Protocolli di rete

- Un **protocollo** è la definizione formale di comportamenti esterni per entità comunicanti. Definisce formato, ordine di invio e ricezione dei messaggi e le azioni da intraprendere durante la trasmissione o ricezione dei messaggi.
- Un **protocollo orientato alla connessione** prevede che prima di poter trasmettere dati si devono deve stabilire la comunicazione, negoziando una connessione tra mittente e destinatario, che rimane attiva anche in assenza di scambio di dati e viene esplicitamente chiusa quando non più necessaria. Altrimenti è definito **connectionless**.
- Nei **protocolli di rete** le entità comunicanti sono componenti hardware o software di qualche dispositivo. Un esempio di protocollo potrebbe essere una richiesta ad un Web server, che avviene si digita l'URL di una pagina web nel browser: prima il computer da cui viene fatta la richiesta manda un messaggio di richiesta di connessione al web server e attende una risposta, quando il server web riceve il messaggio può trasmettere la risposta al computer. Sapendo che ora ha ricevuto il via libera per la connessione, il computer può richiedere un documento Web al server attraverso un messaggio GET con specificato il nome del documento. Infine, il server restituisce il documento richiesto al computer.

Protocolli con architettura a strati

- Ogni strato della rete ha una funzione specifica e comunica con il layer immediatamente superiore e inferiore. Quando uno strato invia dati a un altro, questi dati vengono chiamati **Protocol Data Unit (PDU)**. La PDU è il formato dei dati che viene scambiato tra gli strati durante la comunicazione.
- Ogni layer nella pila di protocolli usa i servizi offerti dal layer sottostante (n-1) e fornisce servizi al layer sovrastante (n+1). I dati scambiati tra i layer sono denominati **Service Data Unit (SDU)**.
- Due dispositivi che si trovano nello stesso layer di due sistemi separati sono **entità peer**.
- Le regole operative tra entità peer sono chiamate **procedure**, che specificano come i dati devono essere scambiati e gestiti.

VANTAGGI DEL LAYERING

- È una struttura esplicita che permette di identificare e comprendere le relazioni tra le parti di un sistema
- Fornisce modularità, che semplifica la manutenzione e l'aggiornamento del sistema: una modifica nell'implementazione di un layer rimane trasparente agli altri layer, le sue interfacce con cui comunica agli altri non vengono cambiate

Stack del protocollo internet

Lo stack di protocolli di Internet è composto da cinque livelli: fisico, di collegamento (link), di rete, di trasporto e applicativo:

1. Il **livello applicativo** è dove risiedono le applicazioni di rete e i relativi protocolli del livello applicativo. Nel livello applicativo di Internet troviamo molti protocolli, come:
   - HTTP (che consente la richiesta e il trasferimento di documenti web),
   - SMTP (che consente il trasferimento di messaggi e-mail),
   - FTP (che consente il trasferimento di file tra due sistemi finali).

2. Il protocollo del **livello di trasporto** di Internet (TCP o UDP) in un host sorgente passa un segmento del livello di trasporto e un indirizzo di destinazione al livello di rete.
3. Il **livello di rete** di Internet instrada un datagramma attraverso una serie di router tra la sorgente e la destinazione.
4. Per spostare un pacchetto da un nodo (host o router) al nodo successivo lungo il percorso, il livello di rete si affida ai servizi del **livello di collegamento (link)**.
    - In particolare, a ogni nodo, il livello di rete passa il datagramma al livello di collegamento, che lo consegna al nodo successivo lungo il percorso.
    - A questo nodo, il livello di collegamento passa il datagramma al livello di rete.
5. Il compito del **livello fisico** è spostare i singoli bit all'interno del frame da un nodo al successivo.
    - I protocolli in questo livello dipendono dal collegamento utilizzato e, inoltre, dal mezzo di trasmissione effettivo del collegamento.

Encapsulation

Il concetto di incapsulamento descrive come i dati vengano preparati per la trasmissione in rete aggiungendo informazioni specifiche a ogni livello del modello di protocollo:

- Livello applicativo: l'applicazione genera un messaggio.
- Livello di trasporto: il **messaggio** viene incapsulato in un segmento con un header contenente informazioni come il numero di porta e i bit per il controllo degli errori.
- Livello di rete: il **segmento** diventa un datagramma con un header che include gli indirizzi IP sorgente e destinazione.
- Livello di collegamento: il **datagramma** è ulteriormente incapsulato in un frame con un header che include dettagli come gli indirizzi MAC.
- Livello fisico: il **frame** viene trasformato in segnali fisici per la trasmissione.

Ad ogni livello, il pacchetto ha un header con informazioni per quel livello e un payload, che è il pacchetto ricevuto dal livello superiore. Questo approccio modulare rende i sistemi flessibili, scalabili e indipendenti dalla tecnologia.

PERCORSO FISICO CHE I DATI SEGUONO

I dati seguono un percorso fisico che scende lungo lo stack di protocolli del sistema sorgente, attraversa gli stack di protocolli di uno switch di livello collegamento (link-layer) e di un router, e risale poi lo stack di protocolli nel sistema di destinazione.
I router e gli switch di livello collegamento sono entrambi commutatori di pacchetto (packet switches). Analogamente ai sistemi finali, anche router e switch organizzano il loro hardware e software di rete in livelli. Tuttavia, non implementano tutti i livelli dello stack di protocolli; generalmente implementano solo i livelli inferiori.
Infine, i sistemi finali (host) implementano tutti e cinque i livelli dello stack di protocolli. Questo riflette l'architettura di Internet, che pone gran parte della complessità ai margini della rete, ossia nei sistemi finali.

Servizi di rete

Un servizio di rete è un’applicazione che viene eseguita sull’application layer dei protocolli di rete. Alcuni esempi sono: e-mail, web, messaggistica, social network, P2P file sharing.

Esempio di servizio di rete: servizio name server, che in base al nome presente sull'URL fornito dal computer fornisce l'indirizzo IP del web server associato. Comunica l'indirizzo al client, e il client lo usa per richiedere una risorsa al web server con il metodo GET. Il web server risponde con il documento html.
Per creare un servizio di rete, è necessario scrivere programmi che:

- vengano eseguiti su sistemi finali (end systems) diversi,
- comunichino tra loro attraverso la rete.
- ad esempio, un software per server web comunica con un software per browser.

Principio end-to-end:

- I dispositivi intermedi della rete (come router e switch) non eseguono servizi di rete.

PROCESSI COMUNICANTI
All'interno dello stesso host, due processi comunicano tramite comunicazione interprocesso (definita dal sistema operativo). I processi su host diversi comunicano scambiandosi messaggi.
Il processo client è il processo che inizia la comunicazione mentre il processo server attende di essere contattato.
Un socket è un endpoint di una comunicazione bidirezionale tra due programmi in esecuzione su una rete.
Un socket è associato a un numero di porta, permettendo al livello di trasporto di identificare l'applicazione destinataria del messaggio.
Per ricevere messaggi, un processo deve avere un identificatore. L'identificatore include sia l'indirizzo IP che il numero di porta associato al processo sull'host.

ARCHITETTURA CLIENT-SERVER
Il server:

- È un host sempre attivo.
- Ha un indirizzo IP permanente.
- Utilizza data center per la scalabilità.

I client:

- Comunicano con il server.
- Possono essere connessi in modo intermittente.
- Possono avere indirizzi IP dinamici.
- Non comunicano direttamente tra di loro.

ARCHITETTURA PEER-TO-PEER

- Nessun host è sempre attivo
- Peer = combinazione di client e server.
- I peer richiedono servizi ad altri peer e, in cambio, forniscono servizi ad altri peer.
- Scalabilità autonoma: nuovi peer aumentano sia la capacità di servizio che la domanda di nuovi servizi.
- I peer sono connessi in modo intermittente e cambiano indirizzo IP.
- Gestione complessa.

Transport Control Protocol TCP

- Definito nello standard RFC 793, inventato da Khan e Cerf nel 1973
- Prevede il trasporto affidabile tra processo mittente e processo destinatario.
- Il flusso è controllato, il mittente non sovraccarica il destinatario.
- Controllo dei traffici intensi: riduce la velocità di invio quando la rete è sovraccarica.
- Non fornisce temporizzazione, garanzia di throughput minimo (quantità effettiva di dati trasmessa in uno specifico periodo di tempo) o sicurezza.
- È **orientato alla connessione**, è richiesto una configurazione tra i due punti di comunicazione.

User Datagram Protocol UDP

- Definito nel RFC 768, inventato da Reed e Postel nel 1978.
- Si basa sul concetto di datagram.
- Trasferimento di dati non affidabile tra il processo mittente e il processo destinatario.
- Non fornisce affidabilità, controllo del flusso, controllo della congestione, temporizzazione, garanzia di throughput, sicurezza o setup della connessione.

WORLD WIDE WEB

- Il WWW è uno spazio informativo basato su Internet, dove i documenti e le altre risorse sono identificati da indirizzi.
- Una pagina web è un documento HTML che è collegato ad altre pagine web o risorse.

UNIFORM RESOURCE IDENTIFIERS URI

- Gli URI (sono utilizzati in tutto l'HTTP come mezzo per identificare le risorse [RFC7231].
- Nel contesto del WWW, gli URI sono informalmente indicati come Uniform Resource Locators (URL).

http://www.example.com/hello.txt

http = protocollo
www.example.com = nome host
/hello.txt = risorsa

HyperText Transfer Protocol HTTP

È un protocollo al livello applicativo definito nel RFC7231 che fornisce un'interfaccia uniforme per interagire un una risorsa a prescindere dal suo tipo, natura o implementazione, attraverso la manipolazione e trasferimento di rappresentazioni.
Adotta un modello client-server in cui l'user-agent inizializza la connessione HTTPT e invia richieste HTTPT e l'origine server accetta o rifiuta le connessioni HTTPT ed è il proprietario delle risorse richieste.
Sia il client che il server possono avere una cache locale, ossia una memoria locale.
Può essere utilizzata un'applicazione intermediaria, **il proxy**, che ha funzionalità sia del client che del server. Le ragioni del utilizzo sono ad esempio l'HTTP tunneling.
Il server può usare un'applicazione intermediaria che agisce per il suo conto e di cui il cliente non è al corrente, **il gateway**.

HTTP usa TCP

- Il client inizializza la connessione creando la socket usando porta 80.
- Il server accetta la connessione dal client.
- Vengono scambiati messaggi HTTP (messaggi del livello applicativo del protocollo) tra il browser, cioè il client HTTP, e il Web server.
- Alla fine viene chiusa la connessione HTTP.

HTTP è privo di stato

Il server non mantiene alcuna informazione su richieste passate dei client. I protocolli che mantengono lo stato sono complessi, e se il server o il client smette di funzionare, le due visioni dello stato potrebbero essere inconsistenti quindi devono essere riconciliate.

CONNESSIONE HTTP NON PERSISTENTE

- Viene inviata al massimo una risorsa per connessione dopo di che la connessione viene chiusa.
- Scaricare più risorse richiede molteplici connessioni.
- **RTT:** round trip time, il tempo impiegato da un packet di piccole dimensioni per spostarsi dal client al server e poi dal server al client.
- Nella connessione non persistente, per iniziare la connessione viene impiegato un RTT e per fare e soddisfare la richiesta viene impiegato un ulteriore RTT, dopo si ha un tempo di attesa per la trasmissione del file.
- Quindi la connessione non persistente utilizza 2RTT per risorsa.
- L'overhead del sistema operativo potrebbe aumentare a causa delle numerose richieste, che necessitano allocazione di risorse.
- I browser devono spesso aprire più connessioni TCP parallele per ottenere risorse collegate.

CONNESSIONE HTTP PERSISTENTE

- Il server lascia la connessione aperta dopo l'invio di una risposta e i messaggi seguenti tra lo stesso client e server sono inviati attraverso la connessione già aperta.
- Il client invia richieste non appena incontra una risorsa referenziata.
- Sufficiente un solo RTT per l'inizializzazione della connessione, e dopo 1 RTT per ogni risorsa.
- Permette l'uso del **pipelining**, una tecnica in cui vengono inviati più richieste HTTP su una singola connessione TCP senza attendere le risposte corrispondenti, che riduce ulteriormente il tempo di risposta.

Formato messaggio HTTP

Un messaggio HTTP è composto da tre parti principali:

- Start-line (riga iniziale) → Specifica la richiesta o la risposta.
- Header fields (campi di intestazione) → Contengono informazioni aggiuntive sul messaggio.
- Message body (corpo del messaggio) → Contiene i dati effettivi della richiesta o risposta (opzionale).

La start-line può essere di due tipi:
- Request-line → Per le richieste HTTP (quando un client invia una richiesta a un server).
- Status-line → Per le risposte HTTP (quando un server risponde al client).

HEADER HTTP
Le intestazioni HTTP sono campi di metadati inclusi nelle richieste e risposte HTTP. Seguono il formato MIME (Multipurpose Internet Mail Extensions) definito nella RFC 822 e forniscono informazioni importanti sulla trasmissione dei dati:
- Caratteristiche generali della trasmissione: data e ora della trasmissione, versione del protocollo MIME utilizzato, modalità di codifica usata per la trasmissione della risposta HTTP, il cache control, specifica se la connessione deve essere mantenuta aperta o chiusa dopo la risposta, eventuali proxy o gateway attraverso i quali passa la richiesta
- Caratteristiche dell'entità trasmessa: tipo di contenuto, lunghezza, codifica applicata al corpo del messaggio, lingua del contenuto, location del contenuto (URL dal quale è stato recuperato), scadenza oltre la quale la risorsa non è più considerata valida, la data dell'ultima modifica della risorsa.
- Caratteristiche della richiesta (es. informazioni sul client)
- Caratteristiche della risposta

Messaggi di richiesta HTTP
La prima riga è la request-line che contiene il metodo dell'azione che si svuole richiedere, l'URL e la versione di HTTP.
Alla fine del header c'è il carriage return, cioè una riga vuota che lo separa dal resto del messaggio.

