Tecnologie internet

Ollari Ischimji Dmitri

2021

Contents

1 Tecnlogie internet **4**
1.1 HTTP . . . 5
1.1.1 Network protocols . . . 5
1.1.2 Protocols layer . . . 5
1.1.3 TCP(Transmission Control Protocol) . . . 5
1.1.4 UDP(User Datagram Protocol) . . . 5
1.1.5 HTTP overview . . . 5
1.1.6 Messaggio HTTP . . . 6
1.1.7 Metodi HTTP . . . 6
1.1.8 Messaggi HTTP idempotenti e sicuri/non sicuri . . . 7
1.1.9 Metodi . . . 7
1.1.10 Caratteristiche dell'header HTTP . . . 7
1.1.11 Codici di stato per HTTP . . . 7
1.1.12 Cookies . . . 8
1.1.13 Proxy . . . 8
1.1.14 Web caching . . . 8
1.2 Apache server HTTP . . . 9
1.3 HTML . . . 10
1.4 CSS . . . 11
1.5 XML . . . 12
1.6 JSON . . . 13
1.7 Search Engine for the Web . . . 14
1.7.1 Web search . . . 14
1.7.2 Motori di ricerca . . . 14
1.7.3 Web crawling . . . 14
1.7.4 PageRank . . . 14
1.8 JavaScript - Basics . . . 15
1.8.1 Incorporare JS in HTML . . . 15
1.8.2 Parole riservate per JavaScript . . . 15
1.8.3 Costanti e variabili . . . 15
1.8.4 Funzioni . . . 16
1.8.5 ArrowFunction . . . 16

1.8.6 Oggetti . . . 16
1.9 JavaScript Execution Window DOM . . . 17
1.9.1 Client Side JS timeline . . . 17
1.9.2 Same-origin policy . . . 17
1.9.3 Window Object . . . 17
1.9.4 DOM . . . 18
1.9.5 Esempio DOM . . . 18
1.10 Client-side JavaScript . . . 19
1.10.1 Gestione eventi . . . 19
1.10.2 AJAX - Asynchronous JavaScript and XML . . . 19
1.10.3 JQuery . . . 20
1.10.4 Client-side Storage . . . 21
1.10.5 Scripted Graphics . . . 21
1.10.6 TypeScript . . . 21
1.10.7 Frameworks and Development Tools . . . 21
1.10.8 Frameworks . . . 21
1.10.9 Altri strumenti . . . 21
1.11 React . . . 23
1.11.1 Introduzione . . . 23

2 Architettura Orientata ai Servizi 24
2.1 Quality of Service(QoS) . . . 24
2.1.1 SoapUI . . . 24
2.2 RESTful Services . . . 25
2.2.1 Componenti architetturali di REST . . . 25
2.2.2 Servizi RESTful . . . 25
2.2.3 Routes vs. Endpoints . . . 25
2.2.4 Guide di design REST . . . 25
2.2.5 Consumare un servizio REStful con JQUERY . . . 26
2.2.6 Express . . . 26
2.3 Microservizi . . . 27
2.3.1 Stili di comunicazione . . . 27
2.3.2 Implementing Microservice Communication . . . 27

3 Sistemi Peer-to-Peer 28
3.1 Introduzione . . . 28
3.1.1 Variabili di stato . . . 28
3.1.2 problemi di design . . . 29
3.1.3 Strategia di design per gli schemi di overlay . . . 30
3.1.4 Modello ibrido(HM) . . . 30
3.1.5 Modello decentralizzato non strutturato(DUM) . . . 31
3.1.6 Modello decentralizzata non strutturato(DUM) . . . 32
3.1.7 Modello decentralizzato strutturato(DSM) . . . 32
3.1.8 Schema overaly stratificato . . . 33
3.2 P2P - schemi di overlay popolari . . . 34
3.2.1 Content sharing . . . 34

3.3 Kademlia . . . 35
3.3.1 Distanza tra identificatori . . . 35
3.3.2 Struttura dati dei nodi . . . 35
3.3.3 Protocolllo RPCs . . . 35
3.3.4 Ricerca di nodi recursiva . . . 36
3.4 Skype . . . 37
3.4.1 Registrazione . . . 37
3.4.2 Login . . . 37
3.5 WebRTC e PeerJS . . . 38
3.5.1 WebRTC . . . 38
3.5.2 No plugins . . . 38
3.5.3 WebRTC app . . . 38
3.5.4 STUN e TURN . . . 38
3.5.5 Sicurezza . . . 38
3.5.6 PeerJS . . . 38

4 Blockchain **39**
4.1 Definizione di blockchain . . . 39
4.2 P2P Network . . . 39
4.3 Motivi per usare una blockchain . . . 40
4.4 Coin vs Token . . . 40
4.5 Transazioni . . . 40
4.6 Bitcoin . . . 41
4.6.1 Merkle Tree . . . 41
4.6.2 Transazioni Bitcoin . . . 42
4.7 Smart contract . . . 42
4.7.1 Concorrenza e smart contract . . . 43
4.8 Ethereum . . . 43
4.8.1 Ethereum accounts . . . 43
4.8.2 Smart contract ethereum . . . 43
4.8.3 Verifica di Merkle in ethereum . . . 44
4.9 Modelli di consenso . . . 44
4.9.1 Proof of Work(PoW) . . . 45
4.9.2 Proof of Stake(PoS) . . . 45
4.9.3 Eventual-consensus PoS protocols . . . 46
4.9.4 Blockwise-BA PoS protocols . . . 46
4.9.5 Problema di non avere stacking . . . 46
4.9.6 Attacco a lungo termine . . . 46
4.9.7 Blockchain ≈ Merkle hash function . . . 47
4.9.8 Eclipse attack . . . 47
4.9.9 Riassunto delle possibili falle di sicurezza . . . 47
4.9.10 Modello di consenso Round Robin . . . 47
4.9.11 Modello di consenso Proof of Authority/Proof of Identity 48
4.9.12 Proof of Elapsed Time(PoET) . . . 48

List of Figures

1 Cookies 8
2 Esempio di web caching 8
3 Architettura apache HTTPD 9
4 Parole riservate in JS 15
5 Struttura p2p 28
6 Strategie di design reti p2p 31
7 Strategie di design per gli schemi di overlay 31
8 Modello ibrido 32
9 Modello Decentralizzato non strutturato 32
10 Modello Decentralizzato strutturato 33
11 Schema di overlay a strati 33
12 Esempio di struttura dei k-buckets 35
13 Illustrazione del lookup dei nodi 36
14 Usi di una blockchain 40
15 Struttura Blocco bitcoin 41
16 Merkle Tree 41
17 Applciazione del Merkle tree 44
18 Biforcazione della blockchain 46

List of Tables

1 Tecnlogie internet

1.1 HTTP

1.1.1 Network protocols

Un protocollo è la definizione del comportamento fra due entità che devono comunicare.

