È un protocollo applicativo orientato a sistemi iper-mediali (non limitato ad essi), ovvero per lo scambio di dati tipizzati. È un protocollo senza stato e generale, supporta tipizzazione e la negoziazione della rappresentazione dei dati.

Alla base c'è il paradigma *client-server* la comunicazione avviene per mezzo di un protocollo di trasporto affidabile --> TCP

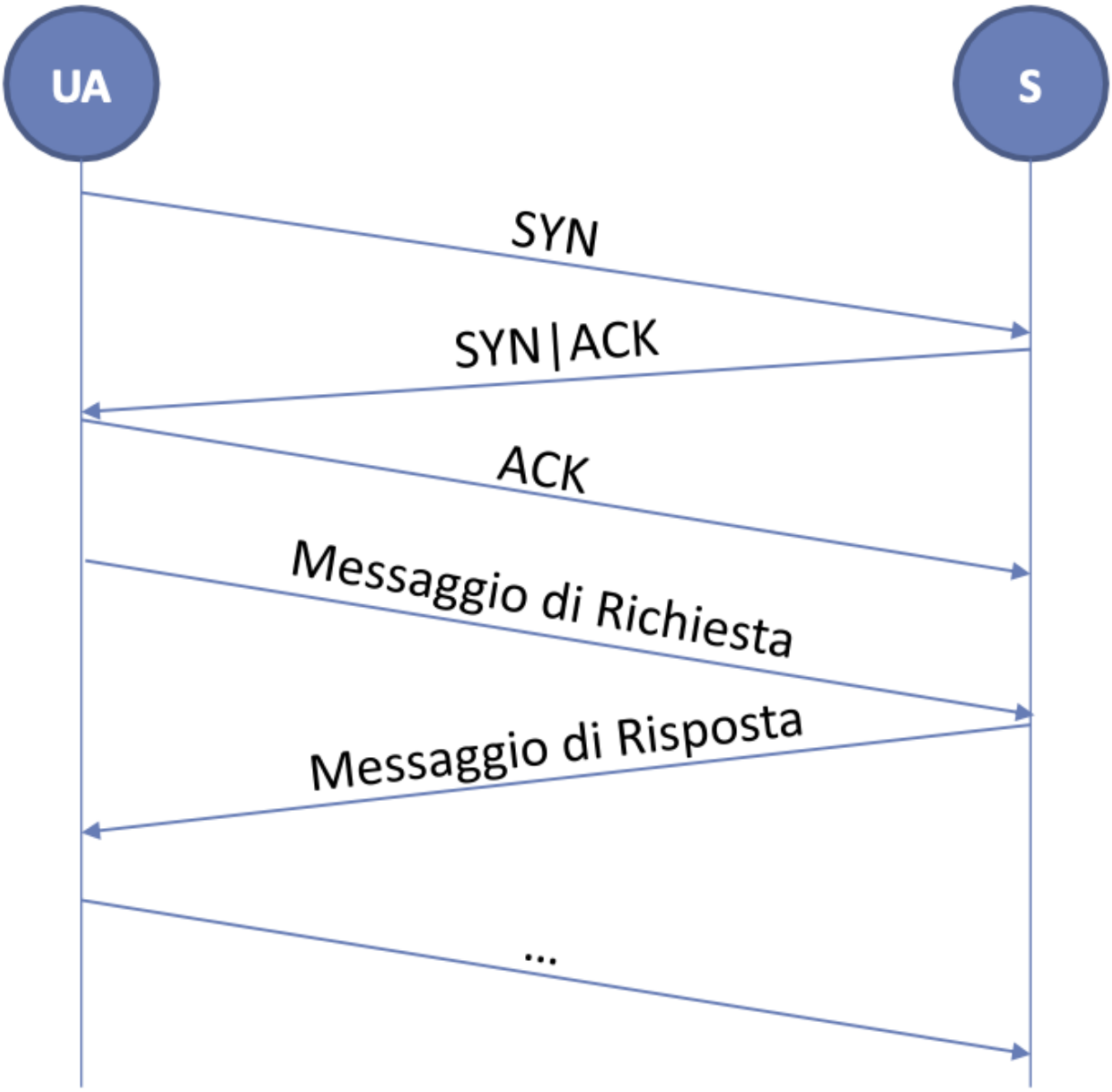
Schema di comunicazione

- Il server è un programma che accetta connessione e attende richieste HTTP: con il termine **Orign Server** si indica il programma che è in grado di generare la risposta autoritativa per una determinata risorsa.
- Il client è un programma che stabilisce una connessione verso un server con l'obiettivo di inviare una o più richieste HTTP, anche chiamato **User Agent**

L'obiettivo della richiesta HTTP è chiamato **Risorsa**. HTTP non si interessa della tipologia di risorsa, si limita ad offrire una interfaccia per l'interazione. Ogni risorsa è identificata da un `URI` (standard).