METODI
Un metodo **idempotente** è un metodo HTTP che può essere chiamato più volte senza produrre risultati diversi. Non importa se il metodo viene chiamato una sola volta o dieci volte: il risultato dovrebbe essere lo stesso. Solo i metodi idempotenti possono essere pipelined.
Un metodo **sicuro** è un metodo HTTP che non modifica le risorse. Ad esempio, l'uso di GET o HEAD su un URI di risorsa non dovrebbe mai modificarne il contenuto.

GET	Sicuro e Idempotente. Viene usato per richiedere una risorsa senza modificarla. La risposta contiene una rappresentazione della risorsa.
POST	Non sicuro, Non idempotente. Usato per creare o aggiornare risorse subordinate. Il server assegna un nuovo identificatore (URI) alla risorsa creata.

PUT	Idempotente ma non sicuro. Usato per creare o aggiornare una risorsa specifica. A differenza di POST, il client specifica l'URI della risorsa. Inviare la stessa richiesta più volte produce sempre lo stesso risultato.
DELETE	Idempotente ma non sicuro. Elimina la risorsa specificata
HEAD	Idempotente e sicuro. Simile al GET, ma in questo caso il server deve rispondere solo con il header della risposta, non con il corpo del messaggio. Usato per verificare la validità, l'accessibilità e la coerenza della cache di una URI.

DIFFERENZA TRA POST E PUT

Quando viene fatto il POST di una risorsa su una raccolta, il server si occupa di associare la nuova risorsa al genitore e assegnarle un ID (nuovo URI).
È una buona pratica restituire "201 Created" e la posizione della risorsa appena creata, perché la sua posizione era sconosciuta al momento dell'invio. Questo permette al client di accedere alla nuova risorsa in seguito, se necessario.
Quando viene usato PUT per creare una risorsa il client specifica l'ID, ossia URI, della risorsa da creare.
Aggiornare una risorsa con POST permette al client di mandare tutti i valori disponibili da modificare oppure solo un sottoinsieme di valori.
Con PUT, l'aggiornamento avviene su tutti gli attributi, quindi il client deve mandare i valori per tutti gli attributi disponibile per garantire l'idempotenza.
A livello pratico POST viene usato per creare le risorse e PUT per aggiornarle.

CARATTERISTICHE DEL HEADER NELLA RICHIESTA HTTP

Le righe del header contengono:

- **User-Agent**: una stringa che descrive il client che ha originato la richiesta.
- **Referer**: URL della risorsa (es. pagina web) da cui ha avuto origine la richiesta. Deve essere vuoto se l'URL richiesto è stato digitato manualmente nel browser o selezionato dai segnalibri. Importante per la profilazione degli utenti e la pubblicità.
- **Host**: Nome di dominio e porta a cui viene effettuata la richiesta, perché l'URI nella richiesta rappresenta solo la parte locale al server.
- **From**: Indirizzo e-mail del richiedente. Richiede il consenso dell'utente, quindi raramente utilizzato.
- **Range**: Specifica un intervallo (in byte) della richiesta, utilizzato per riprendere download interrotti.
- **Accept, Accept-Charset, Accept-Encoding, Accept-Language**: Utilizzati per la negoziazione del formato. Il client indica i formati che è in grado di accettare e il server

GET CONDIZIONALE

Il GET condizionale serve per non inviare la rappresentazione della risorsa se il client possiede già una versione aggiornata nella cache. In questo modo non si hanno ritardi nella trasmissione e si ha un minore utilizzo di banda.
Il client deve specificare la data della copia memorizzata nella cache all'interno della richiesta HTTP. Se la copia è ancora valida, la risposta del server non contiene la rappresentazione della risorsa ma restituisce "HTTP/1.0 304 Not Modified".

Messaggi di risposta HTTP

Il messaggio di risposta contiene una riga iniziale con lo stato, le righe del header e i dati restituiti, ossia il file richiesto.
l codice di stato appare nella prima riga di ogni risposta del server al client. È composto da tre cifre: la prima indica la classe e le altre due specificano il tipo di risposta.

1xx - Informational: risposta temporanea mentre la richiesta è in fase di elaborazione.
2xx - Successful: il server ha ricevuto, compreso e accettato la richiesta.
3xx - Redirection: il server ha ricevuto e compreso la richiesta, ma sono necessarie ulteriori azioni da parte del client per completarla.
4xx - Client error: errore del client, come sintassi errata o richiesta non autorizzata.
5xx - Server error: il server non è in grado di soddisfare la richiesta.

Esempi: 100 Continue (if the client has not yet sent the body), 200 Ok (successful GET), 201 Created (succesful POST/PUT)

CARATTERISTICHE DEL HEADER NELLA RISPOSTA HTTP

- **Server**: stringa che descrive il server (tipo, versione, sistema operativo).
- **WWW-Authenticate**: contiene una sfida (codice speciale) per il cliente in caso di status code 401 (Unauthorized), il client genererà la sfida ricevuta dal server per generare un nuovo codice di autorizzazione da utilizzare per la prossima richiesta.
- **Accept-Ranges**: specifically i tipi di range che possono essere accettati (bytes o nessuno).

Cookies

I cookies sono una tecnologia usata dai siti Web per identificare e tenere traccia degli utenti. La tecnologia cookie ha 4 componenti:

- Una riga riguardante i cookies nel header del messaggio di risposta HTTP (**set-cookie:)**
- Una riga riguardante i cookies nel header del messaggio di richiesta HTTP(**cookie:**)
- Un file di cookie sul sistema del utente e gestito dal browser del utente
- Un database di backend del sito Web

I cookie vengono utilizzati per autenticare gli utenti, mantenendoli connessi ai loro account, per salvare gli articoli nei carrelli della spesa durante gli acquisti online, per offrire raccomandazioni personalizzate in base alla navigazione e per memorizzare informazioni temporanee, come lo stato della sessione nelle e-mail web.

Proxy

I proxy sono applicazioni intermedie tra client e server attraverso i quali passano messaggi HTTP e hanno sia funzionalità del client che del server.
Ci sono due tipi di proxy:

- **Proxy trasparente**: non modifica la richiesta o la risposta oltre quanto necessario per l'autenticazione e l'identificazione del proxy stesso. Un esempio è il tunneling HTTP.
- **Proxy non transparente**: modifica la richiesta o la risposta per offrire servizi aggiuntivi all'utente, come annotazioni di gruppo, trasformazione dei tipi di media, riduzione del protocollo o filtraggio per l'anonimato.

WEB CACHING

La cache Web utilizza un server proxy per soddisfare le richieste dei client senza coinvolgere il server di origine. L'utente configura l'indirizzo del proxy nel browser per abilitare l'accesso Web con cache.
Il browser invia tutte le richieste HTTP al proxy:

- Se la risorsa richiesta è già nella cache, il proxy la restituisce direttamente.
- Altrimenti, il proxy la recupera dal server di origine e poi la invia al client.

Il proxy funziona sia come server per il client originale che come client per il server di origine.
Tra i vantaggi della cache Web:

- Riduce il tempo di risposta per le richieste dei client.
- Diminuisce il traffico sulla rete di un'istituzione.
- Permette ai fornitori di contenuti con risorse limitate di distribuire efficacemente i loro contenuti.

Per misurare l'occupazione media di un server o di una risorsa durante un determinato periodo di tempo si utilizza l'**Intensità del Traffico**, calcolata con la formula ***aL/R*** dove:

- a è IL TASSO MEDIO DI ARRIVO DEI PACCHETTI
- L è la lunghezza media dei pacchetti
- R è la capacità del canale in bit al secondo

La cache Web aiuta a ridurre il traffico di rete, migliorare le prestazioni e prevenire il sovraccarico del server. Senza di essa, il ritardo sul collegamento di accesso può aumentare, rallentando le richieste degli utenti. Una soluzione costosa sarebbe potenziare la velocità del collegamento, ma un'alternativa più efficiente è l'uso di una cache istituzionale, che diminuisce le richieste ai server di origine, ottimizzando le risorse senza richiedere grandi investimenti.

Cos'è una SOA?
La Service-Oriented Architecture (SOA) è lo stile architetturale prevalente nei middleware attuali per i sistemi informativi distribuiti e l'integrazione delle applicazioni aziendali (EAI). L'obiettivo fondamentale della SOA è permettere a agenti e componenti software debolmente accoppiati di cooperare tra loro. Un servizio è un'unità di lavoro (operazione atomica) eseguita da un fornitore per ottenere un risultato necessario a un consumatore.

- **Gartner:** Un tipo di computing a più livelli che aiuta le organizzazioni a condividere logica e dati tra più applicazioni e modalità d'uso.
- **IBM:** Un'architettura applicativa in cui tutte le funzioni sono definite come servizi indipendenti con interfacce invocabili ben definite, che possono essere chiamate in sequenze definite per formare processi aziendali.
- **OASIS:** Un paradigma per organizzare e utilizzare capacità distribuite che possono essere sotto il controllo di domini di proprietà diversi. Fornisce un mezzo uniforme per offrire, scoprire, interagire con e utilizzare capacità per produrre effetti desiderati coerenti con precondizioni e aspettative misurabili.

Ruoli dei partecipanti e interazioni
La SOA si basa sulle interazioni tra tre ruoli:
- Fornitore: il proprietario del servizio
- Registro o Broker: gestisce i repository di informazioni sui fornitori e le loro risorse software
- Consumatore: scopre e invoca le risorse software fornite da uno o più fornitori

Cos'è un servizio?
La definizione generale di servizio è una procedura, un metodo o un oggetto con un'interfaccia pubblica e stabile, che può essere invocato da un client. Un servizio può essere considerato come la caratterizzazione astratta e l'incapsulamento dell'interfaccia di un contenuto, risorsa o capacità computazionale specifica (ad esempio, la capacità di spostare file, creare processi, fornire informazioni, verificare i diritti di accesso).

Cos'è un'interfaccia di servizio?
Un'interfaccia di servizio è definita in termini di protocollo da utilizzare per interagire con il servizio, formato dei dati scambiati e comportamento atteso dopo lo scambio di alcuni messaggi.
Interfaccia = protocollo + formato + comportamento
- Protocollo: come interagire con il servizio
- Formato: come sono strutturati i dati scambiati
- Comportamento: cosa fa il servizio

Ciclo di vita di un servizio
1) Creazione: il servizio viene pubblicato tramite
- registrazione in un servizio di directory (nelle architetture centralizzate)
- diffusione di annunci tramite messaggi (nelle architetture decentralizzate)

2) Approvvigionamento: il fornitore e il consumatore stabiliscono un contratto di fornitura del servizio, tramite
- scoperta: il consumatore trova il servizio più adatto
- negoziazione: il contratto viene concordato tra le due parti

3) Esecuzione: il servizio viene consumato.

Qualità del Servizio (QoS)
Una stessa interfaccia di servizio può corrispondere a diverse implementazioni del servizio, con fornitori e qualità del servizio (QoS) differenti.
La QoS è legata agli aspetti non funzionali che influenzano il modo in cui un servizio viene consumato come prestazioni, disponibilità, robustezza, autorizzazioni richieste, costi.
Il fornitore e il consumatore devono stabilire un **Accordo sul Livello di Servizio (SLA)**, che è un accordo sulla QoS.
I meccanismi di scoperta sono importanti per permettere ai clienti di confrontare le diverse implementazioni del servizio e selezionare quella più adatta.

Un'**API** (Interfaccia di Programmazione delle Applicazioni) è un insieme di regole e protocolli che permette a diverse applicazioni di comunicare e interagire tra loro.

Strumenti di sviluppo open source
- Apache CXF, un framework per creare servizi con supporto a vari protocolli come SOAP e REST.
- GlassFish, il primo server Java EE 7 che supporta l'approccio Java-first.
- WildFly, che supporta standard avanzati per servizi web.
- Express, una piattaforma leggera per sviluppare servizi RESTful.

SoapUI è uno strumento open source utilizzato per testare e monitorare i servizi web, in particolare quelli basati su SOAP e REST. Permette di inviare richieste, verificare risposte e automatizzare i test per garantire il corretto funzionamento dei servizi.

Postman è uno strumento per testare API, principalmente per servizi REST. Consente di inviare richieste HTTP, esaminare le risposte, e automatizzare i test. È molto utilizzato per lo sviluppo e il debug delle API, con funzionalità come la gestione di collezioni di richieste e l'automazione dei test.

cURL è uno strumento da linea di comando utilizzato per trasferire dati tramite vari protocolli di rete, come HTTP, HTTPS, FTP, e molti altri. È particolarmente utile per testare e interagire con API, permettendo di inviare richieste e ricevere risposte direttamente dal terminale.