1.1.2 Protocols layer

Le fuzioni del network sono strutturate con un modello layer, i dati scambiati fra i layer prendo il nome di SDU(Service Data Protocol). I dati che vengono scambiati fra uno strato protocollare ad un'altro prendo il nome di PDU(Protocol Data Unit).

1.1.3 TCP(Transmission Control Protocol)

- Trasporto affidabile
- controllo del flusso
- controllo della congestione
- connection-oriented
- non fornisce: timing, sicurezza e minimumvthroughput guarantee

1.1.4 UDP(User Datagram Protocol)

- Trasferimento non affidabile

1.1.5 HTTP overview

Http usa un modello client server, con le seguenti caratteristiche:

- user agent: client manda richieste al server(browser, bot, ecc)
- origin server: programma che accetta o rifiuta le richieste
- local cache: memoria locale(possonoaverla sia il client che il server)
- proxy: applicazione intermediaria che svolge il compito sia di client che di server
- gateway: applicazione intermediaria(non conosciuta dal client) che permette di nascondere le specifiche del server

HTTP usa TCP e sfruttano la porta 80, HTTP non mantiene informazioni riguardo richieste passate del client. Le connessioni che permette HTTP possono essere persitenti o non, nel caso delle connessioni non persitenti, dopo ogni invio di dati, la connessione viene chiusa. Nel caso delle connessioni persistenti(Invio di un file grande come un video) la connessione rimane aperta fino all'accertato invio delle risorse.

Per velocizzare l'invio e la ricezione dei pacchetti, si ricorre al pipelining, al posto di inviare un pacchetto ed aspettare la notifica di avvenuta ricezione, con il pipelining si invia i pacchetti in blocchi per ridurre il tempo complessivo.

1.1.6 Messaggio HTTP

Il messaggio HTTP è composto di un header e un body, l'header viene popolato da:

- caratteristiche di transmissione generali
- caratteristiche dell'entità di transmissione
- caratteristiche richieste
- caratteristiche di risposta

Esempio di richiesta HTTP:

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

N.B. \r\n è la combinazione per andare a capo e, se usata senza altro testo, ha lo scopo di seprare le varie parti del pacchetto come request, header e body.

1.1.7 Metodi HTTP

- GET: trasferisce la risorsa
- HEAD: trasferisce solo l'header
- POST: effettua un processo risosa-specifico sul payload
- PUT: rimpiazza tutte le rappresentazioni di una certa risorsa
- DELETE: elimina tutte le rappresentazioni di una certa risorsa
- CONNECT: stabilisci la connessioni con il server
- OPRIONS: descrivi le opzioni della comunicazione
- TRACE: fai un messsage loop-back

1.1.8 Messaggi HTTP idempotenti e sicuri/non sicuri

- idempotente: Metodo che se chiamato una o cento volte fornisce sempre lo stesso risultato
- sicuro: metodo che non modifica la risorsa(GET e POST ecc)

1.1.9 Metodi

- GET: **Sicuro e idempotente**, usato per richiedere una risorsa
- POST: **Non sicuro e nemmeno idempotente**(perche ripetere la stessa azione porta a creare duplicati), usato per creare/aggiornare una risorsa
- PUT: **Idempotente ma non sicuro**, usato per aggiornare o creare una risorsa
- DELETE: **Idempotente ma non sicuro**, elimina le risorse specificate
- HEAD: **Sicuro e idempotente**, usato per richiedere l'header di una risorsa. Usato per controllarer la validità di un URL, ecc

1.1.10 Caratteristiche dell'header HTTP

- User-agent: Descrive il client che ha fatto la richiesta
- Referer: URL del'elemento dal quale arriva la richiesta URL
- Host: Dominio e porta ai quali fare la richiesta
- From: utilizza la mail e l'user-agent della persona che fa la richiesta
- Range: specifica la grandezza della risorsa(utile nei download)
- Accept, Accept-Charset, Accept-Encoding, Accept-Language: formati di negoziazione per permettere al client e al server di usare le stesse Tecnologie.
- If-Modified-Since, If,Unmodified-Since: usato nelle richieste HTTP condozionali!!
- Authorization, Proxy-Authorization

1.1.11 Codici di stato per HTTP

- 1xx - informazionale
- 2xx - successo
- 3xx - reindirizzato
- 4xx - errore del client
- 5xx - errore del server

1.1.12 Cookies

I Cookies vengono solitamente utilizzati per autenticazione, carrelli della spesa, raccomandazioni e mantenimento dello stato dell pagina web.

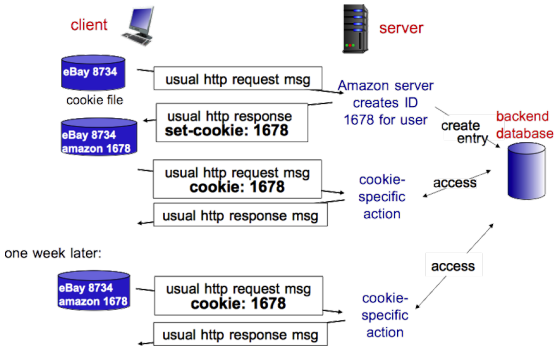

Figure 1: Cookies

1.1.13 Proxy

Un proxy è un intermediario che ha sia funzioni client che server.

1.1.14 Web caching

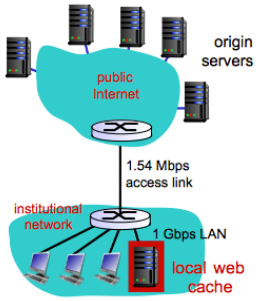

Figure 2: Esempio di web caching

1.2 Apache server HTTP

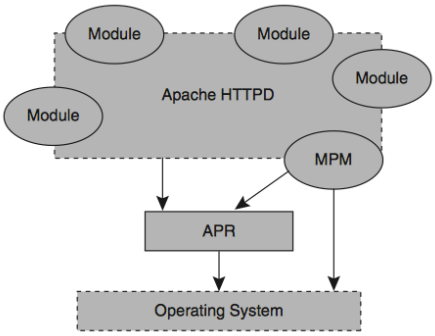

Figure 3: Architettura apache HTTPD

Apache HTTPD server si appogia al sistema operativo mediante l'APR(Apache Portable Runtime) e mediante moduli permette il fornimento di servizi web.

1.3 HTML

Linguaggio di markup per rappresentare documenti nel web.

1.4 CSS

Il css vine utilizzato per applicare stile ai documenti HTML.

1.5 XML

EXtensible Markup Language viene utilizzato per trasportare informazioni, viene utilizzato nei documenti HTML per arricchire il contenuto mediante dati dinamici.

1.6 JSON

Moderna versione di XML, rappresenta gli oggetti nel linguaggio di scripting JavaScript.

