di reperire i blocchi connettendosi al *NodoN* e di salvarli in un proprio file locale “blocchi\_blockserver.txt”, inoltre si occupa di fornire ad n *BlockClient* dei servizi di analisi della blockchain.

Il *BlockClient* rappresenta un generico client che vuole reperire informazioni riguardanti i blocchi della blockchain, esso si connette al *BlockServer* per interrogare la blockchain. In particolare, i *BlockClient* possono richiedere di:

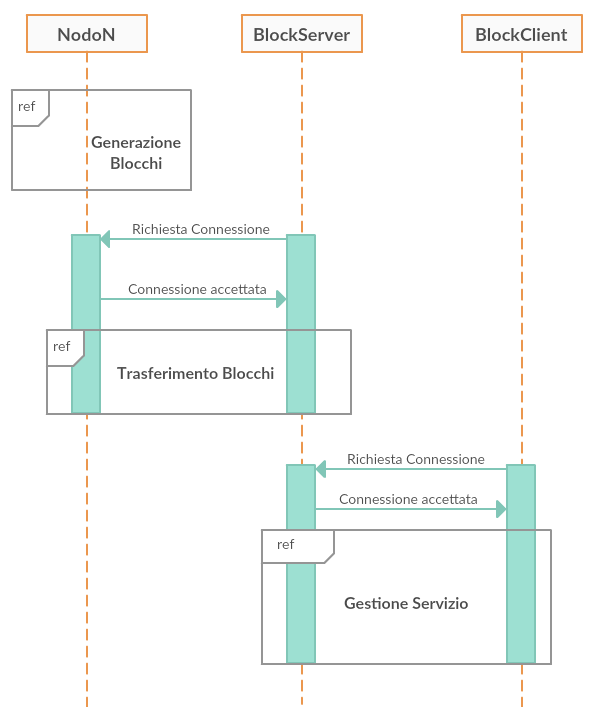
* Visualizzare le ultime n transazioni.
* Visualizzare i dati di una specifica transazione
* Visualizzare la somma dei valori di tutta la blockchain
* Visualizzare il numero di transazioni in cui è coinvolto un indirizzo specifico.
* Visualizzare i dati di tutte le transazioni in cui è coinvolto un indirizzo specifico.
* Visualizzare il bilancio delle transazioni in cui è coinvolto un indirizzo specifico.

## Protocollo di livello trasporto

Nel livello trasporto del TCP/IP si è scelto di utilizzare il protocollo **TCP** invece dell’UDP, dato che bisogna assicurare che i blocchi vengano correttamente trasferiti dal NodoN al BlockServer con un preciso ordine, bisogna considerare anche l’importanza delle transazioni che nella realtà dei fatti contengono denaro ed esse non possono essere perse, e infine dato che il TCP mantiene lo stato della connessione si possono monitorare e gestire ipotetiche cadute di connessione.

# **Descrizione e schemi del protocollo applicazione**

## Schema generale

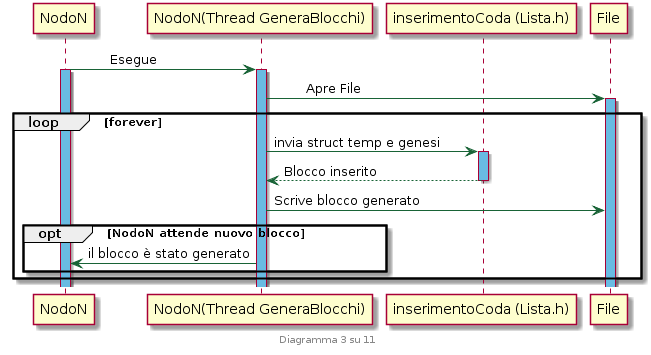


### Descrizione schema generale

Lo schema raffigurato a [Pagina 6] rappresenta per sommi capi le macro-azioni che ogni entità effettua all’interno della rete, verranno approfondite in seguito nel dettaglio. In questo schema vengono descritte le operazioni nell’ordine più consono rispetto alle richieste del progetto, ma la reale implementazione tiene conto delle possibili anomalie che possono far si che alcune azioni non vengano eseguite nell’ordine prestabilito a [pagina 6] .