Le risorse sono eterogenee, ma c'è bisogno di avere un'astrazione per la loro gestione. Il server si occupa di fornire una (o anche più) rappresentazioni della stessa risorsa.

Essendoci sotto il TCP è sempre presente il Three way Handshake



> Alcune volte il protocollo HTTP implementa la logica di *Request-Response*, il serve può mandare dati solo se il client prima fa una richiesta.

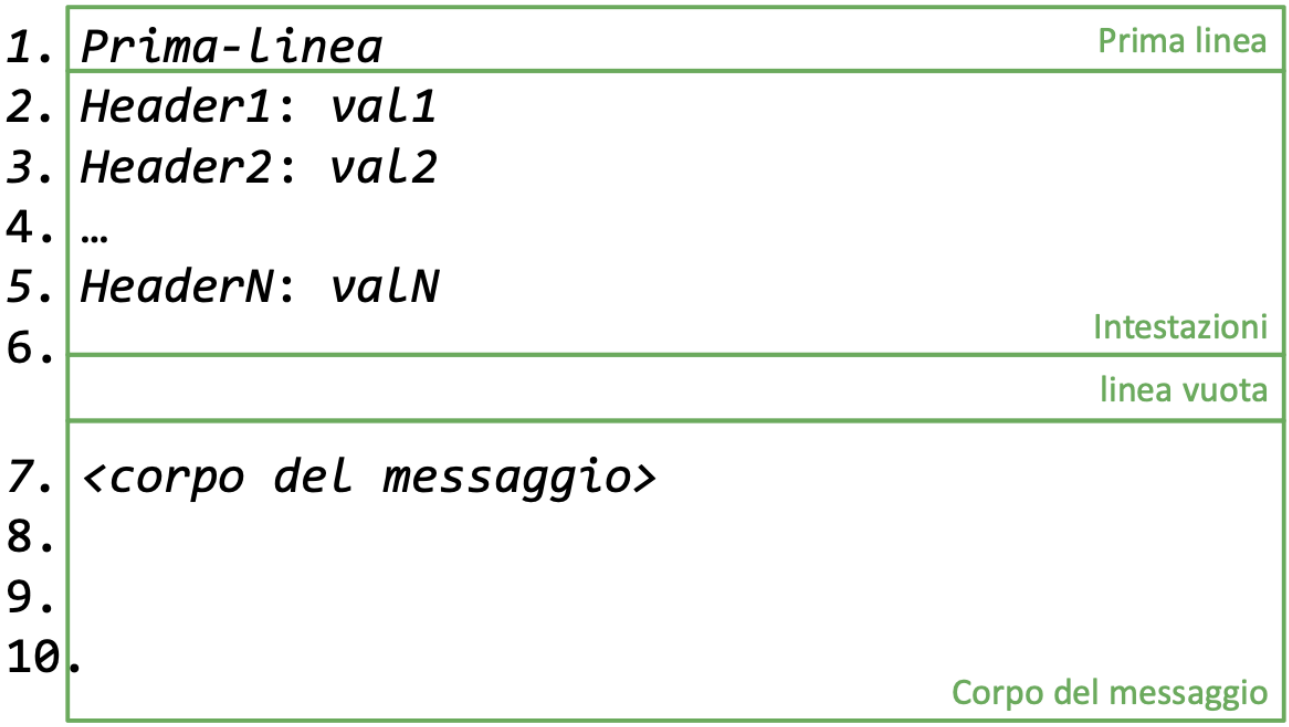
Composizione dei messaggi

Tutti i messaggi sono composti da una sezione di **Intestazione** e da un **Payload**. È un protocollo testuale, quindi può essere letto da un utente umano.

Intestazione

È organizzata per linee:

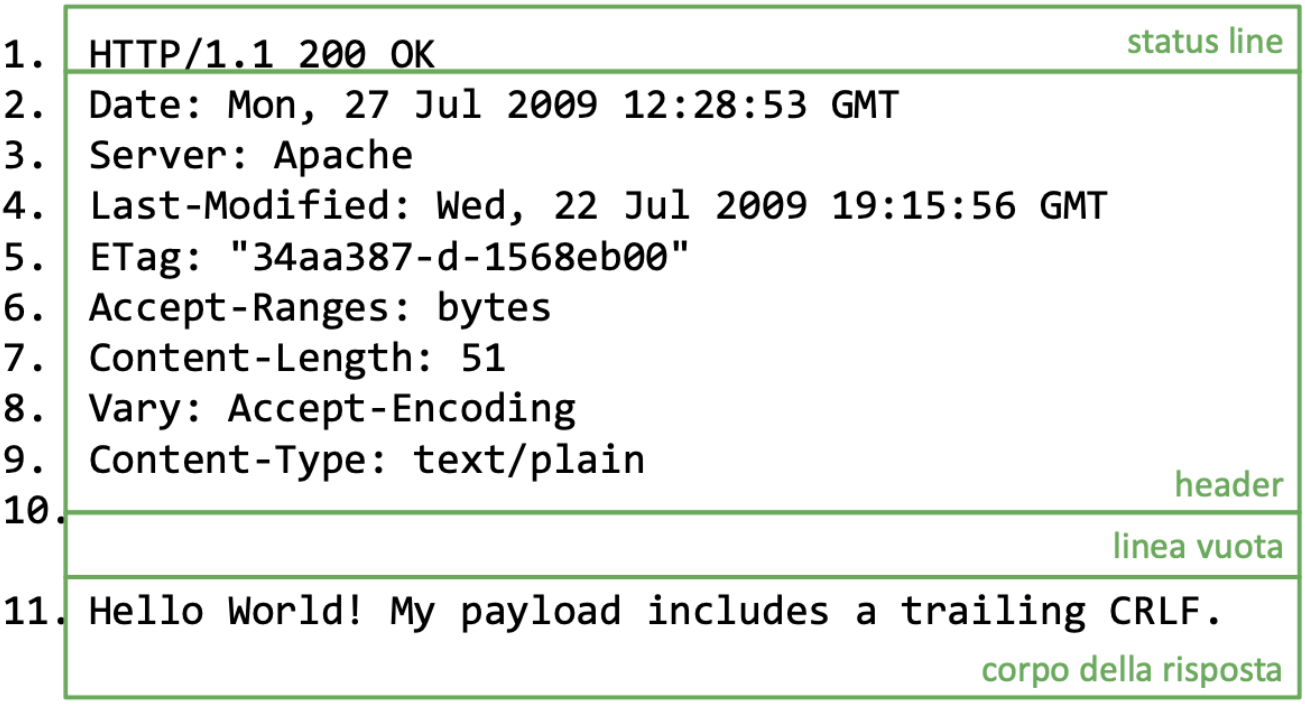
1. La prima indica l'operazione desiderata (richiesta) o il risultato ottenuto (Risposta)
2. Successivamente sono riportati campi aggiuntivi detti **header**
3. una linea vuota separa la sezione di intestazione dal payload del messaggio



Richiesta

1. Il messaggio di richiesta inizia con una **request-line**:
   - contiene **metodo**, **URI** della risorsa, e la **versione** del protocollo
2. Seguono un numero variabile di campi **header**, uno per linea:
   - modificatori, informazioni del client, metadati relativi alla rappresentazione, ...
3. Segue una linea vuota che indica la fine dell'intestazione della richiesta, se è presente sarà riportato il **contenuto** della richiesta.

Risposta



1. La risposta inizia con una **status-line** --> contiene la **versione** del protocollo, un **codice di ritorno** e una **ragione testuale**
2. Seguono un numero variabile di header, uno per linea
3. Segue il corpo della risposta

Per garantire un certo grado di compatibilità futura, ogni messaggio contiene la versione del protocollo nella versione `HTTP/X.Y`:

- si utilizza la major version (X) massima supportata dal server e la minor (Y) massima supportata dal client

Status Code

È una sorta di codice gerarchico a 3 cifre:

1. categoria
2. sottocategoria
3. sottocategoria Lo standard definisce che bisogna gestire almeno le 5 macro categorie presenti:

- `1xx` --> richiesta ricevuta, processo ancora in corso (per un processo lento)
- `2xx` --> richiesta ricevuta, interpretata e accettata
  - `200` --> OK
- `3xx` --> richiedono al client di fare altre azioni:
  - `301` --> *moved permanently* la risorsa è presente in un'altra posizione
  - `304` --> *not modify* la risorsa è già presente in cache
- `4xx` --> errore lato client
  - `400` -->*Bad Request* la richiesta è stata fatta male
  - `401/403` --> risorsa non accettabile
  - `404` --> *not found* richiesta di una risorsa non esistente
- `5xx` -->errore lato server

URI

Gli URI sono usati per identificare le risorse:

- specificare il target delle richieste
- Reindirizzamenti
- Definire relazioni tra entità Definisce una sintassi uniforme per riferimenti a risorse di qualsiasi tipo;
- Estendibile: permette l'introduzione di nuovi tipi di indentificatori
- Uniforme: Gli stessi identificatori possono essere usati in contesti diversi Una risorsa è qualsiasi cosa possa essere identificata un un URI. La sintassi è organizzata gerarchicamente, con componenti elencati in ordine dal più al meno significativo