Un documento json rappresenta i dati in una maniera più leggibile di xml.

```
{"employees":[
    {
        "firstName":"John",
        "lastName":"Doe"
    },
    {
        "firstName":"Anna",
        "lastName":"Smith"
    },
    {
        "firstName":"Peter",
        "lastName":"Jones"
    }
]}
```

Al posto di:

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

1.7 Search Engine for the Web

1.7.1 Web search

L'operazione di ricerca nel web è molto simile al ricevere informazioni specifiche da un determinato server, solo che utilizza il linguggio naturale per cercare i documenti html.

1.7.2 Motori di ricerca

I motori di ricerca hanno tre caratteristiche principali:

1. Crawling: navigare in internet per cercare tutti i contenuti
2. Indexing salvare e organizzare tutti i documenti trovati durante il Crawling
3. Ranking riorganizzare i risultati della ricerca in base all'accuratezza nel rispondere al'argomento ricercato(se cerca cane, non mi aspetto di trovare navi militari ecc)

L'indice di tutti i documenti deve essere regolarmente aggiornato per includere nuove pagine.

1.7.3 Web crawling

Un crowler basilare(robot, bot, spider) consiste in:

- Una coda di URI da visutare
- Un metodo per ricevere le risorse delle pagine con HTTP
- Un page parser per estrarre informazione
- Una connessione all'indicizzatore del motore di ricerca

Il procedimento ordinario consiste in: prendere un URL, analizzare e trovare URL all'interno del testo per aggiungere questi link nella coda di analisi e continuare a indicizzare le pagine.

1.7.4 PageRank

Algoritmo per il page ranking inventato da Larry Page, utilizzato da Google per il suo motore di ricerca. PageRank è un algoritmo che analizza i link fra le risorse e il risultato viene utilizzato per il ranking. Avendo una pagina p e questa pagina è collegata alle pagine $q_1, ..., q_n$.

$$PR(p) = (1 - d) + d[\frac{PR(q_1)}{C(q_1)} + ... + \frac{PR(q_n)}{C(q_n)}]$$

1.8 JavaScript - Basics

1.8.1 Incorporare JS in HTML

```
<!DOCTYPE html>
<html>
    <body>
    <h1>My Web Page</h1>
    <p id="demo">A Paragraph</p>
    <button type="button" onclick="myFunction()">Try it</button>
    <script>
        function myFunction() {
        document.getElementById("demo").innerHTML = "Paragraph changed.";
        }
    </script>
    </body>
</html>
```

Solitamente si preferisce includere il documento JS a parte e poi importarlo con delle chiamate nel documento HTML.

1.8.2 Parole riservate per JavaScript

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Words marked with* are new in ECMAScript 5 and 6.

Figure 4: Parole riservate in JS

1.8.3 Costanti e variabili

Si usa **const** per le costanti e **let** per le variabili.

1.8.4 Funzioni

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

1.8.5 ArrowFunction

```
    const sum = (x, y) => { return x + y; };
```

1.8.6 Oggetti

```
let person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue",
    fullName:function() {return this.firstName + " " + this.lastName;}
};
```

Si può accedere ad un oggetto in due modi:

- objectName.propertyName
- objectName[propertyName]

Per accedere ai metodi: objectName.methodName()

1.9 JavaScript Execution Window DOM

1.9.1 Client Side JS timeline

1. Il browser web crea un oggetto documento e comincia a fare il parsing del documento HTML, la proprietà *document.readyState = loading*.
2. Quando il parser incontra un tag script senza async o defer, scarica lo script e lo eseguisce come se fosse uno script in linea
3. Quando il parser incontra script che hanno attributi come async o defer, semplicement scarica gli script ma non esegue nulla, continua con il parsing della pagina
4. quando tutto il documento ha finito il parsing, *document.readyState = interactive*
5. Ogni script che ha l'attributo defer, vine eseguito nell'ordine con il quale appaiono nel documento
6. Il browser accende l'evento *DOMContentLoaded* nel document object, che rappresenta la transiziona dagli script sincroni a quelli asincroni
7. quando tutta la pagina è stata caricata, compresi tutti gli script async, *document.readyState = complete*

1.9.2 Same-origin policy

Politica per la quale JavaScript, può interagire solo con il codice della sua finestra del browser, come se fosse in un sandbox(container ecc), per motivi di sicurezza.

1.9.3 Window Object

Le sue proprietà principali solo legate a:

1. timers: permette di schedulare una funzione dopo un'intervallo di tempo.
2. browser location e navigation: rappresenta l'URL corrente.
3. browser history
4. browser and screen info: Browser vendor e version number
5. dialog boxes: Semplici box di dialogo
6. document element: ogni elemento dell'HTML che ha un id può essere selezionato con questo oggetto.
7. cross-window comunication

1.9.4 DOM

DOM sta per Document Object Model e viene usato per prendere, modificare e aggiungere elementi HTML.

1.9.5 Esempio DOM

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

1.10 Client-side JavaScript

1.10.1 Gestione eventi

- onchange
- onclick
- onmouseover
- onmouseset
- onkeydown
- onload

1.10.2 AJAX - Asynchronous JavaScript and XML

AJAX permette alla pagina di essere aggiornata in maniera asincrona senza aggiornare la pagina.

1.10.3 JQuery

JQuery è una libreria che ha lo scopo di semplificare l'uso di JavaScript. JQuery ha le seguenti funioni:

- HTML/DOM manipulation
- CSS manipulation
- HTML event handling
- Effects and animations
- AJAX
- Utilities

GUARDA w3schools.com

1.10.4 Client-side Storage

Le applicazioni web possono usare le API del browser per salvare dati nel client,

- Web Storage
- Cookies
- File system access

1.10.5 Scripted Graphics

Utilizzo L'HTMl canvas in accoppiata con JS per disegnare lato client, rendendo meno opesante l'app lato server.

1.10.6 TypeScript

TypeScript implementa un controllo sulle variabili, JS non controlla se una stringa è davvero una stringa, TS lo fa!!(Come tutti i linguaggi normaliXD) TS offre le generics come in C++ dentro a JS.

1.10.7 Frameworks and Development Tools

I framework vengono usati come scheletri per JS permettendo allo sviluppatore di preoccuparsi meno sulla struttura del codice e la manutenzione. Vantaggi di Framework in JS:

- Efficenza
- Sicurezza
- Costo

1.10.8 Frameworks

- Angular
- React
- Vue

1.10.9 Altri strumenti

- Mobile-first sites development tools: Bootstrap
- Documentation tools: Swagger, JSDog, JGrouseDog, YUIDoc, Docco
- Testing tools: Jasmine, Mocha, PhantomJS, Protractor
- Debugging tools: JavaScript Debugger, Chrome Dev Tools, ng-inspector, Augury

- Security tools: Snyk, Node Security Project, RetireJS, Gemnasium, OS-Sindex
- Code optimization and analysis: JSLint, JSHint, ESLint, Flow

1.11 React

1.11.1 Introduzione

2 Architettura Orientata ai Servizi

SOA è l'Architettura prevalente per distribuire informazioni. SOA mette a disposizione l'interfaccia per un servizio(procollo, formato, comportamento) e chiunque ha l'accesso al servizio, rispettando i criteri, può effettuare richieste.