Da come si evince dalla figura a [pagina 6] la prima azione viene effettuata dal *NodoN* , esso ha il compito generare una blockchain, in seguito il *NodoN* si mette in attesa di connessione di un *BlockServer*, che quando si connette richiede il trasferimento dei blocchi a partire da un blocco specifico. Dopo che il BlockServer ha costruito una propria copia della blockchain, è pronto per accettare connessioni con i *BlockClient* che richiederanno dei servizi per analizzare le transazioni.

## Schema Generazione Blocchi



### Descriziome Generazione Blocchi

Prima di analizzare la generazione di blocchi, è di fondamentale importanza introdurre la struttura dei blocchi e il loro contenuto:

1. **struct** Blocco
2. {
3. **int** n;
4. **int** tempo;
5. **struct** Transazione ts;
6. **struct** Blocco \*next;
7. } ;

n: Rappresenta l’identificativo progressivo dei blocchi, che in questo progetto parte da 0 con il blocco **genesi**.

tempo: Rappresenta un tempo randomico compreso tra 5 e 15 secondi che il NodoN deve attendere prima di inserire un nuovo blocco nella blockchain.

next: Rappresenta un puntatore al successivo blocco, considerando che la blockchain viene rappresentata come una lista.

La transazione ts è a sua volta una struttura così definita:

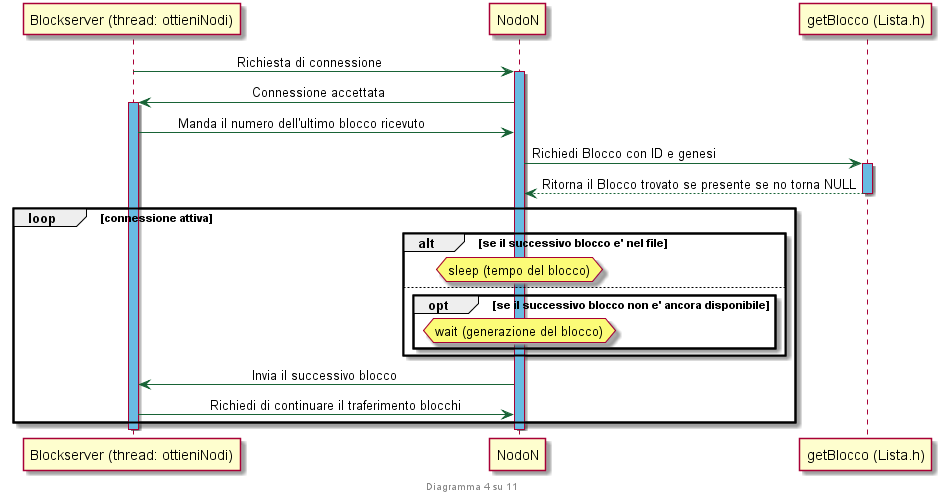
1. **struct** Transazione
2. {
3. **char** ipMittente[16];
4. **int** portaMittente;
5. **int** credito;
6. **char** ipDestinatario[16];
7. **int** portaDestinatario;
8. **int** numRandom;
9. };

Stabilite le strutture che vengono utilizzate, il *NodoN* appea va in esecuzione , lancia il thread produci che si occupa di aprire il file, per un tempo indefinito si occuperà di generare dei blocchi che verranno inseriti in coda alla blockchain e sul file. Una nota particolare va fatta sulla scrittura nel file, poiché ci si avvale di una struttura temporanea per evitare di scrivere dati superflui come il campo “next” della struttura blocco:

1. **struct** temp
2. {
3. **int** n;
4. **int** tempo;
5. **struct** Transazione ts;
6. };

Infine, il thread ha il compito di svegliare suo padre *NodoN* nel caso in cui quest’ultimo sia in attesa di un nodo che il thread ha appena generato.