REST

REST è l'acronimo di **Representational State Transfer** che si basa su un protocollo di comunicazione senza stato, client-server e cacheable.
Representational State Transfer è inteso per evocare l'immagine di come è composta un'applicazione Web ben progettata: una rete di pagine web (una macchina a stati virtuale), dove l'utente avanza nell'applicazione selezionando i collegamenti (transizioni di stato), il che porta alla visualizzazione della pagina successiva (che rappresenta il prossimo stato dell'applicazione) trasferita all'utente e renderizzata per il suo utilizzo.
Ogni attività di gestione dello stato deve essere eseguita dal client.
Piuttosto che utilizzare protocolli complessi come SOAP per connettere le macchine, si utilizza frequentemente l'HTTP semplice per effettuare chiamate tra le macchine, con i server Web standard (ad esempio, Apache httpd) che fungono da server REST.
Le applicazioni RESTful possono utilizzare le richieste HTTP per inviare dati (creare e/o aggiornare), leggere dati (ad esempio, effettuare query) e eliminare dati, quindi possono utilizzare HTTP per tutte e quattro le operazioni CRUD (Create/Read/Update/Delete).

REST è una soluzione leggera per implementare i servizi, alternativa alle tecnologie dei Web Service (SOAP, WSDL, ecc.).

- Nonostante sia semplice, REST è **completo**; praticamente non c'è nulla che si possa fare nei Web Service che non possa essere fatto con un'architettura RESTful.
- REST non ha una "**specifica standard**". Non ci sarà mai una raccomandazione W3C per REST, poiché è uno stile architetturale, non un protocollo.

COMPONENTI DELL'ARCHITETTURA RESTful

- **Risorse**: sono l'elemento chiave del design RESTful. Le singole risorse sono identificate nelle richiesta usando URL oppure URI nei sistemi REST basati sul web. Le risorse sono concettualmente separate dalle rappresentazioni che vengono restituite al client. Ad esempio, il server potrebbe mandare dati dal suo database come un fil XML o JSON, che non riflettono la rappresentazione interna al server.
- **Rete di risorse**: una singola risorsa non dovrebbe essere eccessivamente grande e non dovrebbe contenere dettagli troppo approfonditi. Quando è necessario accedere a ulteriori informazioni, lo si fa attraverso dei link contenenti nella risorsa originale.
- **Priva di stato della connessione**: le interazioni sono prive di stato anche se i server e le risorse possono avere uno stato. Ogni nuova richiesta dovrebbe contenere le informazioni necessarie per completarla e non può fare riferimento a richieste precedenti con lo stesso client.
- **Cacheable**: le risposte dovrebbero poter essere memorizzate nella cache quando è possibile, con data di scadenza associata. Il protocollo dovrebbe permettere al server di specificare quale risorse possono essere salvate nella cache e per quanto tempo. A questo proposito vengono usati i header dei messaggi HTTP e i client devono rispettare le specifiche di caching del server per ogni risorsa.
- **Server proxy**: possono essere usati all'interno dell'architettura per migliorare performance e scalabilità. Può essere usato qualsiasi proxy standard HTTP.

Servizi RESTful

Come i servizi Web, i servizi RESTful sono:

- Indipendenti dalla piattaforma, quindi dal sistema operativo del server
- Indipendenti dal linguaggio, diversi linguaggi possono comunicare tra loro
- Basati su standard, può essere eseguito su HTTP, CoAP, etc.
- Possono essere usati facilmente in presenza di firewall
- Non offrono funzionalità di sicurezza built in come cifratura, gestione delle sessioni, garanzia del QoS. Però questa possono essere aggiunte costruendo sopra HTTP:
  - per la sicurezza, spesso vengono utilizzati token nome utente/password;
  - per la crittografia, REST può essere utilizzato sopra HTTPS (socket sicuri);

REST e risposte dei server

Nel contesto REST, i messaggi di risposta dai server contengono dei file XML, però possono essere usati anche altri formati come CSV o JSON. A differenza dei servizi SOAP, REST non è vincolato a XML in nessun modo.
Ogni formato ha i suoi vantaggi e svantaggi. XML è facile da espandere (i client dovrebbero ignorare i campi sconosciuti) ed è sicuro dal punto di vista del tipo di dati; CSV è più compatto; e JSON è facile da analizzare nei client JavaScript (e facile da analizzare anche in altri linguaggi).
Una opzione non è accettabile come formato di risposta REST, tranne che in casi molto specifici:
HTML, o qualsiasi altro formato destinato al consumo umano e che non può essere facilmente elaborato dai client.
L'eccezione specifica è, ovviamente, quando il servizio REST è documentato per restituire un documento leggibile dall'uomo; e quando si considera l'intero WWW come un'applicazione RESTful, scopriamo che HTML è in realtà il formato di risposta REST più comune.

Gli **endpoint** sono funzioni disponibili tramite l'API.
Una **route** è il "nome" che usi per accedere agli endpoint, ed è usato nell'URL. Una rotta può avere più endpoint associati, e quello che viene utilizzato dipende dal verbo specificato.
Esempio con l'URL http://example.com/wp-json/wp/v2/posts/123
La rotta è wp/v2/posts/123 – La rotta non include wp-json perché wp-json è il percorso di base per l'API stessa. Questa rotta ha 3 endpoint:

- GET attiva un metodo get_item, restituendo i dati del post al client.
- PUT attiva un metodo update_item, prendendo i dati da aggiornare e restituendo i dati aggiornati del post.
- DELETE attiva un metodo delete_item, restituendo i dati del post ora eliminato al client.

Per aggiornare, oltre ad usare PUT, si può utilizzare anche il metodo **PATCH** per modifiche parziali a una risorsa esistente. A differenza di PUT, che sostituisce completamente la risorsa, PATCH modifica solo i campi specifici forniti nel corpo della richiesta, lasciando invariati gli altri dati della risorsa. È idempotente ma non sicuro.

Linee guida per il design REST

Non usare URL "fisici".
Un URL fisico punta a qualcosa di fisico, ad esempio, un file XML: "http://www.acme.com/inventory/product003.xml".
Un URL logico non implica un file fisico: "http://www.acme.com/inventory/products/003". Anche se con l'estensione .xml, il contenuto potrebbe essere generato dinamicamente. Ma dovrebbe essere "visibile all'umano" che l'URL è logico e non fisico.

Le query non dovrebbero restituire un sovraccarico di dati.
Se necessario, fornire un meccanismo di paginazione. Ad esempio, una richiesta GET per una "lista di prodotti" dovrebbe restituire i primi n prodotti (ad esempio, i primi 10), con link per il prossimo/precedente.

Anche se la risposta REST può essere qualsiasi cosa, assicurati che sia ben documentata, e non cambiare il formato di output alla leggera (poiché romperebbe i client esistenti).
Anche se l'output è leggibile dall'uomo, i client non sono utenti umani.

Piuttosto che lasciare che i client costruiscano URL per azioni aggiuntive, includi gli URL effettivi nelle risposte REST.
Ad esempio, una richiesta "lista di prodotti" potrebbe restituire un ID per ogni prodotto, e la specifica dice che dovresti usare http://www.acme.com/products/(product-id) per ottenere dettagli aggiuntivi. Questo è un cattivo design. Piuttosto, la risposta dovrebbe includere l'URL effettivo con ogni elemento: http://www.acme.com/products/001263, ecc.

Alcune delle applicazioni più famose delle API REST sono i social media come Facebook, YouTube. Poi altri servizi come Google, Dropbox e Amazon.

Architettura orientata ai microservizi

Il monolite

Quando tutte le funzionalità in un sistema devono essere distribuite e messe in esecuzione insieme si parla di un sistema monolitico.
In un **monolite a processo singolo** tutto il codice è confezionato in un unico processo.
Il **monolite modulare** è una variazione in cui il processo singolo è composto da moduli separati. Ognuno può essere sviluppato indipendentemente, ma tutti devono essere disponibili al momento del deployment.
Un **monolite distribuito** è un sistema che consiste in più servizi ma per specifiche ragioni il deployment deve coinvolgere tutti i servizi del sistema.

Microservizi

I microservizi sono servizi rilasciabili in modo indipendente, modellati attorno a un dominio di business.
Un servizio incapsula una funzionalità e la rende accessibile ad altri servizi esponendo uno o più endpoint di rete (ad esempio, una coda o una REST API).

L'approccio dei microservizi è emerso dall'uso nel mondo reale, basandosi su una migliore comprensione dei sistemi e dell'architettura per fare bene l'architettura orientata ai servizi (SOA). Si dovrebbe pensare ai microservizi come a un approccio specifico per SOA.

Dall'esterno, un singolo microservizio è trattato come una scatola nera, nascondendo il più possibile e esponendo il meno possibile tramite interfacce esterne. I dettagli implementativi interni sono completamente nascosti al mondo esterno. Questo significa che le architetture a microservizi evitano l'uso di database condivisi nella maggior parte dei casi; invece, ogni microservizio incapsula il proprio database, quando necessario.

Avere **confini di servizio chiari e stabili** che non cambiano quando cambia l'implementazione interna, porta a sistemi con **accoppiamento più debole** e **coesione più forte**.

Le due caratteristiche chiave dei microservizi sono:
- **Independent Deployability**: si può apportare una modifica ad un microservizio e fare il suo deployment per rendere disponibile la modifica agli utenti, senza dover fare il deployment di qualsiasi altro servizio.
- **Modellato attorno ad un dominio di business**: in questo modo è più facile introdurre nuove funzionalità e facilitare la ricombinazione dei microservizi in modi diversi per offrire nuove funzionalità.

Non esiste una risposta definitiva per quanto riguarda la dimensione che dovrebbe avere un microservizio. I microservizi dovrebbero avere un'interfaccia il più piccola possibile. Tuttavia, avere troppi microservizi "piccoli" può aumentare la complessità del sistema, richiedendo nuove competenze e tecnologie.

Un grande vantaggio dei microservizi è che supportano l'**eterogeneità tecnologica**. Possiamo scegliere lo strumento più adatto per l'implementazione di ogni servizio e cambiare facilmente la tecnologia di un microservizio senza influenzare gli altri.

Come progettare i microservizi

DOMINIO

Usare il dominio come meccanismo principale per identificare i confini dei microservizi è, in generale, l'approccio più adatto.

COESIONE

Si dovrebbero raggruppare le funzionalità in modo che sia possibile apportare modifiche nel minor numero di punti possibile.

ACCOPPIAMENTO

L'obiettivo principale di un microservizio è poter modificare e distribuire un servizio senza dover cambiare altre parti del sistema. Quindi, l’accoppiamento deve essere debole.
- **Accoppiamento di Dominio**: si verifica quando un microservizio deve interagire con un altro perché necessita delle funzionalità che quest'ultimo fornisce.

- **Accoppiamento di Passaggio**: si verifica quando un microservizio trasmette dati a un altro microservizio solo perché sono devono essere trasmessi a un ulteriore microservizio a valle, che non comunica direttamente con il microservizio originale : una modifica ai dati richiesti a valle può causare cambiamenti significativi a monte.
- **Accoppiamento Comune**: si verifica quando due o più microservizi utilizzano lo stesso set di dati. Dobbiamo evitare di usare lo stesso database per più microservizi. Piuttosto, fare in modo che solo un microservizio interagisca con il database e gestisca le interazioni tra altri microservizi e il database.

La natura dei dati che gestisci può influenzare il modo in cui decomponi i servizi.
La segregazione dei dati è spesso determinata anche da preoccupazioni legate alla privacy e alla sicurezza.

Stili di Comunicazione nei Microservizi

- **Sincrono Bloccante**: un microservizio effettua una chiamata a un altro microservizio e blocca l'operazione in attesa della risposta.
- **Asincrono Non Bloccante**: il microservizio che invia la chiamata può continuare l'elaborazione indipendentemente dal fatto che la chiamata venga ricevuta o meno.
- **Richiesta-Risposta**: un microservizio invia una richiesta a un altro microservizio affinché esegua un'operazione. Si aspetta di ricevere una risposta che lo informi del risultato.
- **Event-Driven**: i microservizi emettono eventi che altri microservizi consumano e a cui reagiscono di conseguenza. Il microservizio che emette l'evento non è a conoscenza di quali microservizi, se presenti, consumano gli eventi emessi.
- **Dati Condivisi**: i microservizi collaborano tramite una fonte di dati comune.

La scelta della tecnologia dovrebbe essere guidata principalmente dallo stile di comunicazione desiderato. Le tecnologie di comunicazione comuni sono:

- **Remote Procedure Calls (RPC):** Consente di invocare metodi locali su un processo remoto.
- **SOAP**: Protocollo basato su XML per lo scambio di informazioni tra sistemi distribuiti.
- **gRPC**: Framework ad alte prestazioni basato su HTTP/2 e Protobuf.
- **CORBA** Standard per la comunicazione tra applicazioni distribuite, indipendente dal linguaggio.
- **REST**: Permette l'accesso alle risorse attraverso semplici metodi, come quelli HTTP (GET, POST, PUT, DELETE).
- **GraphQL**: Permette di effettuare query personalizzate per ottenere dati da più microservizi downstream.
- **Message Brokers**: Middleware che gestisce la comunicazione asincrona tra microservizi, utilizzando code o topic.

Per la comunicazione sincrona richiesta-risposta si usano gRPC, REST con HTTP e GraphQL.
Per comunicazione asincrona si possono usare i Message Brokers.

Buone Pratiche per la Comunicazione tra Microservizi

- Facilitare la retrocompatibilità, per evitare problemi con i consumatori del servizio.
- Rendere esplicita l'interfaccia, per una chiara definizione delle funzionalità esposte.
- Mantenere le API indipendenti dalla tecnologia, per garantire flessibilità e interoperabilità.
- Progettare servizi semplici per i consumatori, riducendo la complessità dell'integrazione.
- Nascondere i dettagli di implementazione interna, per limitare la dipendenza da specifiche interne del microservizio.