2.1 Quality of Service(QoS)

Un astessa interfaccia può corrispondere a diverse implementazioni, QoS è legato ad aspetti non-funzionali che influenzano come il servizio viene consumato:

- Performance
- Availability
- Robustness
- Required authorizetions
- Cost

Il fornitore del servizio ed il suo consumatore devono stabilire un SLA(Service Level Agreement) per gestire le possibilità di utilizzo delle "API".

2.1.1 SoapUI

SoapUI è usato per testare i servizi:

- Soap
- REST
- WSDL

2.2 RESTful Services

Rest sta per Representational State Transfer, si appoggia su comunicazioni stateless client-server.

2.2.1 Componenti architetturali di REST

- Risorsa: Sono la chiave di un vero design RESTful. Le risorse sono identificate da richieste, le risorse sono concettualemente separate dalla rappresentazione che viene inviata al client.
- Risorsa Web: Risorse piccole (Tipo la filosofia UNIX o KISS), se dentro alla risorsa ci sono elementi interessanti che però sono in altre risorse, è possibile includerle mediante link.
- Non c'è uno stato di connessione
- La risposta dovrebbe essere cacheable
- Possono essere usati dei server proxy

2.2.2 Servizi RESTful

- indipendenti dalla piattaforma
- indipendenti dal linguaggio
- basati su standard

2.2.3 Routes vs. Endpoints

Gli endpoint sono accessibili dalla API, una route è il "nome" usato per accedere all'endpoint mediante URL. Esempio: http://example.com/wp-json/wp/v2/posts/123

- la route è wp/v2/posts/123
- La rout ha 3 endpoint

2.2.4 Guide di design REST

1. Non usare indirizzi URL fisici: prova.com/ciao.xml, ma usare indirizzi logici come prova.com/saluti/002
2. Le query devono restituire il minor numero di dati possibile, se la query restituisce tanti oggetti sarebbe meglio restituirne 10 alla volta
3. La risposta di un servizio REST può essere qualsiasi cosa, è necessaria una documentazione chiara.
4. Se una risposta ha funzionalità aggiuntive, la query dovrebbe comunicare gli URL di possibili azioni e non lasciar creare URl all'utente.

2.2.5 Consumare un servizio RESTful con JQUERY

Il servizio è: http://rest-service.guides.spring.io/greeting.
Il servizio risponde: "id":1,"content":"Hello, World!".

```
public/hello.js
$(document).ready(function() {
    $.ajax({
        url: "http://rest-service.guides.spring.io/greeting"
    }).then(function(data) {
        $('.greeting-id').append(data.id);
       $('.greeting-content').append(data.content);
    });
});
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello jQuery</title>
        <script src="https://ajax.googleapis.com/ajax/libs/
        jquery/3.6.0/jquery.min.js"></script>
        <script src="hello.js"></script>
    </head>
    <body>
        <div>
            <p class="greeting-id">The ID is </p>
            <p class="greeting-content">The content is </p>
        </div>
    </body>
</html>
```

2.2.6 Express

Express.js permette di creare velocemente e facilmente API robuste.

2.3 Microservizi

Un Microservizio è un servizio indipendente che assolve us una certa problematica specifica, (sempre filosofia UNIX), si tende ad avere un applicazione che necessita di tanti microservizi che si occupano di spefifiche azioni. Vengono trattati come delle scatole nere che adempiono ad un compito!

2.3.1 Stili di comunicazione

- Sincrono bloccante
- asincrono non bloccante
- richiesta risposta
- event-driven
- dati comuni

2.3.2 Implementing Microservice Communication

Good practices:

- backwards compatibility
- interfaccia esplicita
- API indipendente dalla tecnologia
- servizi semplici per il cliente
- nascondi le implementazioni interne

Tecnologie:

- Remote Procedural Calls
  - SOAP
  - gRPC
  - CORBA
- REST
- GraphQL
- Message Brokers

3 Sistemi Peer-to-Peer

3.1 Introduzione

Il paradigma P2P, abilita due o più entità a collaborare spontaneamente in un network di peers("nodi uguali"), usando informazioni e metodi di comunicazioni appropriati senza l'utilizzo di una autorità centralizzata!!

Si può fare distinzione fra tipi di reti P2P:

- sistemi p2p dove gli utenti controllano le proprie risorse
- p2p dove i dati sono controllati da server centralizzati
- p2p ibridi con parte di utenza decentralizzata e parte server

Le attività dei sistemi p2p sono guidate dal'ambiente(environment) e dai feedback interni alla rete. Gli input dell'environment targhettano pochi peer

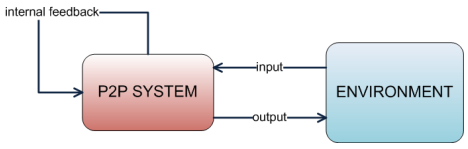

Figure 5: Struttura p2p

della rete. I nodi p2p quando ricevono dei dati dalla rete possono essere progrmmati in due modi:

- statico: una volta settato il nodo svolgere le sue funzioni sempre nello stesso nodo
- Piano adattivo: poù cambaire in corso d'opera in base a cosa gli viene detto dall'environment

3.1.1 Variabili di stato

Un sistema p2p è una rete di peers, quando si entra nella rete, ogni peer si connette con tutti gli altri. I peer condividono risorse come: banda, cpu, storage, ecc.

- Risorse replicabili: dati
- risorse consuabili: storage, cpu ecc
- Servizi distribuiti: servizi che rihiedono l'esecuzione su molti peer
- servizi locali: funzioni esposte da un peer per permettere l'accesso alle risorse locali

Ogni peer ha un tempo di vita L che viene modellata secondo: il rolo del peer, disponibilità della risorsa eed eventi non predicibili. Variabili misurate dal peer:

- uso delle risorse condivise
- tempo di risposta dei vicini
- query hit ratio (QHR)

Variabili impostate dal pper:

- dimensione della tabella di instradamento
- strategia di riempimenti delle tabelle di instradamento
- algoritmo per l'instradamento dei messaggi
- banda di unload
- banda download

Si usano metodi stocastici per misurare la distribuzione delle risorse e la loro popolarità.

3.1.2 problemi di design

La topologia della rete, il grado di centralizzazione e l'instradamento del messaggio sono cruciali per le operazioni del sistema(scalabilità, sicurezza, tolleranza agli errori, auto maantenimento).

Efficacia ed efficenza:

- scalability
- boostrapping
- connectivity management
- search performance
- consitency
- stability
- load balance
- asymmetric bandwidth

Sicurezza:

- Attacchi passivi
  - eavrsdropping

  - traffic analysis

- Attacchi attvi
  - spoofing
  - man in the middle
  - playback or replay
  - local data alteration
  - no forwarding
  - free riding
  - distributed denial of service (DDOS)
  - network poisoning

- sicurezza
  - trust managment

3.1.3 Strategia di design per gli schemi di overlay

nei sistemi p2p la posizione dell'informazione gioca un ruolo importante per l'overlay scheme.