## Schema Trasferimento Blocchi



## Descrizione Trasferimento Blocchi

Ci si riferisce al thread *ottieniNodi* con il nome del thread master che lo compone: *BlockServer*.

Il *BlockServer* per ottenere i nodi che utilizzerà per le analisi di un generico *BlockClient* si connetterà al *NodoN*, successivamente gli invia l’identificativo dell’ultimo blocco in suo possesso, per far sì che il *NodoN* gli invii i blocchi a partire dal successivo blocco posseduto da *BlockServer*. Dopo la richiesta entrambe le entità entreranno in un loop che potrà essere interrotto solo se una delle due parti si disconnette.

Il *NodoN* controlla l’identificativo che ha ricevuto da *BlockServer*, se questo identificativo corrisponde ad un blocco presente nel file, *NodoN* preleverà questo blocco e attenderà il tempo randomico del blocco, poiché considerando che era stato generato in una precedente esecuzione del *NodoN*, dato che in questa nuova esecuzione non ha atteso il tempo randomico, questo comportamento viene simulato prima di inviarlo al *Blockserver*.

Nel caso in cui sia *NodoN* che *BlockServer* hanno la stessa copia della blockchain, allora il *NodoN* dovra attendere che il thread **produci** generi un nuovo blocco da inviare a *BlockServer*.

Mentre nel caso in cui il blocco viene generato al momento allora non ci sarà nessuna attesa da parte del *NodoN*, poiché si fa fede all’attesa effettuata durante la generazione del blocco.

## Schema Analisi BlockChain

# Descrizione Analisi BlockChain

Le implementazioni delle scelte che si evincono dallo schema a [pagina 14] verranno approfondite nelle corrispettive descrizioni degli schemi.

La richiesta di connessione consiste in una connect verso l’indirizzo del server, a connessione effettuata ambo le controparti avranno a disposizione un descrittore di socket che utilizzeranno per farsi delle richieste.

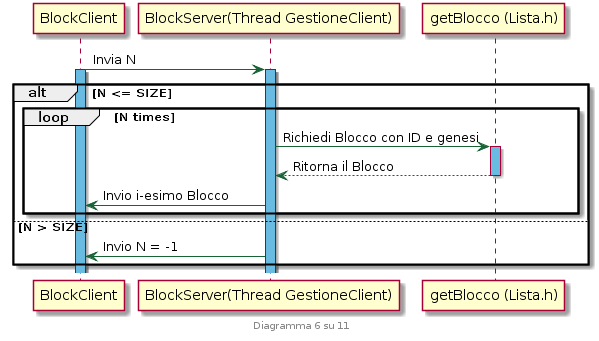
Per ogni connessione con i *BlockClient*, il BlockServer creerà un thread che avrà il compito di gestire tutte le richieste del corrispettivo *BlockClient* ad esso assegnato.

Quindi il thread GestioneClient, fin quando il *BlockClient* vuole richiedere dei servizi, dovrà restare attivo.

Come prima operazione che è stata prevista dal protocollo applicazione, ci si aspetta che il *BlockClient* invii un intero che corrisponderà ad una scelta di un determinato servizio.

I servizi previsti dal protocollo applicazione sono 6, in più è presente una scelta 0 che viene effettuata quando il *BlockClient* vuole terminare l’analisi della blockchain.

## Schema Scelta Uno



### Descrizione della scelta uno

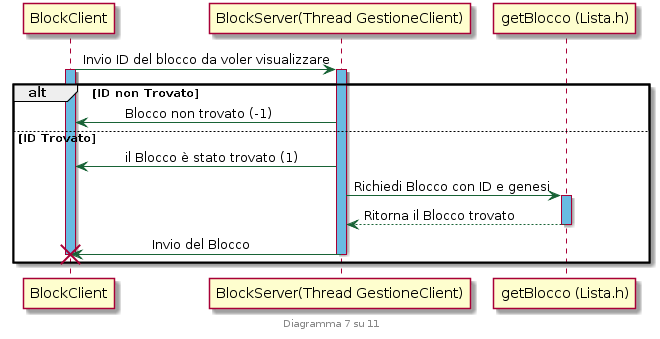
Il BlockClient invierà con una FullWrite un intero N al BlockServer, questo valore corrisponderà al numero degli ultimi blocchi delle blockchain che il BlockClient vuole visualizzare.