Workflow nei microservizi

TWO-PHASE COMMIT

Nel contesto delle transazioni distribuite, uno degli attori principali è il coordinatore della transazione. Il processo si articola in due fasi:

- **Fase di Preparazione (Prepare Phase)**: Tutti i partecipanti alla transazione si preparano al commit e notificano al coordinatore di essere pronti a completare la transazione.
- **Fase di Commit o Rollback**: Il coordinatore decide se confermare (commit) o annullare (rollback) la transazione, inviando il comando a tutti i partecipanti.

Il two-phase commit è da evitare perché introduce latenza elevata, può portare a blocchi prolungati se uno dei partecipanti non risponde, non si adatta bene all'architettura a microservizi, che privilegia indipendenza e scalabilità.

SAGAS

Per gestire transazioni distribuite in modo più efficiente, si utilizza il pattern **Sagas**.
Una saga è una sequenza di transazioni locali, ognuna delle quali aggiorna i dati di un microservizio e invoca il passo successivo.
Se un passaggio fallisce, viene eseguita una transazione compensativa per annullare le operazioni precedenti.

Costruzione dei microservizi

- **Integrazione Continua**: assicura che il codice nuovo si integri correttamente con quello esistente. Un CI server rileva il commit del codice, lo estrae dal repository e verifica che il codice si compili correttamente e i test vengano eseguiti con successo. L'integrazione dovrebbe essere effettuata almeno una volta al giorno
- Bisogna evitare branch a lunga vita per lo sviluppo di feature e preferire il Trunk-Based Development, ovvero lavorare direttamente sul ramo principale. Se si utilizzano branch, mantenerli brevi.
- **Build Pipelines**: suddividere la build in più fasi.
- **Continuous Delivery**: avere feedback continuo sulla prontezza alla produzione di ogni commi, e ogni commi è trattato come un potenziale candidato di rilascio.
- **Multi-Repo**: Il codice sorgente di ogni microservizio viene conservato in una repository separata. Il codice da riutilizzare deve essere contenuto in librerie ed essere una dipendenza esplicita dei microservizi downstream.

Deployment

- Per quanto riguarda il deployment e scalabilità è possibile ospitare più database logicamente isolati sulla stessa infrastruttura fisica. Così, anche se condividono lo stesso hardware e motore database, rimangono indipendenti e non possono interferire tra loro a meno che non venga esplicitamente permesso e richiesto.
- Quando viene fatto il deployment di un software, viene eseguito in un ambiente, e ogni ambiente ha uno scopo specifico. Si hanno ambienti diversi per le diversi parti della pipeline di build: sviluppo, integrazione continua, pre-produzione, produzione.

PRINCIPI DI DEPLOYMENT NEI MICROSERVIZI

- **Esecuzione isolata**: ogni microservizio deve essere indipendente, senza interferenze con altri servizi.
- **Focus sull'automazione**: deployment deve essere automatizzato per ridurre errori e aumentare efficienza.
- **Infrastructure as Code**: infrastruttura deve essere gestita come codice, utilizzando strumenti come Kubernetes.
- **Deployment senza interruzioni del servizio**
- **Gestione del Desired State**: strumenti di orchestrazione devono garantire che il sistema mantenga sempre lo stato desiderato, auto-ripristinandosi in caso di problemi

OPZIONI PER FARE IL DEPLOYMENT

- Macchina fisica: microservizio eseguito direttamente su un server fisico.
- Virtual Machine
- Container, usando Docker e tool di orchestrazione come Kubernetes.
- Application Container: ambienti ottimizzati per eseguire specifiche applicazioni.
- Platform as a Service: piattaforme che permettono di automatizzare il deployment e la gestione dell'infrastruttura.
- Function as a Service: servizi che permettono di eseguite codice senza bisogno di server dedicati.

I microservizi possono essere eseguiti in container separati, oppure più microservizi possono essere eseguiti in un unico application container.

Sistemi peer to peer

Un sistema P2P è un particolare tipo di sistema distribuito, cioè un sistema hardware e software che contiene più di un elemento di elaborazione, elemento di archiviazione, processi concorrenti o più programmi, che funzionano sotto un regime controllato in modo debole o rigoroso.

Il paradigma P2Psi è rivelata una soluzione molto interessante per la condivisione di risorse scalabili e ad alto rendimento tra entità computazionali decentralizzate, utilizzando sistemi di informazione e comunicazione appropriati senza la necessità di un coordinamento centrale.

Un sistema P2P è un sistema complesso i cui elementi (nodi peer) sono collettori, fornitori e consumatori di risorse. La condivisione delle risorse si basa sulla collaborazione tra peer, ognuno dei quali ha una visione limitata del sistema ma contribuisce al corretto funzionamento dell'intero sistema. Ci sono tre tipi di sistemi peer-to-peer:

- Sistemi in cui ogni peer è di proprietà di un utente, che interagisce con esso tramite un'interfaccia utente.
- Sistemi in cui peer autonomi gestiscono risorse, sensori e attuatori per fornire servizi agli utenti.
- Sistemi ibridi in cui alcuni peer sono di proprietà degli utenti e da questi gestiti, mentre altri peer sono autonomi.

Le attività di un sistema peer-to-peer sono guidate dall'ambiente e dai feedback interni. Gli input ambientali di solito hanno come target un numero molto limitato di peer. Quando un peer riceve un input (dall'ambiente o da altri peer), le sue strutture interne mappano l'input in un output. Il processo di mappatura potrebbe richiedere al peer di cooperare con altri peer, scambiando messaggi per scoprire e infine consumare risorse. Comunque, reazioni localizzate seguono input localizzati.

La reazione di un peer agli input diretti o indiretti provenienti dall'ambiente è definita dalla sua struttura interna, che può essere basata:

- Su regole statiche condivise da ogni peer (protocolli).
- Sulla base di un piano adattivo che determina successive modifiche strutturali in risposta all’ambiente.

Variabili di Stato

Un sistema P2P è una rete di sovrapposizione di peer. Quando si unisce alla rete, ogni peer si connette ad altri peer secondo una strategia predeterminata. Le connessioni possono cambiare nel tempo, a seconda delle disconnessioni dei nodi e di altri fattori.
Un peer condivide risorse: larghezza di banda, cache, CPU, spazio di archiviazione, file, servizi, applicazioni. La condivisione implica che le risorse siano individuabili.
Alcuni peer condividono una quantità minima di risorse ma ne consumano molte, sono chiamati **free riders**, e alcuni sistemi li penalizzano.

Ci sono due tipi di risorse:
- Risorse replicabili: possono essere spostati o copiati da un peer all'altro; esempi sono dati e file.
- Risorse consumabili: non possono essere acquisiti (mediante replicazione), ma possono essere utilizzati direttamente solo tramite contratto con i loro ospiti; esempi sono risorse di base come spazio di archiviazione, larghezza di banda, cicli di CPU, ma anche risorse più complesse, come applicazioni e servizi.

I servizi sono raggruppati in due categorie:
- Servizi distribuiti, la cui esecuzione coinvolge molti pari.
- Servizi locali, unità funzionali esposte dai peer per consentire l'accesso remoto alle loro risorse locali.

Le variabili di stato che possono essere **misurate** del peer sono:
- utilizzo di risorse condivise (ad esempio, cache, larghezza di banda)
- tempo di risposta dei vicini
- rapporto tra richieste soddisfatte e inoltrate (QHR query hit ratio)

Le variabili di stato che possono essere **impostate** dal peer sono:
- dimensione della tabella i routing, che contiene la lista dei peer vicini
- strategia di riempimento della routing table
- algoritmo di propagazione dei messali
- banda messa a disposizione per mandare e ricevere dati

Dinamiche dei sistemi peer-to-peer

I nodi di una rete P2P non costituiscono una popolazione nel senso tradizionale del termine, perché la loro crescita numerica non è il risultato della riproduzione dei nodi esistenti. I nodi si uniscono o abbandonano la rete per molte ragioni diverse e imprevedibili, anche se la presenza di risorse interessanti è la motivazione fondamentale per gli utenti. Alcuni nodi sono sempre connessi, altri si uniscono per consumare risorse e poi se ne vanno, ecc.

Ogni peer ha una **lifetime *L*** che può essere calcolata considerando i seguenti fattori: il ruolo del peer nel sistema, la disponibilità delle risorse e gli eventi imprevedibili (ad esempio, guasti hardware).
Nei sistemi peer-to-peer (P2P), il tempo di vita residuo (Residual Lifetime Distribution) di un peer rappresenta quanto tempo rimarrà attivo prima di disconnettersi dalla rete.
La Shifted Pareto Distribution è una variante della distribuzione di Pareto, utilizzata per modellare il tempo di vita residuo nei sistemi peer-to-peer (P2P) eterogenei, dove pochi nodi rimangono connessi a lungo e la maggior parte si disconnette rapidamente.
La Distribuzione Esponenziale rappresenta processi di durata senza memoria, come il tempo di connessione degli utenti nei sistemi P2P quando il comportamento è più casuale.Le distribuzioni heavy-tailed (Pareto) hanno memoria: se un utente è già rimasto a lungo, è più probabile che continui a restare.
Le distribuzioni esponenziali sono senza memoria: la probabilità di restare online non dipende dal tempo già trascorso.

DISTRIBUZIONE DELLE RISORSE

Ci sono vari modi in cui le risorse possono essere distribuite tra i nodi di una rete P2P:

- Uniforme: ogni risorsa è distribuita equamente tra tutti i peer.
- Zipf-like: alcune risorse sono più diffuse di altre, seguendo una legge di potenza.

POPOLARITÀ DELLE RISORSE

La popolarità delle risorse è misurata dalla frequenza con cui vengono richieste. Anche la popolarità delle risorse può essere di tipo uniforme o zipf-like.

Problemi di design

La topologia, la struttura e il grado di centralizzazione della rete di overlay, nonché i meccanismi di routing e localizzazione dei messaggi impiegati per i messaggi e le risorse, sono fondamentali per il funzionamento del sistema, in quanto ne influenzano la scalabilità, la sicurezza, la tolleranza agli errori e l'automanutenibilità.

Le sfide della progettazione P2P che riguardano l'efficacia e l'efficienza sono:

- **Scalabilità**: quando aumenta il numero di peer della rete, deve essere mantenuta almeno lai stessa qualità, anche se idealmente il funzionamento della rete dovrebbe migliorare se ci sono più peer connessi.
- **Bootstrapping**: come i nuovi peer devono scoprire altri nodi per unirsi alla rete, ad esempio usando un bootstrap server, che aspetta che un nuovo nodo si connetta.
- **Gestione della connettività**: mantenere attive le connessioni e scambiare informazioni sulla topologia della rete.
- **Prestazioni della ricerca**: riguarda la velocità con cui gli utenti trovano le risorse, dipende dal algoritmo di routing.
- **Consistenza**: misura quanto i dati trovati siano aggiornati rispetto alle risorse effettivamente disponibili.
- **Stabilità**: la frequenza con cui i peer entrano ed escono dalla rete si misura con il churn rate. Un elevato churn rate può frammentare la rete e interrompere la comunicazione tra i peer.
- **Bilanciamento del carico**: misura la distribuzione equa del carico computazionale, di storage e della banda tra i nodi.
- **Larghezza di banda asimmetrica**: la capacità di upload limitata dei peer può diventare un collo di bottiglia.

Le sfide della progettazione P2P riguardano anche la sicurezza e sono dovute alla natura aperta e indipendente dei nodi della rete. Sono vulnerabili ad attacchi passivi, impercettibili, come l'intercettazione di messaggi o analisi del traffico. Sono vulnerabili anche ad attacchi attivi, che intaccano il funzionamento della rete:

- **Spoofing**: un peer si spaccia per un altro per alterare le comunicazioni.
- **Man-in-the-middle**: l'attaccante intercetta e modifica i messaggi tra due peer senza che se ne accorgano.
- **Replay attack**: un attaccante intercetta e riutilizza messaggi precedenti per simulare scambi legittimi.

- **Local Data Alteration**: modifica i dati locali di un peer per comprometterne il funzionamento.
- **No-forwarding**: ISP o peer bloccano o limitano il traffico P2P, ostacolandone il funzionamento.
- **Free Riding**: alcuni peer utilizzano risorse senza condividerne, riducendo la collaborazione nella rete.
- **DDoS (Distributed Denial of Service):**
  - Query flooding: invio massivo di richieste per sovraccaricare la rete.
  - Utilizzo del P2P per attacchi esterni: peer vengono manipolati per attaccare bersagli esterni con traffico eccessivo.
- **Network Poisoning:**
  - Index poisoning: inserimento di dati falsi nei database per compromettere le ricerche.
  - Route table poisoning: manipolazione delle tabelle di instradamento per reindirizzare traffico a un bersaglio specifico.

Gestione della Fiducia nei Sistemi P2P
- Mancanza di un'autorità centrale per gestire la fiducia tra i peer.
- La fiducia non è transitiva ma è soggettiva (se A si fida di B e B di C, non implica che A si fidi di C), quindi riguarda le singole coppie di nodi.
- La fiducia è importante quando si condividono dati o potenza di calcolo, ma è fondamentale per applicazioni e-commerce.