- server centrale
- peer to peer
- salvato in locale e non pubblico

Gli schemi di overlay:

- modello ibrido(HM)
- modello decentralizzato non strutturato(DUM)
- modello decentralizzato strutturato(DSM)

3.1.4 Modello ibrido(HM)

I peer sono connessi a uno o più nodi centralizzati(server) che pubblicano le risorse. Se un peer ha bisogno di un dato in un server, manda la richiesta, gli ritorna una risposta con il percorso da prendere e prende il dato. Se il serve non è unico, le richieste potrebbere essere instradate a server vicini. Esempi: eMule, BitTorret ecc

	HM	DUM	DSM
Scalability	Low	Medium	High
Bootstrapping	Simple	Complex	Simple
Connectivity Mgmt	Simple	Complex	Complex
Time Complexity	*1*	*O(N)*	*O(logN)*
Space Complexity	*1*	*O(d)*	*O(logN)*
Consistency	High	Low	High
Stability	High	High	Low
Load Balancing	No	No	Yes

Figure 6: Strategie di design reti p2p

	HM	DUM	DSM
Eavesdropping	No	Yes	Yes
Traffic Analysis	Yes	Yes	No
Spoofing	No	Yes	Yes
Man-in-the-middle	Yes	Yes	Yes
Playback or Replay	Yes	Yes	Yes
Local Data Alteration	Yes	Yes	Yes
No-forwarding	No	Yes	No
Free Riding	Yes	Yes	Yes
DDoS	No	Yes	No
Network Poisoning	No	Yes	No

Figure 7: Strategie di design per gli schemi di overlay

3.1.5 Modello decentralizzato non strutturato(DUM)

Ogni peer propaga la richiesta ai peer adiacenti(simile alla strategia flooding) efficace in comunità piccole ma non scalabile. Per migliorare la scalabilità si fa cache delle risorse recenti. Se un'identità viene aggiunta al dato i peer possono non propagare il dato. In network grandi è necessario un time-to-live(TTL) per evitare che un pachetto stalli per sempre.

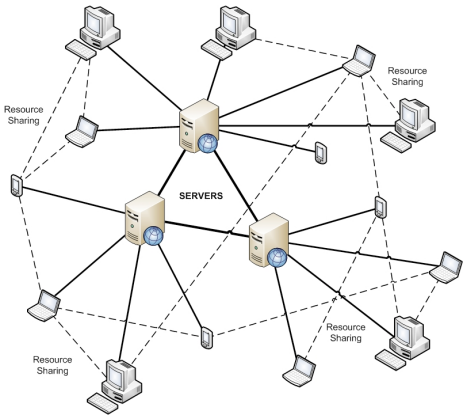

Figure 8: Modello ibrido

3.1.6 Modello decentralizzata non strutturato(DUM)

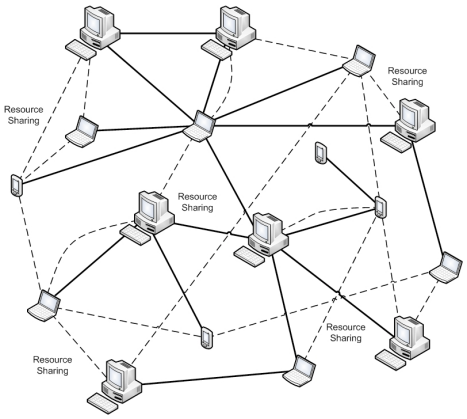

Figure 9: Modello Decentralizzato non strutturato

3.1.7 Modello decentralizzato strutturato(DSM)

Un protocollo globale fa si che ogni noda possa raggiungere la risorsa efficacemente, La tecnica più comune è una hash table distribuita dove ogni risorsa è identificata da una chiave univoca associata ad una chiave di decriptazione.

Ogni peer ha un ID assegnato ed è responsabile di tenere l'id salvato.

Pubblicazione: la descrizione di ua risorsa e la sua coppia chiave risorsa vengono messe nelle tabelle di routing adiacenti all'id del peer. Questo processo viene ripetuto finche ogni peer adiacente conosce l'id del peer in questione.

Il DSM è molto più decentralizzato del DUM.

Lookup: la query viene propagata attraverso i peer con l'ID della risorsa. Pratica comune per stabili la lunghezza dei percorsi fra nodi è:

Nodi	Route Lenght
$O(1)$	$O(N)$
$O(\log(N))$	$O(\frac{\log(N)}{\log(\log(N))})$
$O(\log(N))$	$O\log(N)$
$O\sqrt{(N)}$	$O(1)$

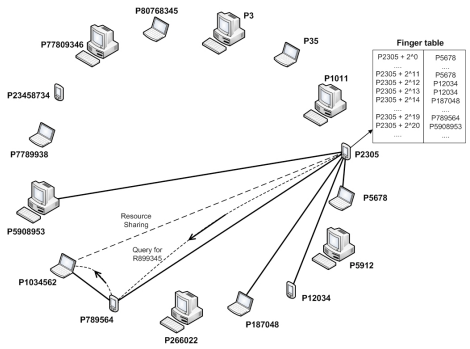

Figure 10: Modello Decentralizzato strutturato

3.1.8 Schema overaly stratificato

In una Struttura di peers con multi layer, si utilizzano dei protocolli per permettere la comunicazione fra vari layer.

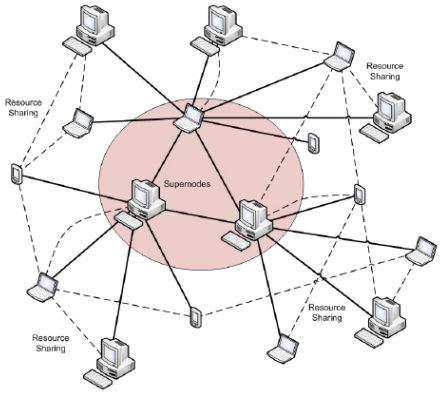

Figure 11: Schema di overlay a strati

3.2 P2P - schemi di overlay popolari

3.2.1 Content sharing

Gli utenti condividono i contenuti contribuend con le proprie risorse. L'rigine della risorsa può essere da un utente o da vari. Esempi sono:

- Napster
- eDonkey(eMule)
- BitTorrent

3.3 Kademlia

Nato nel 2002, questo protocolo si basa su DSM, utilizza delle tabelle di hash distrinuite dove ogni risorsa pubblicata viene associata ad una chiave.

3.3.1 Distanza tra identificatori

Kademlia usa una chiave di 160 bit per gli identificatori, nella fase di pubblicazione, la coppia chiave valore viene associata alla risorsa sulla base del id del nodo più vicino alla risorsa. La distanza si ottiene facendo un XOR logico.