Il Thread GestioneClient una volta che riceve N, in mutua esclusione va a controllare se in quell’istante va a verificare se i blocchi richiesti sono presenti nella blockchain con getBlocco, dato che in concorrenza a questo thread viene eseguito anche l’altro thread che aggiorna la blockchain, la mutua esclusione viene adoperata proprio per garantire l’analisi della blockchain in un istante in cui essa non viene aggiornata.

Come risposta alla richiesta, il BlockServer invia i blocchi solo se essi sono presenti nella blockchain e li invia uno alla volta. Nel caso in cui i blocchi non sono presenti nella blockchain il client riceverà un intero (-1) che gli segnalerà l’assenza dei blocchi.

Così come adoperato per la scrittura sul file (vedi pagina 9), anche per quanto riguarda la scrittura dei blocchi su una socket, non viene adoperata direttamente la **struct Blocco**, ma viene utilizzata la **struct temp,** il motivo è sempre da ricondursi all’inutilità di inviare anche il campo **next** della **struct Blocco**, che non è altro che un indirizzo di memoria del BlockServer, di fatto superfluo per il BlockClient.

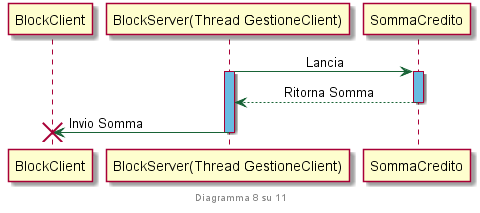
## Schema Scelta Due



### Descrizione scelta due

Il BlockClient invierà, con una FullWrite al BlockServer, un intero che corrisponde all’ID del Blocco che vuole visualizzare, il BlockServer controllerà se il blocco è presente o meno nella blockchain, se tale blocco non è stato trovato allora verrà inviato un intero negativo(-1) al BlockClient, invece se il blocco è stato trovato verrà inviato un intero positivo (1), successivamente con la funzione getBlocco verrà prelevata una copia di quel blocco dalla lista che inizia con il blocco genesi, e successivamente il blocco richiesto verrà inviato a BlockClient sottoforma di struct temp.

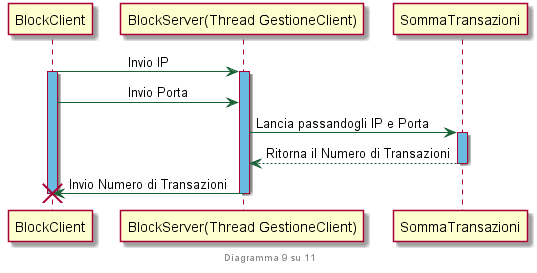
## Schema Scelta Tre



### Descrizione scelta tre

Quando il BlockClient richiede la scelta numero 3, si effettuano le operazioni descritte dal protocollo applicazione nel [Diagramma 8], cioè il BlockServer chiama la funzione SommaCredito che scorrendo tutta la lista, calcolerà con una variabile accumulatore il valore totale di tutte le transazioni, questo valore intero ritornerà al thread chiamante che a sua volta provvederà a inviarlo come valore intero al BlockClient.

## Schema Scelta Quattro



### Descrizione scelta quattro

Il BlockClient invia con una FullWrite un indirizzo IP sottoforma di array di caratteri, e successivamente invia un valore numerico Porta, queste due variabili in coppia definiscono uno specifico indirizzo che il BlockServer dovrà cercare all’interno delle transazioni, sia come destinatario che come mittente, per poi poter sommare il numero di transazioni in cui l’indirizzo è coinvolto, il BlockServer fa ciò avvalendosi della funzione Somma transazioni la quale riceve come argomenti proprio l’IP e porta richiesti dal BlockClient. La funzione SommaTransazioni ritornerà il numero di transazioni che successivamente il Thread GestioneClient invierà al BlockClient.