

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](#_Toc3840873)

[Descrizione e schemi dell’architettura 4](#_Toc3840874)

[Schema della rete 4](#_Toc3840875)

[Descrizione della rete 4](#_Toc3840876)

[Protocollo di livello trasporto 5](#_Toc3840877)

[Descrizione e schemi del protocollo applicazione 6](#_Toc3840878)

[Schema generale 6](#_Toc3840879)

[Descrizione schema generale 7](#_Toc3840880)

[Schema Generazione Blocchi 8](#_Toc3840881)

[Descriziome Generazione Blocchi 8](#_Toc3840882)

# **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:

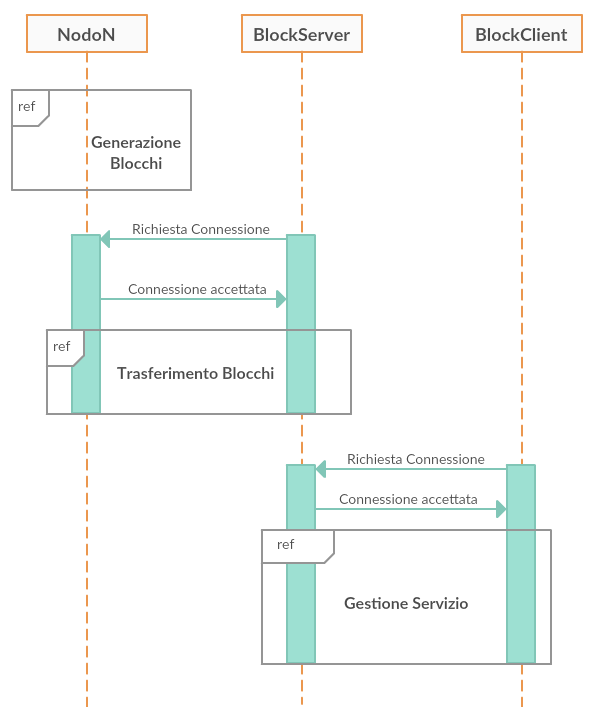
* Visualizzare le ultime n transazioni.
* Visualizzare i dati di una specifica transazione
* Visualizzare la somma dei valori di tutta la blockchain
* Visualizzare il numero di transazioni in cui è coinvolto un indirizzo specifico.
* Visualizzare i dati di tutte le transazioni in cui è coinvolto un indirizzo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

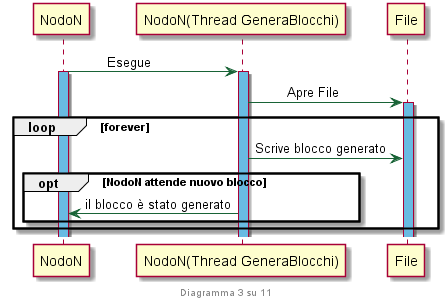


### Descrizione schema generale

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 nella 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.