3.3.2 Struttura dati dei nodi

Ogni nodo ha 160 k-buckets, un bucket è una lista di massimo k tuple (indirizzi IP, UDP port, Node ID) correlate a nodi che hanno una distanza dal nodo attaule compresa tra 2^i e 2^{i+1} con i tra 0 e 159.

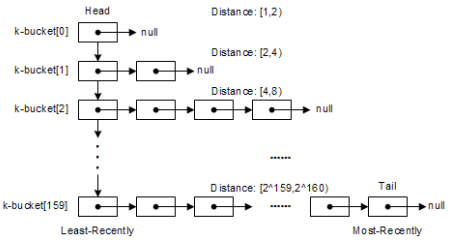

Figure 12: Esempio di struttura dei k-buckets

Quando un nodo riceve un messaggio da un'altro nodo, controlla la k-bucket che contiene la descrizione del sender, se questa descrizione è presente, il nodo muove la richiesta in coda a tutte le richieste, se la descrizione del pacchetto è assente, il noda scarta.

3.3.3 Protocolllo RPCs

kademlia ha 4 RPCs:

- PING: il peer è online
- STORE: salva un dato
- FIND NODE
- FIND VALUE

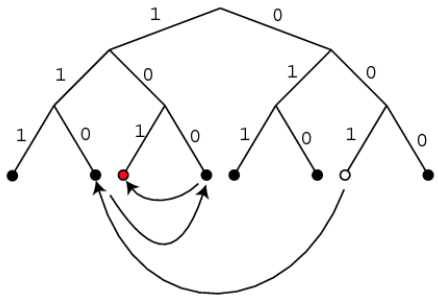

Figure 13: Illustrazione del lookup dei nodi

3.3.4 Ricerca di nodi recursiva

Per publicare una risorsa (coppia chiave valore), il nodo n cerca per i k nodi vicini e invoca il metodo STORE.

Per effettuare il lookup(ricerca) della risorsa, il nodo chiama il motodo FIND VALUE di una certa chiave e propaga a tutti i nodi adiaceni, ripetere finchè non si ottiene il valore della chiave.

3.4 Skype

Applicazione per fornire un servizio VOIP p2p, l'unica parte centralizzata è quella di login.

3.4.1 Registrazione

Durante questa parte l'utente sceglie un username A e una password P_A. Partendo da questi due dati, vengono generate:

- Coppia di chiavi RSA (S_A, V_A)
- S_A è la chiave privata
- V_A è la chiave pubblica
- il peer dell'utente conserva S_A e $H(P_A)$ in una cache locale sicura
- il peer si connette al server di login mediante una connessione basata su AES dove si generano e scambiano chiavi da 256
  - il peer manda al server A, V_A e $H(P_A)$
  - il server controlla che A non sia già registrato
  - se non registrato, procede a registrare A e $H(H(P_A))$ nel DB
  - crea un certificato di identità IC_A contenente $(A, V_A)^{S_{LS}}$ e manda il certificato al peer

3.4.2 Login

- Il peer si identificata con la sua IC al login server
- il server ritorna una lista di supernodi
- l'user avvia una connessione TCP
- il peer annuncia di ai suo peer "amici" di essere on-line
- i peer comunicano passando da dei supernodi(un po come dei router)

3.5 WebRTC e PeerJS

3.5.1 WebRTC

Progetto FOSS che permette di creare browser e mobile app con comunicazione real time. Permette di fare app come quelle per le video chiamae usando semplicemente le API di WebRTC.

3.5.2 No plugins

Molte app web usano WebRTC, ma necessitano di download, di plugin o app native.

3.5.3 WebRTC app

Un'app con WebRTC necessita di:

- GET/SET streaming audio, video e di dati
- GET network info(IP, port) per condividerlo con gli altri per poter comunicare(anche attraverso firewall o reti NAT).
- Comunica segnali per gli errori
- scambia info riguardo le capacita del client(risoluzione, codec, ecc)

3.5.4 STUN e TURN

WebRTC è disegnato per lavorare con le reti p2p, ma WebRTC è costruito per funzionare nella rete tradizionale, quindi necessita di un paio di accorgimenti:

- STUN per prendere l'IP del pc
- TURN per fare da nodi di relay se la connessione p2p fallisca

3.5.5 Sicurezza

La sicurezza è gestita per tutti i componenti di WebRTC, e le proprie API JavaScript possono essere usati solo con HTTP su TLS(HTTPS) o da localhost.

3.5.6 PeerJS

Wrappa l'implementazione browser di WebRTC per offrire una connessione p2p completa, configurabile e facile da usare.

4 Blockchain

4.1 Definizione di blockchain

un ledger distribuito(libro mastro/contabilità) è un DB replicato su diversi nodi computazionali. Ogni nodo ha una copa del ledger, ogni nodo che partecipa, si aggiorna autonomamente. Non c'è un'autorità centrale, la blockchain è una forma delle tecnologie distribuite.

Una blockchain è:

- un sistema che agisce come "fidato e terza parte", non centralizzato, sempre online, per mantenere un stato condiviso tramite scambi e computazioni sicure
- Un ledger distribuito, che salva i dati delle transazioni, raggruppate in blocchi e linkare i blocchi
- gestita da un grande gruppo di network servers
- full node: conserva una copia della blockchain, può create blocchi
- consensus: fra i full node, hanno gli stessi blocchi
- wallet: software per fare transazioni
- public = permissionless: tutti possono
  - essere un utente
  - fare transazioni
  - partecipare al consenso
- private = permissioned: le operazioni della blockchain possono essere fatte da membri autorizzati, full node identificabili, regole per l'accesso dei dati

4.2 P2P Network

I full node collaborativamente mantengono una rete p2p per lo scambio di blocchi e scambi di transazioni.

Le regole del consenso non coprono la parte di networking, i full node possono usare protocolli orienteti alla velocità fra di loro le varie transazioni.

Al primo avvio di una applicazione, questa non consoce l'IP di nessun full node, effettua una query ad un server DNS, questa query prende il nome di DNS seeds, che risponde all'applicazione con il lookup e un indirizzo IP di un full node che accetta nuove connessioni.

Per validare una transazione, un full node deve aver scaricato tutti blocchi della blockchain per poter aggiungere un nuovo blocco di transazioni.

Quando viene scoperto un nuovo blocco, questo viene inviato a tutti i peer della rete mediante la "Block Broadcasting".