Schemi architettura P2P

Il posizionamento delle informazioni sulle risorse condivise svolge un ruolo importante nella caratterizzazione di uno schema di overlay. Le informazioni sulle risorse condivise possono essere:
- Pubblicate su un server centrale
- Pubblicate ad altri peer
- Archiviate localmente dai proprietari delle risorse senza essere pubblicate

Hybrid Model (HM)

Nei sistemi peer-to-peer basati sul modello HM, i peer si collegano a uno o più server centrali, sui quali pubblicano informazioni sulle risorse che offrono per la condivisione. Ogni server centrale mantiene, per ogni risorsa, l'elenco dei proprietari, in alcuni casi replicando parzialmente le liste di altri server conosciuti. Su richiesta di un peer, la directory centrale fornisce l'elenco dei peer che corrispondono alla richiesta o il miglior peer in termini di larghezza di banda e disponibilità. Le ulteriori interazioni avvengono direttamente tra il fornitore della risorsa e il consumatore. Se il server centrale non è unico, le query possono essere inoltrate anche ai server vicini.

Questa architettura è moderatamente influenzata dai problemi di sicurezza. I server centralizzati gestiscono il trasferimento dei file e permettono un maggiore controllo, rendendo molto più difficile falsificare indirizzi IP, numeri di porta, ecc. Tuttavia, per lo stesso motivo, risulta difficile garantire l’anonimato.

Non è ottimo in termini di scalabilità, la maggior parte dei server riduce drasticamente la propria efficienza, misurata in termini di reattività, quando il numero di peer connessi aumenta notevolmente.

Decentralised Unstructured Model (DUM)

Nel Modello Decentralizzato Non Strutturato (DUM), le informazioni sulle risorse sono distribuite tra i peer. Ogni peer propaga le richieste ai peer direttamente connessi secondo una strategia specifica, ad esempio inondando la rete di messaggi (flooding). Questa strategia, sebbene efficace in comunità limitate, consuma molta larghezza di banda e non è molto scalabile.
Per migliorare la scalabilità si possono adottare strategie come:

- Caching delle richieste recenti per evitare ridondanze.
- Flooding probabilistico, in cui ogni peer propaga i messaggi solo a un numero casuale di vicini.
- Identificativi unici nei messaggi, in modo che i nodi possano eliminare duplicati e prevenire la formazione di loop.
- Contatori di Time-To-Live (TTL) nelle reti di grandi dimensioni, per impedire che i messaggi si propaghino indefinitamente.

Decentralised Structure Model (DSM)

Le architetture peer-to-peer basate sul Modello Decentralizzato Strutturato (DSM) si caratterizzano per un protocollo globale coerente che garantisce che qualsiasi nodo possa instradare una ricerca verso un peer che possiede la risorsa desiderata, anche se questa è molto rara. Questo richiede una struttura più organizzata dei collegamenti tra peer.

Il modello più comune è la **tabella di hash distribuita (DHT**):

- Ogni risorsa è identificata da una chiave univoca e associata a una descrizione, formando una coppia <key, value> (chiave, descrizione).
- Ogni peer ha un ID casuale nello stesso spazio delle chiavi delle risorse ed è responsabile di conservare un sottoinsieme delle coppie <key, value>.
- Quando si pubblica una risorsa, il peer inoltra la coppia <key, value> al nodo con l'ID più simile all'ID della risorsa, ripetendo il processo finché non si raggiunge il nodo più vicino.
- Il DSM distribuisce in modo più equo la responsabilità dello storage delle informazioni sulle risorse rispetto al modello DUM.
- Quando un peer cerca una risorsa, la richiesta viene inoltrata progressivamente verso il nodo con l'ID più simile a quello della risorsa cercata.

Si possono imporre dei vincoli sulla topologia:

- Lunghezza massima del percorso: deve essere bassa per completare le richieste rapidamente.
- Grado massimo del nodo: deve essere basso per ridurre il carico di gestione della rete.
- Grado O(log N), lunghezza percorso O(log N) (più comune), non è ottimale ma offre maggiore flessibilità nella scelta dei vicini.

Layered Overlay Schemes

Nelle architetture a livelli, i peer sono raggruppati in strati, ognuno dei quali è organizzato secondo uno schema overlay piatto (HM, DUM o DSM). L'interazione tra i livelli è generalmente definita da un protocollo specifico per l'applicazione.
Esempio tipico: architettura a 2 livelli

- I peer con maggiore larghezza di banda e capacità di elaborazione agiscono come supernodi. Si occupano di instradare e propagare i messaggi nella rete.
- I peer con minori risorse (nodi foglia) sono solo fornitori e consumatori di risorse. Per pubblicare o scoprire risorse condivise, devono connettersi al livello dei supernodi.

Schemi di overlay comuni

Nella condivisione di contenuti P2P, gli utenti possono condividere i propri contenuti contribuendo con le proprie risorse l'uno all'altro. Tuttavia, poiché non vi è alcun incentivo a contribuire con contenuti o risorse, gli utenti possono tentare di ottenere contenuti senza alcun contributo. I contenuti possono essere scaricati da un'unica sorgente, oppure possono essere suddivisi tra più sorgenti.

Soulseek

Soulseek è una rete di condivisione di file e un'applicazione basata sullo schema HM. Viene utilizzata principalmente per scambiare musica, sebbene gli utenti possano condividere una varietà di file. Il server centrale coordina le ricerche e ospita le chat room, ma non partecipa effettivamente al trasferimento dei file, che avviene direttamente tra gli utenti coinvolti. Essendo uno dei primi protocolli HM, presenta molte limitazioni rispetto ad altri. In particolare, non consente il download di file da più sorgenti contemporaneamente.

Napster

Basato anch'esso sullo schema HM, il protocollo Napster ha ottenuto successo grazie a diverse implementazioni gratuite, sia open source (gnapster, Knapster, ecc.) che closed source (il client ufficiale di Napster), e anche a diverse utility correlate. Il client ufficiale è stato lanciato nel settembre 1999 e a metà del 2001 aveva oltre 25 milioni di utenti. Il tribunale distrettuale ordinò a Napster di monitorare le attività della sua rete e di bloccare l'accesso al materiale che violava i diritti quando ne veniva informato. Napster non fu in grado di farlo e quindi chiuse il suo servizio nel luglio 2001. Napster alla fine dichiarò bancarotta nel 2002 e vendette i suoi asset. Nel 2003, una divisione di Roxio Inc. ha rilasciato Napster 2.0 e i suoi server ora offrono servizi di negozio di musica online, con ampi accordi sui contenuti con le cinque principali etichette discografiche, nonché con centinaia di etichette

eDonkey

Il protocollo eDonkey, basato su HM, consente di creare reti di condivisione file per lo scambio di contenuti multimediali. I programmi client si collegano alla rete per condividere file e pubblicarli su server che possono essere configurati da chiunque. I server fungono da

hub di comunicazione per i client e consentono agli utenti di individuare i file all'interno della rete. Implementazione più utilizzata: eMule.

eMule

I server eMule forniscono servizi di indicizzazione centralizzata (simili a Napster) e non comunicano con altri server. Un client eMule utilizza una singola connessione TCP a un server eMule per:

- Accedere alla rete
- Ottenere informazioni sui file desiderati
- Trovare client disponibili

Il client eMule, inoltre, utilizza diverse centinaia di connessioni TCP con altri client, che vengono utilizzate per caricare e scaricare file. Ogni client eMule mantiene una coda di download per ciascun file che condivide. I client che scaricano si uniscono alla coda alla fine e avanzano gradualmente fino a raggiungere la testa, dove iniziano a scaricare il file.
Un client può scaricare lo stesso file da più client eMule, ricevendo frammenti diversi da ciascuno.

CONNESSIONE CLIENT-SERVER

All'avvio, il client si connette tramite TCP a un singolo server eMule.
Il server fornisce al client un ID client che è valido solo durante la durata della connessione client-server.
Successivamente, il client invia al server l'elenco dei file condivisi e la sua lista di download che contiene i file che desidera scaricare.
Il server eMule invia al client un elenco di altri client che possiedono i file che il client connesso desidera scaricare (questi client sono chiamati sorgenti).
La connessione TCP client/server rimane aperta durante tutta la sessione del client.

CONNESSIONE CLIENT-CLIENT

Un client eMule si connette a un altro client eMule (una sorgente) per scaricare un file.
Ogni client ha una coda di download che contiene un elenco di client che aspettano di scaricare i file. Un client eMule può essere nella coda di download di diversi altri client, registrato per scaricare le stesse parti di file in ciascuna.
Quando un client in download raggiunge la testa della coda di download, il client che carica inizia a inviare le parti di file richieste.
Nei primi 15 minuti, un client che sta scaricando può essere prelevato da un client in attesa con un punteggio di coda più alto.
Non si cerca di servire più di pochi client in un dato momento, garantendo una larghezza di banda minima di 2,4 kbyte/sec per ciascuno.

CLIENT ID

L'ID client è un identificatore di 4 byte fornito dal server durante il processo di handshake della connessione.
Gli ID client sono divisi in ID bassi e ID alti. Il server eMule assegna tipicamente un ID basso quando il client non può accettare connessioni in entrata o quando il client è connesso

tramite NAT o server proxy. Un ID alto viene dato ai client che consentono ad altri client di connettersi liberamente alla porta TCP di eMule sulla loro macchina host.

USER ID
L'ID utente è un valore di 16 byte creato concatenando numeri casuali.
Mentre l'ID client è valido solo durante la sessione del client con un server specifico, l'ID utente è unico e viene utilizzato per identificare un client tra le sessioni (l'ID utente identifica la postazione di lavoro).

eMule supporta uno schema crittografico progettato per prevenire frodi e impersonificazione degli utenti. L'implementazione è uno scambio semplice di challenge-response che si basa sulla crittografia RSA a chiave pubblica/privata.

FILE ID
Gli ID dei file vengono utilizzati sia per identificare in modo univoco i file nella rete sia per la rilevazione e il recupero della corruzione dei file.
Si noti che eMule non si basa sul nome del file per identificarlo in modo univoco e catalogarlo. Un file viene identificato da un ID univoco globalmente calcolato tramite l'hashing del contenuto del file.

SISTEMA DEI CREDITI
Quando un client carica file su un peer, il client che sta scaricando aggiorna il suo credito in base alla quantità di dati trasferiti. Il sistema di crediti non è globale, essendo una sorta di fiducia, il credito è soggettivo.

BitTorrent
La specificità di BitTorrent risiede nel concetto di swarm, che si riferisce al gruppo di peer che condividono un torrent: un insieme di repliche dello stesso file.
Un peer che vuole condividere contenuti (file o directory) crea un file di metainformazioni statiche con estensione .torrent e lo pubblica su uno dei server Web dei torrent. Ogni server è responsabile per collegare il file `.torrent a un tracker, che mantiene un registro globale di tutti i provider del file corrispondente.
I tracker sono responsabili per aiutare i downloader a trovarsi tra loro e formare swarms.
Poiché i tracker non partecipano direttamente allo swarm, il modello è di tipo HM (modello ibrido).

SWARMS
- Seeders: I peer che hanno il file completo e lo stanno condividendo.
- Leechers: I peer che possiedono solo alcune parti del file e stanno cercando di scaricare le altre parti.

Il seeder iniziale è il peer che fornisce per primo il contenuto.
Quando un peer si unisce a uno swarm, richiede al tracker una lista degli indirizzi IP dei peer per costruire il suo peer set iniziale. Il peer set sarà aumentato dai peer che si connettono direttamente a questo nuovo peer.
Ogni peer riporta il proprio stato al tracker ogni 30 minuti durante la fase stabile, o quando

si disconnette dal torrent, indicando ogni volta il numero di byte caricati e scaricati da quando si è unito allo swarm.
Ogni peer mantiene una lista degli altri peer che conosce, chiamato peer set. Un peer può inviare dati solo a un sottoinsieme del suo peer set, chiamato active peer set.

TRASFERIMENTO DEI FILE
I file trasferiti tramite BitTorrent sono suddivisi in pezzi di 256KB, e ogni pezzo è suddiviso in blocchi di 16KB. Un peer non può servire pezzi parzialmente ricevuti; solo i pezzi completi possono essere serviti.
BitTorrent cripta sia l'intestazione che il payload per ogni pezzo trasferito. La crittografia utilizza un cifrario da 60-80 bit. Lo scopo non è proteggere i dati, ma offuscare il flusso abbastanza da renderlo non rilevabile senza un impatto significativo sulle prestazioni.

Ogni peer che scarica segnala a tutti i suoi peer quali pezzi possiede. In questo modo, ogni peer conosce la distribuzione dei pezzi nel peer set.
- Peer Interessato: Un peer B è interessato a peer A quando peer A ha pezzi che peer B non possiede.
- Choking e Unchoking: Un peer A choke (blocca) peer B quando decide di non inviare dati a peer B. Al contrario, un peer A unchoke (sblocca) peer B quando decide di inviare dati.

I blocchi possono essere scaricati da peer diversi, e i pezzi sono distribuiti tra i peer nello swarm.

ALGORITMO RAREST FIRST - strategia selezione pezzi da scaricare
I pezzi più rari sono quelli che hanno il minore numero di copie nel peer set.
Ogni peer mantiene una lista del numero di copie di ciascun pezzo nel suo peer set. Utilizza queste informazioni per definire un set dei pezzi più rari, che include tutti i pezzi con il numero minore di copie.
Il peer seleziona il prossimo pezzo da scaricare in modo casuale dal set dei pezzi più rari.
Una volta che un pezzo è stato scaricato, il peer invia un messaggio HAVE ai peer nel suo active peer set, notificando loro che il pezzo è ora disponibile.