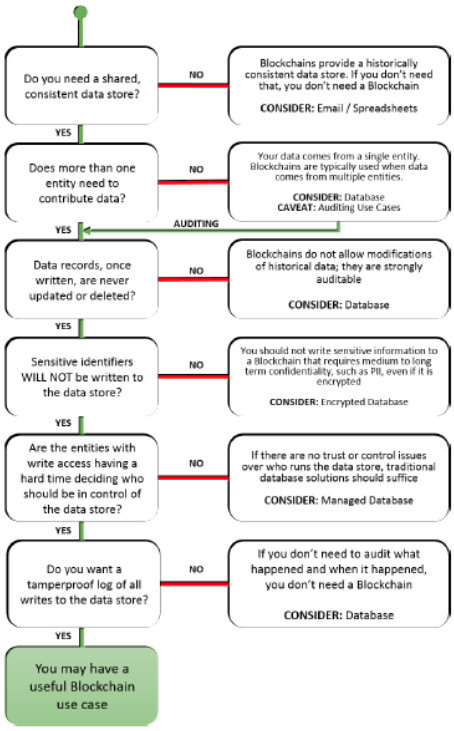

Figure 14: Usi di una blockchain

4.3 Motivi per usare una blockchain

4.4 Coin vs Token

Parlando di criptovalute associate a blockchain, si distinguono:

- Coin: unità di valuta della blockchain, usata per le transazioni
- Token: unità secondaria che risiede nella blockchain con vari scopi:
  - utility tokens: rappresentano il diritto di prendere prodotti/servizi dall'emittente del token
  - security tokens: asset digitali il cui valore deriva da quanto questo asset viene scambiato

Il security token potrebbe essere visto come "un'azione in borsa", può generare interesse, ecc

4.5 Transazioni

Le transazioni rappresentano pagamenti per beni o servizi utilizzando i "coins", le parti vengono identificate da chiavi pubbliche e ogni pagamento viee firmato

digitalmente.

4.6 Bitcoin

- Blocco di genesi: 3 Gennaio 2009
- developer: "Satoshi Nakamoto", Gavin Andresen, Wladimir van der Laan
- 1 BTC = 10^8 satoshi

Le valte basate sulla blockchain, vengono generate tramite un processo decentralizzato e competitivo detto mining. I miners di bitcoin sono i full node che processano le transazioni e rendono sicuro il network.

C'è un numero limitato di bitcoin in circolo e i bitcoin vengono generati con un andamento prevedibile(mantenendo il prezzo stabile).

Visto che il marketcap di bitcoin è ancora piccole, quantità di denaro elevate possono far oscillare il prezzo.

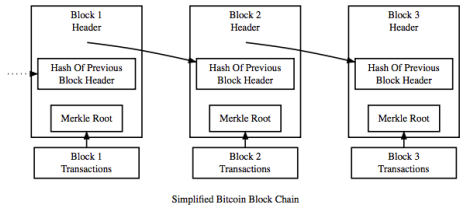

Figure 15: Struttura Blocco bitcoin

Ogni blocco di transazioni, vengono "hashate" nel **Merkle tree**, finchè la hash unica(**Merkle Root**) viene prodotta e salvata nell'header del blocco, insieme all'hash del header del blocco precedente.

4.6.1 Merkle Tree

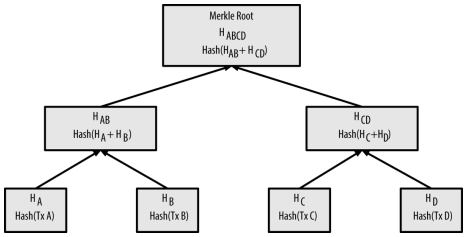

Figure 16: Merkle Tree

Il merkle tree viene creato facendo l'hash di blocchi accoppiati (le foglie dell'albero), Viene ripetuto il procedimento di pairing e hashing finchè non rimane una sola hash.

4.6.2 Transazioni Bitcoin

- Tutte le transazioni sono collegate
- **input**: riferimeto al blocco precedente
- **output**: quantità di valuta da trasferire
- ogni transazioni ha più di un input e più di un output
- Se l'output è maggiore del'input, la transazione è annullata
- se il sender invia 50 ma il reciver vuole 25, ci saranno due transazioni da 25 di cui una sarà il "resto"
- il ricevitore invia la chiave pubblica al pagatore
- il pagatore crea la transazione, specificando l'output e la firma(hash della chiave pubblica del reciver)
- il payer firma la transazione con la propria chiave privata
- la transazione viene inviata dal sender alla rete di full node
- il full node verifica la transazione e se valida la aggiunge alla blockchain

4.7 Smart contract

- Programma salvato nella blockchain
- espressione di logica contrattuale
- può implementare ogni algoritmo
- comportamento deterministico
- può interagire con altri contratti
- rende pubblica un'interfaccia
- può ritornare dati o salvare dati
- il messasggio può rappresentare eventi di interesse del contratto
- le interazioni con gli smartcontract sono salvate in blockchain
- DApp (applicazione decentralizzata): frontend + decentralized logic
- ICO (initial coin offering): crowfounding per un progetto, in cambio di coins specifici
- DAO (decentralized anonymous organization): organizzazione anonima amministrata da uno smart contract

4.7.1 Concorrenza e smart contract

contracts-as-comcurrent-objects analogy: account che usano gli smart contract in una blockchain sono come i threads di un pc.

4.8 Ethereum

- Si può usare un'implemetazione di go
- 60 milioni di ether creti grazie ai contributori del presale
- 12 miglioni dati alla fondazione, ai finanizatori e ai dev
- 5 theters crati ogni blocco(13 sec per minare un blocco)

4.8.1 Ethereum accounts

Due tipologie di account in ethereum:

- normale: controllato da coppie di chiave pubblica
- contratto: controllato dal codice interno

Chiavi private da 256 bit e pubbliche da 512 bit, vengono generate per ogni account.

4.8.2 Smart contract ethereum

Ogni smart contract ha un indirizzo unico, assegnato da una transazione speciale, dopo la quale, il programma fa parte della blockchain. Per compie una interazione, bisogna mandare un messaggio all'indirizzo del contratto. I contratti possono avere stato e memoria interna.

I contratti ethereum vengono scritti in **Solidity**(simile a javascript), compilato ed esequito nella ethereum virtual machine(EVM).

- **kovan, ropsten**: testnet
- **truffle**: framefork(test i contratti in locale)
- **ganache**: crea una blockchain locale per test

Ogni operazione effettuta da un contratto, ha un costo.

- TX = tassa totale operazione
- GU = gas usato per l'operazione
- GP = prezzo del gas

$TX = GU * GP$

E operazioni diverse hanno un consumo diverso di gas.

Il gas limit è il limite è il limite per il quale si è decisi di pagare. Impostare un gas limit troppo alto risulta meno interessante per i miners, che cercano di massimizzare i guadagni per ogni songolo blocco.

I miner guardano il prodotto tra gas price e gas limit per vedere quale contratto è più remunerativo.

4.8.3 Verifica di Merkle in ethereum

In ETH, lo stato è un'enorme struttura dati chiamata **modified merkle patricia trie**, gli header di ogni blocco contengono tre merkle tree(transazioni, receipts(ricevute? e lo stato)).

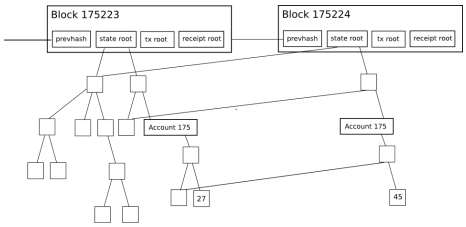

Figure 17: Applciazione del Merkle tree

Bitcoin usa un albero binario, in ETH, usando l'albero di merkle si possono ottenere struttre molto complesse come Transazioni richieste, transazioni in uscita e gli stati dell'account.

Si utilizaz il Merkle patricia tree perchè permette:

- Calcolo facile del nuovo albero dopo la modifica
- La profondità dell'albero e vincolata
- la radice dipende solo dai dati(non l'ordine degli aggiornamenti)

4.9 Modelli di consenso

Principio importante della blockchain è chi è l'utente a creare un nuovo blocco.

In blockchain "permissionless", ci sono vari nodi che creano blocchi che competono per il blocco. Fanno il nuovo blocco per vincere criptovalute.

Motivati dal guadagno.

Le seguenti proprietà sono in gioco:

- Si parte dal blocco di genesi
- gli utenti sono daccordo con il modello di consenso sui blocchi creati
- ogni blocco è linkato al precedente mediante un hash

- ogni utente può verificare i blocchi.

I modelli di consenso comuni sono:

- Proof of work
- Proof of Stake
- Round Robin
- Proof of Authority/Proof of Identity
- Proof of Elapsed Time

4.9.1 Proof of Work(PoW)

I full node competono a creare il nuovo blocco. Spesso si utilizzano gli ASIC, PC specilizzati in fare coalcoli per la creazione dei blocchi. Clacolano l'hash di alcuni dati finchè il risultato è simile alla traccia, il primo nodo che finisce l'operazione, propone il nuovo blocco nella blockchain.

Per ogni blocco aggiunto il full node prende una ricompensa in criptovalute.

La difficolta del problema di hash viene ricalcolato continuamente per mantenere lo stesso tempo stabile fra ogni blocco creato.

In Bitcoin, SHA-256 crea una hash del merkle tree + blocco precedente + header + random finchè il risultato non rispetta la soglia.

I nuovi blocchi vengono aggiunti solo se risolvere la loro hash era difficile quanto stabilito dal metodi di consenso. Ogni N blocchi (N = 2016 in bitcoin), la sogni di difficoltà si ribilancia calcolando il tempo tra il primo e l'ultimo blocco.

In ETH l'algoritmo si chiama **Ethash**, la funzione per minare combina e fà l'hash (Keccak-256) di: random, hash dell'header del blocco e dati random del set.

L'algoritmo itera finchè la soglia non è raggiunta.

Il "Miner" che raggiunge l'hash di difficoltà accettata dalla soglia, prende la commissione, aggiunge il blocco alla blockchain e diffonde la notizia.

Non conviene aggiungere tante trasazioni in un blocco perchè rallenta la creazione del blocco.

Se qualcuno possiede il 51% di hash rate della blockchain, potrebbe modificare le transazioni e evitare che nuove transazioni vengano fatte.

Se due nodi creano un blocco nello stesso momento, la chain si biforca e se nella blockchain 2 viene aggiunto un nuovo blocco, questa seconda chain diventa più affidabile della prima poichè più difficile da ricreare(ha un blocco in più).

PoW è molto dispendiosa in quantità di energia.

4.9.2 Proof of Stake(PoS)

La blockchain mantiene proprietà di una popolazione di coins "staked" dai partecipanti. Per aggiungere un blocco, la probabilità di creare un nuovo blocco aumenta se si hanno più coins in stacking.

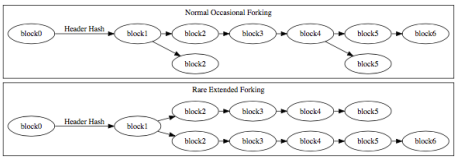

Figure 18: Biforcazione della blockchain

Vantaggi:

- minor rischio di centralizzazione
- efficienza energetica

Svantaggi:

- eleggere il creatore del blocco implica randomicità
- brecce di sicurezza.

Gli attaccanti possono fare l'hash dell'intera blockchain per sferrare l'attacco "grinding attack".

4.9.3 Eventual-consensus PoS protocols

Questo protocollo applcia una forma di "longest-chain rule" alla blockchain, l'immutabilità della blockchain aumenta con la quantità di blocchi.

4.9.4 Blockwise-BA PoS protocols

Questo protocollo ottiene l'immutabilità di ogni singolo blocco mediante il protocollo *Byzantine Agreement*.

Esempio: ALGORAND

4.9.5 Problema di non avere stacking

Visto che la validazione dipende dallo stacking, non avere stacking porta problemi come la Biforcazione.

Portando a fenomeni come il double spending, competizione per blocchi su catene diverse ecc.

4.9.6 Attacco a lungo termine

Legato al concetto di creare "brench" senza sforzo, si riferisce alla capacità di alcuni set di stackholders di eseguire la blockchain dal blocco di genesi e produrre una alternativa alla blockchain.

Due attacchi:

- posterior corruption

- stake-bleeding

Per evitare i long range attack:

- checkpoints: gli ultimi K blocchi possono essere riorganizzati
- valutazione della chiave crittografica
- restrixioni della densita della chain

4.9.7 Blockchain ≈ Merkle hash function

La struttura della blockchain è molto simile allo schema di merkle per la funzione di hash, dove la compressione è iterata su un messaggio che viene processato in blocchi.

La funzione di hash di Merkle deve soddisfare tre proprietà di sicurezza:

- preimage resistance
- second preimage resistance
- collision resistance

4.9.8 Eclipse attack

L'attacco opera al livello della rete p2p, un nodo sotto attacco ottiene una vista distorta della blockchain perchè l'attaccante controlla le comunicazioni.

4.9.9 Riassunto delle possibili falle di sicurezza

PoW:

- 51% attack

PoS:

- Nothing at Stacke
- Long-Range attack

Problemi generali:

- second preimage attack
- Eclipse attack

4.9.10 Modello di consenso Round Robin

Usato da network **Permissioned**, non ha puzzle crittografici e ha una potenza richiesta ridotta. I nodi a turno creano i blocchi ed è presente un timeout per prevenire che nodi in "stallo" alterino la crescita della blockchain.

4.9.11 Modello di consenso Proof of Authority/Proof of Identity

I nodi che creano blocchi devono avere un'identità verificata e hanno una reputazione da preservare, minore è la reputazione, minore sono le possibilità di creare blocchi. Utilizzabile solo nelle reti **permissioned**

4.9.12 Proof of Elapsed Time(PoET)

Ogni nodo richiede un timer di attesa all'**hardware time source**, l'hardware genera il tempo di attesa e una volta ricevuto il timer, vanno in idle. Una volta ripartito dall'idle, il nodo, crea un blocco, tutti gli altri blocchi interrompono l'idle e il processo riparte.

Utilizzabile in network **Permissioned e trusted**.