ALGORITMO DI CHOCKING
Questo algoritmo rappresenta la strategia di selezione dei peer utilizzata in BitTorrent, introdotta per garantire un livello ragionevole di reciprocità nelle operazioni di upload e download.
- Al massimo 4 peer possono essere interessati a un peer contemporaneamente.
- Al massimo 4 leechers possono essere sbloccati (unchoked) da un peer contemporaneamente.

Strategia dei leechers:
- Ogni 10 secondi, vengono sbloccati i 3 peer più veloci nel download.
- Ogni 30 secondi, un peer remoto viene sbloccato casualmente (optimistic unchoke), per valutare nuovi peer e aiutarli a ottenere il primo pezzo.

Strategia dei seeders:
- Ogni 10 secondi, i peer unchoked sono ordinati in base al tempo di sblocco.
- Per due periodi consecutivi, i primi 3 peer sono mantenuti unchoked, e un quarto peer viene selezionato casualmente per essere sbloccato.
- Dopo 3 periodi, i primi 4 peer sono mantenuti unchoked.

Gnutella

Il protocollo Gnutella basato su DUM è stato pubblicato alla fine del 1999 da Nullsoft.
Un nodo Gnutella (servent) si connette alla rete stabilendo una connessione con un altro nodo, tipicamente uno dei vari host noti che sono quasi sempre disponibili. In generale, l'acquisizione dell'indirizzo di un altro peer non fa parte della definizione del protocollo.
Messaggi specifici di Gnutella:
- Group Membership (Ping e Pong, per la scoperta dei peer)
- Search (Query e QueryHit, per la scoperta dei file)
- File Transfer (Push, un meccanismo che consente a un servent dietro un firewall di contribuire con dati basati su file alla rete)

Una volta che un servent riceve un messaggio QueryHit, può avviare il download diretto di uno dei file descritti dal Result Set del messaggio.
I file vengono scaricati fuori dalla rete, ossia viene stabilita una connessione diretta tra il servent di origine e quello di destinazione per eseguire il trasferimento dei dati. I dati dei file non vengono mai trasferiti attraverso la rete Gnutella. Il protocollo di download dei file è HTTP.

Ogni messaggio Ping o Query ricevuto da un nodo viene inoltrato a tutti i vicini del nodo (flooding). La specifica non fornisce raccomandazioni sulla frequenza con cui un peer dovrebbe inviare i messaggi Ping, sebbene gli sviluppatori di peer dovrebbero fare ogni sforzo per minimizzare il traffico di Ping sulla rete.
Per evitare la congestione della rete, i messaggi Ping e Query sono sempre associati a un Time To Live (TTL), che rappresenta il numero massimo di volte che il descrittore può essere inoltrato prima di essere rimosso dalla rete.

$$\mathbf{TTL(0) = TTL(i) + Hops(i)}$$

I descrittori Pong e QueryHit possono essere inviati solo lungo lo stesso percorso che ha trasportato i messaggi Ping e Query in ingresso.
Inoltre, vari studi mostrano che la distribuzione delle query Gnutella è simile alla distribuzione delle richieste HTTP su Internet (entrambi seguono la legge di Zipf). Pertanto, il meccanismo di caching del proxy utilizzato nel contesto Web potrebbe avere applicazioni utili anche nel contesto P2P di Gnutella.
La differenza tra la cache di Gnutella e la cache del Web riguarda la posizione del contenuto. Nella cache tradizionale del Web, il contenuto è fornito da proxy che sono server Web ben definiti. Al contrario, in Gnutella, il contenuto è la somma delle risposte a una query e quindi è fornito da diversi nodi.

Mute

La rete Mute (basata su DUM) utilizza gli Utility Counters (UC) al posto dei TTL.
Le query di ricerca vengono inoltrate con uno schema di broadcast, proprio come avviene in una rete basata su TTL. Tuttavia, l'UC di una query viene modificato prima di essere inoltrato in due modi distinti:

- Se un nodo genera dei risultati per una determinata query, aggiunge il numero di risultati generati all'UC prima di inoltrare la query.
- Il numero di vicini a cui un nodo prevede di inoltrare la query viene aggiunto all'UC.

I nodi in una rete basata su UC impongono un limite all'UC: se l'UC di una query raggiunge questo limite, la query viene scartata senza essere inoltrata ai vicini.

Freenet

Il sistema P2P basato su DUM Freenet è stato concepito da Clarke nel 1999, e lo sviluppo pubblico dell'implementazione di riferimento open source è iniziato all'inizio del 2000.
Nel progettare Freenet, gli autori si sono concentrati su:

- privacy per i produttori, consumatori e detentori di informazioni
- resistenza alla censura delle informazioni
- alta disponibilità e affidabilità attraverso la decentralizzazione
- memorizzazione e instradamento efficienti, scalabili e adattivi

A ogni file che deve essere condiviso viene assegnata una Content-Hash Key (CHK), calcolata eseguendo l'hash (con SHA-1) del contenuto del file da memorizzare.
Una descrizione breve e leggibile dall'uomo viene anch'essa sottoposta a hash, diventando una Signed-Subspace Key (SSK).

Per aggiungere un nuovo file, un nodo invia in rete un messaggio di inserimento contenente il file e la sua CHK assegnata, il che causa la memorizzazione del file su un insieme di nodi. I file con i puntatori alla CHK possono anche essere pubblicati utilizzando gli SSK.

Ogni nodo mantiene una **routing table** che elenca gli indirizzi di altri nodi e le CHK/SSK che pensa siano memorizzati su di essi.
I messaggi di inserimento e richiesta vengono inoltrati al nodo associato alla chiave più vicina alla chiave da inserire o da richiedere.

Per cercare un file, vengono inviati SSK request. Queste possono consentire al nodo di trovare le CHK appropriate. Successivamente, vengono inviate CHK request.
Quando una CHK request raggiunge un nodo in cui il file è memorizzato, il file viene inviato indietro lungo il percorso della CHK request.

DKS

Il Distributed K-ary Search (DKS) è probabilmente il protocollo basato su DSM più conosciuto, che include Chord e Pastry. Ogni istanza di DKS è una rete overlay completamente decentralizzata caratterizzata da tre parametri:

- N: il numero massimo di nodi che possono essere nella rete
- k: l'arità della ricerca all'interno della rete (la dimensione dello spazio delle chiavi è una potenza di k)
- f: il grado di tolleranza ai guasti

Una volta che questi parametri sono definiti, la rete risultante ha diverse proprietà desiderabili.
Ad esempio, ogni richiesta di ricerca viene risolta in al massimo $\log_k N$ salti nell'overlay, e ogni nodo deve mantenere solo $(k-1)\log_k N + 1$ indirizzi di altri nodi per scopi di instradamento.

Chord

Chord è una rete di ricerca distribuita basata su DKS(N, 2, f). Utilizza una funzione di hash (come SHA-1) che assegna a ciascun nodo un identificatore a m bit. Anche ogni risorsa ha un identificatore a m bit, quindi la dimensione dello spazio delle chiavi è 2^m. Gli identificatori dei nodi sono ordinati in un cerchio.
La chiave di una risorsa K viene assegnata al primo nodo in senso orario il cui identificatore è maggiore o uguale a K. Questo nodo è chiamato successore della chiave K.

ALGORITMO DI RICERCA DI BASE

Ogni nodo conosce il suo successore (cioè il nodo con identificatore immediatamente successivo al proprio). Una query per una chiave specifica K viene inoltrata in senso orario fino a raggiungere il primo nodo il cui identificatore è maggiore o uguale a K. La risposta alla query segue il percorso inverso, tornando al nodo che ha originato la query.
Con questo approccio, il numero di nodi da attraversare è O(N).

ALGORITMO DI RICERCA OTTIMIZZATO

Per accelerare la ricerca della risorsa, ogni nodo mantiene una finger table con al massimo m voci. La i-esima voce è: **finger[i] = successore(node_id + 2^(i-1)), con 1 ≤ i ≤ m**.
Utilizzando la tabella delle dita, per trovare il nodo il cui identificatore precede quello del successore della chiave cercata, il numero di nodi da attraversare è O(log2N) con alta probabilità. Ogni voce della tabella delle dita include sia l'identificatore Chord sia l'indirizzo IP (e il numero di porta) del nodo pertinente.

STABILIZZAZIONE

Per garantire che le ricerche vengano eseguite correttamente mentre il set di nodi partecipanti cambia, Chord introduce un protocollo di stabilizzazione che ogni nodo dovrebbe eseguire periodicamente in background e che aggiorna le finger table e i puntatori ai successori. Se qualsiasi sequenza di operazioni di join viene eseguita intercalata con le stabilizzazioni, dopo un certo periodo, ogni nodo sarà in grado di raggiungere qualsiasi altro nodo nella rete seguendo i puntatori ai successori.

Kademlia

Come protocollo basato su DSM, Kademlia specifica la struttura della rete, le regole di comunicazione tra i nodi e il modo in cui le informazioni devono essere scambiate.
Kademlia si basa sulla tecnologia **Distributed Hash Table (DHT)**, dove ogni risorsa pubblicata è associata a una coppia <chiave, valore>. Il valore può essere la risorsa stessa o semplicemente un descrittore della risorsa. La chiave è solitamente un hash del descrittore della risorsa.

DISTANZA TRA IDENTIFICATORI

Kademlia utilizza chiavi a 160 bit per identificare risorse e nodi. Durante la fase di pubblicazione, la coppia <chiave, valore> associata alla risorsa viene assegnata al nodo il cui identificatore è il più vicino, rispetto alla chiave della risorsa. La metrica adottata si basa sull'operazione logica XOR. Siano X e Y due identificatori -> $d(X,Y)=X \oplus Y$.

- $d(X,X)=0$
- $d(X,Y) > 0$ se X è diverso da Y
- $d(X,Y)=d(Y,X)$
- $d(X,Y)+d(Y,Z)=d(X,Z)$

STRUTTURA DATI DEI NODI

Ogni nodo ha 160 k-buckets. Un k-bucket è una lista di (max k) tuple <indirizzo IP, porta UDP, ID del nodo> relative ai nodi la cui distanza dal nodo corrente è compresa tra 2^i e 2^i+1 (con i compreso tra 0 e 159). Ogni k-bucket è ordinato come segue: al capo, il nodo meno recentemente contattato; alla coda, il nodo più recentemente contattato.

DISTANZA TRA IDENTIFICATORI

Quando un nodo riceve un messaggio da un altro nodo, verifica il k-bucket che dovrebbe contenere il descrittore del mittente. Se tale descrittore viene trovato, il nodo lo sposta alla coda del bucket; altrimenti, lo aggiunge alla coda del bucket. Se il k-bucket è pieno, il nodo invia un messaggio PING al nodo il cui descrittore si trova al capo del bucket (il nodo meno recentemente visto). Se non si ottiene risposta, il descrittore del nodo contattato viene rimosso dal bucket e quello del nuovo nodo viene aggiunto alla coda del bucket. In caso contrario, il descrittore del nuovo nodo viene scartato.
I pochi nodi che hanno tempo di vita lungo hanno più probabilità di restare online per più tempo, quindi vengono preferiti i nodi disponibili da più tempo.

RPC

- PING: il nodo n1 invoca PING sul nodo n2 per verificare se è online.
- STORE: il nodo n1 invoca STORE(<chiave, valore>) sul nodo n2 per fare in modo che memorizzi la coppia chiave valore.
- FIND_NODE: il nodo n1 invoca FIND_NODE(<chiave>) sul nodo n2 per ottenere i descrittori dei k nodi del k-bucket di n2 che sono più vicini alla chiave richiesta.
- FIND_VALUE: il nodo n1 invia FIND_VALUE(<chiave>) a n2, se a n2 è stato chiesto di seguire uno STORE con un valore associato a quella chiave, allora viene restituito il valore.

RICERCA RICORSIVA DEI NODI
Il nodo n seleziona α nodi dal k-bucket che è più vicino a un identificatore X. Se tale k-bucket non contiene α nodi, il nodo n ottiene i nodi α con ID più vicini a X, tra quelli che n conosce. Successivamente, n invoca FIND_NODE(X) sui nodi selezionati, in parallelo. Dai risultati ottenuti, il nodo n prende i k nodi più vicini a X. Tra questi, n seleziona α nodi che non sono stati ancora interrogati e invoca FIND_NODE(X) su di essi, in parallelo. Il processo viene ripetuto ricorsivamente. Se un giro di FIND_NODE non restituisce alcun nodo più vicino rispetto al target, n invoca FIND_NODE(X) su tutti gli altri nodi k−α (dall'ultimo insieme di nodi più vicini ottenuti). Il processo termina quando n ha interrogato con successo tutti i k nodi più vicini rispetto al target.

- Intuizione Geometrica: la distanza tra nodi dello stesso sotto-albero è più piccola rispetto alla distanza tra nodi di due sotto-alberi distinti. Kademlia sfrutta questo concetto.
- Ad ogni nuovo passo verso il target, il numero di possibili candidati viene dimezzato (in teoria), quindi mi avvicino al risultato.

PUBBLICAZIONE E RICERCA DI NODI
Per pubblicare un descrittore di risorsa (coppia <chiave, valore>, dove il valore è il descrittore e la chiave è un hash del descrittore), il nodo n cerca i k nodi più vicini rispetto alla chiave della risorsa.
Successivamente, n invoca STORE(<chiave, valore>) su di essi.
La ricerca di una coppia <chiave, valore> funziona allo stesso modo: il nodoNn cerca i k nodi più vicini alla chiave della risorsa e quindi invoca FIND_VALUE(<chiave>) su di essi.
Sia la pubblicazione che la ricerca sono caratterizzate da un tempo di esecuzione $O(\log_2 N)$, dove N è il numero di nodi nella rete.

CONNETTIVITÀ
Ogni nodo n1 entra nella rete tramite un nodo conosciuto n2.
Il nodo n1 inserisce n2 nel k-bucket più adatto, quindi avvia una ricerca del nodo con la sua chiave come parametro.
In questo modo, n1 popola i propri k-buckets e notifica la sua presenza a tutti i nodi contattati.
In stato stazionario:
la tabella di routing di ogni nodo contiene i k nodi più vicini.
un k-bucket è vuoto se, nella rete, non ci sono nodi con ID che ricadono in quel k-bucket.
la probabilità che un nodo venga contattato da un nodo a una distanza compresa tra 2^i e 2^{i+1} è costante e indipendente dal valore di i.

WebRTC
WebRTC è un progetto libero e open-source che fornisce a browser e applicazioni mobili funzionalità di Comunicazione in Tempo Reale (RTC) attraverso API semplici.

WebRTC offre agli sviluppatori di applicazioni web la possibilità di scrivere applicazioni multimediali avanzate e in tempo reale (ad esempio, video chat) direttamente sul web, senza bisogno di plugin, download o installazioni.

L'obiettivo è aiutare a costruire una solida piattaforma RTC che funzioni su più browser e su più piattaforme.
Molte applicazioni web già utilizzano RTC, ma richiedono download, app native o plugin.

Le applicazioni WebRTC devono svolgere diverse operazioni:
- Ottenere/fornire streaming audio, video o altri dati.
- Ottenere informazioni di rete come indirizzi IP e porte, ed eseguire lo scambio di queste informazioni con altri client WebRTC (noti come peer) per stabilire una connessione, anche attraverso NAT e firewall.
- Coordinare la comunicazione di signaling per segnalare errori e avviare o chiudere sessioni.
- Scambiare informazioni sui media e sulle capacità dei client, come risoluzione e codec supportati.

WebRTC è progettato per funzionare peer-to-peer, consentendo agli utenti di connettersi nel modo più diretto possibile.
Tuttavia, WebRTC è stato sviluppato per affrontare le sfide del networking nel mondo reale:
- Le applicazioni client devono attraversare gateway NAT e firewall.
- La comunicazione peer-to-peer necessita di soluzioni alternative nel caso in cui la connessione diretta fallisca.

Per gestire questa complessità, le API WebRTC utilizzano:
- Server STUN per ottenere l'indirizzo IP pubblico del computer.
- Server TURN per funzionare come server di inoltro (relay) nel caso in cui la comunicazione peer-to-peer non sia possibile.

SICUREZZA
- La crittografia è obbligatoria per tutti i componenti WebRTC.
- Le API JavaScript di WebRTC possono essere utilizzate solo da origini sicure (HTTPS o localhost).
- La specifica WebRTC assume che, una volta concessa a una pagina web l'autorizzazione all'accesso ai media, questa sia libera di condividere tali media con altre entità a sua discrezione.

PeerJS
PeerJS avvolge l'implementazione WebRTC del browser, fornendo un'API per connessioni peer-to-peer completa, configurabile e facile da usare.
Fornendo solo un ID, un peer può creare una connessione P2P per lo scambio di dati o flussi multimediali con un peer remoto.

Blockchain

Un registro distribuito è un database replicato uguale identico su più server. Ogni nodo ha una copia identica del database. Ogni nodo deve aggiornare indipendentemente il database, però le copie devono essere mantenute identiche, quindi l'azione deve essere replicata su tutti i server. Non c'è un'autorità centrale che gestisce tutto.
La blockchain è una forma particolare di registro distribuito.

La Blockchain è:

- Un sistema che agisce come una terza parte affidabile (robusto) e fidata, non centralizzata, sempre online, per preservare uno stato condiviso, mediare gli scambi e fornire calcoli sicuri.
- Un registro distribuito che memorizza dei dati di transazione, raggruppati in blocchi che costituiscono una crescente e inalterabile lista collegata, gestita da un grande gruppo di server collegati.
-

I nodi che memorizzano l'intera copia della blockchain sono dei **full nodes** e possono creare blocchi. Il **consenso** tra full nodes, vuol dire che tutti i nodi completi hanno la stessa versione della blockchain.
Gli utenti che non vogliono creare blocchi, ma vogliono solo effettuare transazioni e verificarne la validità, usano uno strumento **wallet**, le azioni sono protette utilizzando crittografia asimmetrica.

- **Blockchain pubblica**: Chiunque può essere un utente (tramite un wallet) o partecipare con un nodo completo, eseguire transazioni e prendere parte al processo di consenso che determina l'evoluzione della blockchain. Esempi sono Bitcoin, Ethereum o Algorand.
- **Blockchain privata**: Blockchain gestita da un'entità riconosciuta, con nodi completi identificabili e regole per l'accesso ai dati. Utilizzate da istituti come banche, che hanno bisogno di maggiori garanzie di fiducia.
  - **Consorzio**: blockchain privata gestita da più entità.

Reti P2P

I nodi completi mantengono collaborativamente una rete peer-to-peer per lo scambio di blocchi e transazioni. Molti wallet utilizzano questo stesso protocollo per connettersi ai nodi completi.
Le regole di consenso non coprono il networking, quindi i nodi completi possono utilizzare reti e protocolli alternativi, come la rete di inoltro rapido dei blocchi utilizzata da alcuni miner e i server dedicati alle informazioni sulle transazioni usati da alcuni programmi wallet.
Quando vengono avviati per la prima volta, i programmi non conoscono gli indirizzi IP di alcun nodo completo attivo. Per scoprirne alcuni, possono interrogare server specifici chiamati **DNS seed**. La risposta alla richiesta include uno o più record con gli indirizzi IP di nodi completi che potrebbero accettare nuove connessioni in entrata.
Prima che un nodo completo possa validare transazioni non confermate e blocchi recentemente minati, deve scaricare e convalidare tutti i blocchi a partire dal **genesis block**

fino alla punta attuale della migliore blockchain. Questo processo è chiamato Initial Block Download (IBD) o sincronizzazione iniziale.
- Trasmissione dei blocchi: quando un miner scopre un nuovo blocco, lo trasmette ai suoi nodi vicini.
- Trasmissione delle transazioni: simile alla trasmissione dei blocchi, ma riguarda le transazioni.

Motivi per cui usare una blockchain:
- Necessito di un magazzino di dati consistente e distribuito in più copie diverse.
- Se più entità devono contribuire alla creazione dei dati, oppure se ho bisogno di un audit dei dati.
- Se i dati scritti non devono più essere aggiornati o cancellati.
- Se non devono essere memorizzate informazioni sensibili. —> se ci son dati sensibili usare un database protetto da crittografia
- Mantenere l'anonimato degli utenti.
- L'entità che possono aggiungere dati non distinguono facilmente l'entità che ha il controllo del database. Voglio più entità che scrivano nel database ma non c'è fiducia reciproca, quindi nel blockchain tutti si verificano a vicenda.
- È necessario un log di chi scrive i dati.

Coin e Token
Quando si parla di criptovaluta associata a una blockchain, è importante distinguere tra:
- Coin: unità di valuta virtuale propria della blockchain, utilizzata per le transazioni.
- Token: unità secondarie che risiedono in una blockchain e possono avere vari scopi.

Esistono due tipi di token:
- Utility token: rappresenta il diritto di ottenere prodotti o servizi dall'emittente del token.
- Security token: asset digitale il cui valore deriva da un asset negoziabile e, di conseguenza, è soggetto alle leggi governative. Non si cancellano, possono essere venduti e passare tra proprietari. Avere dei security token è equivalente del possedere quote di una società.

Transazioni
Le transazioni descrivono i pagamenti per beni e servizi effettuati tramite coin.
Le parti coinvolte sono identificate dalle loro chiavi pubbliche e ogni pagamento deve essere firmato digitalmente

Bitcoin
Nel Bitcoin, la moneta virtuale è generata in un processo decentralizzato e competitivo (il mining). I miners sono full nodes che processano transazioni e mettono in sicurezza il network utilizzando hardware specializzato. Ricevono Bitcoin in cambio. In circolo c'è un numero limitato di Bitcoin, e quelli nuovi sono creati ad una frequenza prevedibile e decrescente: la domanda deve seguire l'inflazione per mantenere il prezzo stabile.

Ogni transazione di un blocco è hashata in un Merkle Tree finché non viene prodotta un'hash unica (detta Merkle Root), salvata nell'header del blocco, assieme all'hash del blocco precedente. Il Merkle Tree viene creato hashando dati accoppiati (le foglie) ricorsivamente finché non rimane un solo hash.

Le transazioni Bitcoin sono linkate insieme: nell'input si fa riferimento all'output di una transazione precedente, l'output riporta la quantità di valuta trasferita. Ogni transazione ha uno o più input, e uno o più output. Se output>input, la transazione non è valida. Se l'input è 50 ma il pagante vuole inviare solo 25, ci saranno due output: 25 per il destinatario, 25 per il pagante. Una transazione avviene quindi così:

- Il destinatario invia la propria chiave pubblica al pagante.
- Il pagante crea la transazione, specificando l'output e lo "signature script", che include l'hash della chiave pubblica del destinatario.
- Il pagante firma la transazione con la propria chiave privata.
- La transazione viene inviata dal pagante alla rete di nodi completi.
- I nodi completi verificano la transazione e, se valida, la aggiungono alla blockchain.

Smart Contracts

Uno smart contract è un programma memorizzato nella blockchain, creato attraverso transazioni speciali e dotato di un indirizzo univoco che esprime una logica contrattuale. Questo programma può implementare qualsiasi algoritmo ma deve avere un comportamento deterministico, può interagire con altri smart contract ma bisogna fare attenzione alle esecuzioni concorrenti.

Per permettere l'interazione lo smart contract deve esporre un'interfaccia, un insieme di funzioni pubbliche e può restituire o memorizzare dati.
Le interazioni con lo smart contract vengono memorizzate nella blockchain come transazioni e sono quindi tracciabili.
L'interazione si basa sul passaggio di messaggi, che possono rappresentare eventi di interesse per il contratto (es. "la rata dell'auto è stata pagata").

Le operazioni multi-transazionali degli smart contract (in blockchain come Ethereum) presentano forti analogie con i problemi classici della concorrenza in memoria condivisa:

- Contratti come oggetti concorrenti: gli account che utilizzano smart contract nella blockchain sono simili ai thread che usano oggetti concorrenti in una memoria condivisa.
- Concetti condivisi con lo studio della concorrenza: interferenza, sincronizzazione e gestione delle risorse.

Tra le applicazioni che usano la tecnologia smart contract ci sono:

- DApp (Decentralized Application): applicazione che funziona su un sistema di calcolo decentralizzato.
- ICO (Initial Coin Offering): crowdfunding per un progetto, in cambio di token specifici dell'ICO.
- DAO (Decentralized Autonomous Organization): organizzazione autonoma governata da uno smart contract.

Ethereum

Ethereum è stato rilasciata nel 2014 con l'obiettivo di fornire un World Computer flessibile, condiviso e sicuro.
Ethereum consente agli sviluppatori di programmare e distribuire smart contract sulla blockchain, permettendo transazioni complesse che vanno ben oltre il semplice trasferimento di valuta virtuale.

In Ethereum, l'offerta totale di ether e il suo tasso di emissione sono stati decisi in base alle donazioni raccolte durante la prevendita del 2014.

ACCOUNT

Ethereum ha due tipi di account:
- Account normali: controllati da coppie di chiavi pubbliche-private (ossia utenti umani).
- Account di smart contract: controllati dal proprio codice interno.

Ethereum utilizza ECDSA (Elliptic Curve Digital Signature Algorithm) con la curva ellittica secp256k1 per validare l'origine e l'integrità dei messaggi.
Un account Ethereum è rappresentato dagli ultimi 20 byte dell'hash Keccak-256 della chiave pubblica.
Best practice: usare un hardware wallet per conservare in sicurezza la chiave privata.
Differenza tra Ethereum e Bitcoin: Ethereum ha account con saldo ma Bitcoin si basa solo su transazioni.

SMART CONTRACTS

- Ogni smart contract ha un indirizzo univoco, assegnato tramite una transazione speciale.
- Una volta creato, lo smart contract diventa immutabile sulla blockchain.
- Interazione: scambio di messaggi firmati tra un account e l'indirizzo dello smart contract (e viceversa).
- Gli smart contract sono stateful, ovvero possono memorizzare dati.

Gli smart contract di Ethereum sono scritti in Solidity (linguaggio simile a JavaScript), sono compilati in bytecode, composto da istruzioni di basso livello, e vengono eseguiti sulla Etherum Virtual Machine.
Ethereum gestisce uno stato globale, che contiene:
- Tutti gli account e i loro saldi.
- Uno stato della macchina, che può variare da blocco a blocco secondo regole predefinite.
- La capacità di eseguire codice arbitrario.

Le regole di modifica dello stato da un blocco all'altro sono definite dalla EVM. Un esempio di smart contra potrebbe essere una urna elettorale virtuale.

COSTO DELLE TRANSAZIONI

Transaction Fee = Gas Used × Gas Price
- Gas Used: unità di Gas effettivamente consumate per la transazione.
- Gas Limit: unità di Gas che l'utente è disposto a pagare. Se troppo basso, c'è il rischio di Out of Gas (transazione fallita).

- Gas Price: prezzo per unità di Gas, deciso dall'utente. Più è alto, più velocemente i miner includeranno la transazione in un blocco.

Il Gas Price medio varia continuamente.
Il Block Gas Limit (BGL) è il massimo Gas che un blocco può contenere. Attualmente è 30M.
Il Gas non utilizzato viene rimborsato.
I miner massimizzano il profitto scegliendo le transazioni con valore maggiore (Gas Price × Gas Used).

Nel 2021, l'Ethereum Improvement Proposal (EIP-1559) ha modificato il calcolo e la destinazione delle commissioni.

La fee viene calcolata in base a:
- Base Fee: Determinata automaticamente dalla rete in base alla congestione. Viene bruciata (non va ai miner). La Base Fee mira a mantenere i blocchi al 50% della capacità e viene determinata in base al contenuto dell'ultimo blocco confermato. A seconda di quanto è pieno il nuovo blocco, la Base Fee viene automaticamente aumentata o diminuita.
- Max Priority Fee: Commissione opzionale, decisa dall'utente, pagata direttamente ai miner come incentivo.
- Max Fee Per Gas: Importo massimo che l'utente è disposto a pagare per unità di Gas.

Se la Base Fee aumenta mentre la transazione è in attesa, la transazione può essere bloccata, eliminata o fallire.
Quindi per assicurarsi che la transazione resti valida per almeno 6 blocchi pieni consecutivi è necessario impostare: **Max Fee = (2 × Base Fee) + Max Priority Fee**.

MERKLE TREE
In Ethereum, lo stato è una gigantesca struttura dati chiamata **Merkle Patricia Trie** modificata. Ogni intestazione di blocco (block header) contiene non uno, ma tre alberi Merkle:
- Albero delle transazioni → registra le richieste di transazione.
- Albero delle ricevute → registra gli esiti delle transazioni.
- Albero dello stato → registra lo stato degli account.

Bitcoin utilizza alberi Merkle binari, strutture dati molto efficienti per autenticare informazioni organizzate in forma di lista (ossia una serie di elementi consecutivi).
In Ethereum, invece, è necessario archiviare informazioni più complesse.
Lo stato di Ethereum è essenzialmente una mappa chiave-valore, dove:
le chiavi sono gli indirizzi degli account,
i valori contengono i dati degli account, tra cui saldo, nonce, codice e archiviazione (storage), che a sua volta è strutturato come un albero.

Lo stato di Ethereum è essenzialmente una mappa chiave-valore, dove:
- le chiavi sono gli indirizzi degli account,
- i valori contengono i dati degli account, tra cui saldo, nonce, codice e archiviazione (storage), che a sua volta è strutturato come un albero.

La **Merkle Patricia Trie** modificata è una struttura dati che consente di:
- Calcolare rapidamente una nuova radice dell'albero (tree root) dopo operazioni di inserimento, aggiornamento o eliminazione, senza dover ricalcolare l'intero albero.
- Limitare la profondità dell'albero.
- Assicurare che la radice dell'albero dipenda solo dai dati e non dall'ordine in cui vengono applicati gli aggiornamenti.

Modelli di consenso

Un aspetto chiave della tecnologia blockchain è determinare quale utente pubblica il prossimo blocco. Questo problema viene risolto implementando uno dei tanti modelli di consenso possibili.
Per le blockchain permissionless, ci sono generalmente molti nodi che competono contemporaneamente per pubblicare il prossimo blocco.
- Di solito, lo fanno per ottenere criptovaluta.
- Sono utenti che non si fidano l'uno dell'altro e spesso si conoscono solo attraverso i rispettivi indirizzi pubblici.
- Ogni nodo partecipante è motivato dal guadagno economico, piuttosto che dal benessere degli altri nodi o della rete stessa.

Proprietà dei modelli di consenso
- Lo stato iniziale del sistema è concordato da tutti (es. genesis block).
- Gli utenti accettano il modello di consenso utilizzato per aggiungere i blocchi.
- Ogni blocco è collegato al blocco precedente, includendo l'hash dell'intestazione del blocco precedente (tranne il genesis block, che ha un hash predefinito, solitamente composto da zeri).
- Gli utenti possono verificare autonomamente ogni blocco.

Proof of Work (PoW)

I full nodes competono per creare il prossimo blocco stabile della blockchain.
Devono eseguire un calcolo complesso, ma facilmente verificabile.
PoW richiede hardware specializzato (ASIC: Application-Specific Integrated Circuit).

Come funziona PoW
- Calcolare l'hash di alcuni dati, che includono un componente pseudo-casuale.
- Ripetere l'operazione fino a ottenere un valore inferiore o uguale a una certa soglia.
- Il primo nodo che riesce nell'impresa propone il suo blocco per l'aggiunta alla blockchain.
- Ogni blocco confermato dà al nodo vincitore una ricompensa in criptovaluta.
- La soglia viene ricalcolata periodicamente per mantenere il tempo medio di mining quasi costante (alcuni minuti).

PoW in Bitcoin
Bitcoin utilizza SHA-256 per calcolare ripetutamente l'hash di:
- Merkle root
- Intestazione del blocco precedente
- Nonce casuale

Il risultato deve essere inferiore alla soglia di difficoltà stabilita dal protocollo.
Ogni 2016 blocchi, la rete calcola il tempo impiegato per minarli e aggiusta la difficoltà di conseguenza.

PoW in Ethereum 1.0 (fino al 15/09/2022)
Ethereum utilizzava l'algoritmo Ethash.
Il mining combinava e hashava (con Keccak-256) un nonce casuale, un hash dell'intestazione del blocco e sezioni casuali del dataset.
L'operazione si ripeteva fino a trovare un valore conforme alla difficoltà.

Attacchi e limiti di PoW
- Il miner che trova il valore corretto ottiene la ricompensa e diffonde il blocco agli altri nodi.
- PoW è energivoro e ha un elevato impatto ambientale.
- Attacco del 51%: un attore con più del 50% della potenza di calcolo potrebbe riscrivere la cronologia delle transazioni e impedire nuove conferme.
- Fork temporanei: se due miner producono un blocco quasi simultaneamente, la blockchain si biforca. Il problema si risolve quando uno dei due rami diventa più lungo, che viene scelto come valido dai peer siccome è più complesso ricrearlo, a patto che i blocchi siano validi.

Proof of Stake (PoS)
La blockchain mantiene la proprietà di un insieme di coin messi in stake dai partecipanti.
Per aggiungere un nuovo blocco, un partecipante viene eletto casualmente.
La probabilità di essere scelto è proporzionale alla quantità di coin in stake.
Vantaggi:
- Minor rischio di centralizzazione rispetto a PoW.
- Maggior efficienza energetica (nessun mining ad alto consumo).

Problemi:
- Come selezionare il partecipante? È necessaria una fonte di casualità sicura.
- Attacco "Nothing at Stake": i validatori potrebbero creare blocchi su più catene concorrenti senza costi aggiuntivi.

Per risolvere il problema della casualità sicura si potrebbe usare il fatto blockchain già contiene dati casuali (es. hash dei blocchi).
Un'idea ovvia sarebbe hashare l'intera blockchain, ma questo metodo è vulnerabile a un attacco di grinding: un attaccante potrebbe provare a creare più blocchi in segreto e scegliere quello che lo farà essere selezionato nuovamente.

Protocollo PoS con consenso eventuale
Questi protocolli seguono la regola della catena più lunga. L'immutabilità di un blocco aumenta gradualmente con il numero di blocchi successivi.

Protocollo PoS con accordo bizantino (Blockwise-BA)
Qui, l'immutabilità di ogni blocco è garantita da un protocollo di Byzantine Agreement (BA) eseguito prima della creazione del blocco successivo. Usato in Algorand.

Tolleranza ai guasti bizantini (Byzantine Fault Tolerance - BFT)
La Tolleranza ai Guasti Bizantini (BFT) è un meccanismo che consente a un sistema distribuito di funzionare correttamente anche in presenza di nodi maliziosi o difettosi che inviano informazioni contraddittorie.

Problema dei Generali Bizantini (BGP):
- Un gruppo di generali deve prendere una decisione comune (es. attaccare o ritirarsi).
- Alcuni generali potrebbero essere traditori, cercando di confondere gli altri.
- Obiettivo: i generali fedeli devono accordarsi su una decisione, evitando che i traditori influenzino il risultato.
- Le condizioni per risolvere il problema sono: tutti i generali fedeli devono prendere la stessa decisione e se il comandante è fedele, tutti i generali fedeli devono seguire il suo ordine.

La blockchain utilizza protocolli di **Byzantine Agreement (BA)** per garantire che tutti i nodi raggiungano un consenso anche in presenza di attori malevoli.
Algoritmi come Algorand e Casper implementano versioni ottimizzate di BFT per garantire sicurezza e scalabilità.

BBA - Protocollo Byzantine Agreement semplificato
- I partecipanti comunicano in rounds su una rete sincrona e completa, con messaggi inviati e ricevuti entro un round.
- Ogni giocatore possiede una chiave pubblica.
- Tutti conoscono una stringa casuale R sufficientemente lunga (es. 256 bit).
- I giocatori onesti seguono il protocollo.
- Obiettivo: tutti i giocatori onesti devono raggiungere lo stesso output $out \in V$.

L'avversario:
- È un algoritmo a tempo polinomiale che conosce tutti i giocatori, le chiavi pubbliche e R.
- Può corrompere giocatori onesti trasformandoli in maliziosi.
- È un t-Avversario se corrompe esattamente t giocatori.
- Non può interferire con i messaggi inviati da giocatori onesti nel round in cui vengono inviati.
- Non può falsificare firme digitali.

Algorand

Algorand è stato fondato da Silvio Micali e ha lanciato il Testnet il 20 luglio 2018.

- Ogni nuovo blocco viene generato tramite il protocollo BA*, che è la versione estesa del BBA* per valori arbitrari.
- I giocatori del protocollo BA* sono scelti come un sottoinsieme più piccolo di tutti gli utenti.
- Ogni giocatore propone un blocco di transazioni e, alla fine, viene scelto un solo blocco valido.
- Ospita anche smart contracts.
- Utilizza un protocollo, la cui correttezza è dimostrata matematicamente.

Ethereum dopo “The Merge” 2022

- I validatori possono gestire nodi di validazione mettendo in stake 32 ETH o inviando criptovaluta in un wallet di staking.
- L'algoritmo sceglie casualmente chi dovrà creare un blocco e chi dovrà verificare e confermare le transazioni di un blocco specifico.
- Il meccanismo di selezione casuale favorisce chi possiede più ETH.
- I validatori propongono blocchi, che vengono poi attestati da altri validatori.

Problema "Nothing at Stake"

Si verifica quando un utente non ha nulla da perdere nel comportarsi in modo scorretto, ma può ottenere grandi vantaggi.
Nel PoS, i validatori competono per proporre il blocco successivo; la selezione del leader dipende dalla quantità di stake.
Una fork della blockchain può verificarsi nei seguenti casi:

- Attacco malevolo.
- Selezione simultanea di due validatori vincitori.

Strategia ottimale per i validatori: competere su entrambe le catene per massimizzare i profitti.
Un avversario potrebbe forzare intenzionalmente una fork per eseguire un attacco di double-spending.

Attacchi a lungo raggio (Long-range attacks)

Concetto di "costless simulation": possibilità di creare nuove biforcazioni della blockchain senza alcuno sforzo.
Un piccolo gruppo di stakeholder potrebbe ricreare la blockchain dall'inizio e produrre una storia alternativa valida.
Tipologie di attacchi a lungo raggio

- Posterior corruption: L’attaccante ruba le chiavi private di account con un basso stake. La nuova blockchain alternativa potrebbe apparire indistinguibile da quella originale.
- Stake-bleeding: L’attaccante usa le commissioni delle transazioni come ricompensa per far funzionare un fork basato su PoS.

Per mitigare questi attacchi:
- Checkpoints: solo gli ultimi K blocchi possono essere riorganizzati.
- Crittografia a chiavi evolutive.
- Statistiche sulla densità della catena per individuare fork sospette.

Blockchain e funzioni hash di Merkle

La struttura della blockchain è simile allo schema Merkle per le funzioni hash, dove una funzione di compressione viene iterata su un messaggio processato in blocchi.
Le funzioni hash di Merkle devono soddisfare tre proprietà crittografiche:
- Preimage resistance
- Second preimage resistance
- Collision resistance

Attacchi di Eclissi (Eclipse Attacks)

Gli attacchi di eclissi avvengono a livello di rete peer-to-peer.
Un nodo attaccato ottiene una visione distorta della blockchain perché l'attaccante controlla i canali di comunicazione e filtra i messaggi scambiati con altri nodi.

Modelli di consenso alternativi

- **Round Robin**: i nodi si alternano nella creazione dei blocchi, con un timeout per evitare blocchi inattivi. Basso consumo energetico, ma senza puzzle crittografici. (Usato in blockchain permissioned).
- **Proof of Authority / Proof of Identity**: solo nodi con identità verificate possono pubblicare blocchi. La reputazione influisce sulla probabilità di essere scelti. (Applicabile solo a blockchain permissioned.)
- **Proof of Elapsed Time (PoET)**: ogni nodo attende un tempo casuale generato da hardware sicuro prima di pubblicare un blocco. Richiede un ambiente di esecuzione fidato (TEE) e blockchain permissioned.