Sintassi

scheme ":" hier-part ["?" query] ["#" fragment]

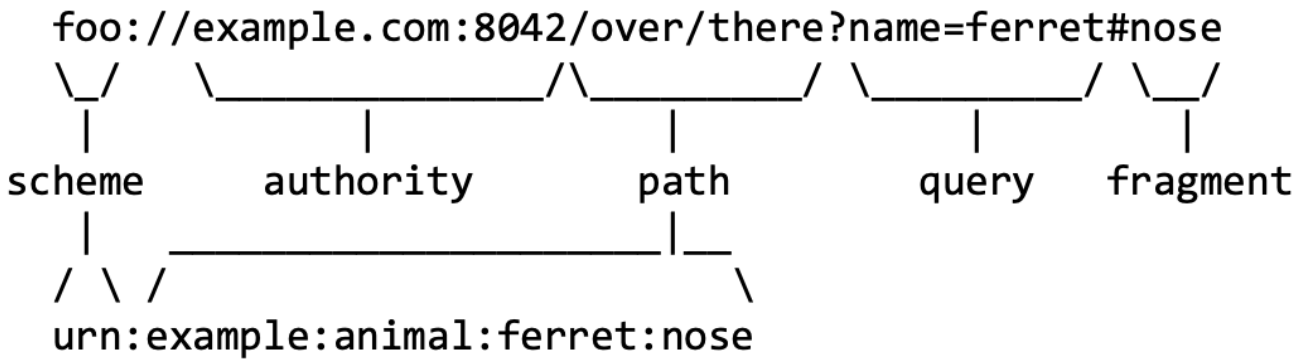
hier-part = authority path

In generale i caratteri che fanno parte del set ASCII possono essere scritti direttamente nella loro forma stampabile, altrimenti è possibile rappresentare i caratteri utilizzando il "percent encoding" (il carattere deve essere rappresentato nella sua forma UTF-8)

- **Scheme** --> si riferisce ad una specifica per assegnare identificatori alle risorse (es. http, https, ftp ...)
- **Hier-part** --> Server authority, è preceduto da `//` e seguito da `/`, `?` o `#` oppure dalla fine dell'URI. Se lo scheme lo prevede definisce l'entità responsabile delegata alla definizione del namespace

  [userinfo "@"] host [":" port]

  l'host può essere:
  - indirizzo IPv6 racchiuso tra `[]`
  - un indirizzo IPv4 in dorma decimale puntata classica
  - Un host identificato in un registry, es DNS il sotto-componente port è opzionale. il **path** contiene informazioni gerarchiche che identificano la risorsa nel sistema di naming adottato, segmenti separati da `/`. questo termina al primo `? o #`
- **Query** --> inizia al primo `?` e termina con l'eventuale `#` o alla fine dell'URI. Contiene informazioni non gerarchiche che identificano la risorsa nel sistema di naming: è una serie di coppie `<chiave>=<valore>` con separatore `&`
- **Fragment** --> frammento che identifica una posizione della richiesta definita precedentemente



URI HTTP

"http:" "//" authority path-abempty ["?" query] ["#" fragment]

- Lo scheme è HTTP ed implica l'utilizzo di TCP
- il campo authority specifica l'origin server per un URI HTTP
- la parte di **path-abempty** è un percorso gerarchico (può essere vuoto)

Metodi HTTP

Get e Head

- `**GET**` serve per richiedere il trasferimento nella rappresentazione corrente della risorsa in oggetto
  - L'identificativo della risorsa non è obbligatoriamente un percorso del filesystem
- **HEAD** come la GET, ma serve per ottenere solo la parte dell'intestazione senza il contenuto

Post

Serve per eseguire una modifica o una gestione specifica sul payload di una richiesta. Ad esempio:

- dati di un modulo da inviare
- Messaggio in un forum di discussione
- Creare una nuova risorsa

Put

Serve per sostiuire tutte le rappresentazioni attuali della risorsa specificata con il payload della richiesta, non implica che una richiesta GET sulla risorsa sia effetuabile

Delete

Serve a rimuovere tutte le rappresentazioni correnti della risorsa

Patch

Si richiede all'applicazione di modificare la risorsa

Solitamente sul BROWSER vengono utilizzate GET e POST. HTTP è un protocollo estendibile --> si possono usare metodi Custom.

Negoziazione del contenuto

- **Proattiva**: il server seleziona la rappresentazione basandosi sulle preferenze espresse dallo user-agent
- **Reattiva**: il server fornisce al client una lista di rappresentazioni tra le quali scegliere
- **Contenuto condizionale**: la rappresentazione consiste di molteplici parti che sono utilizzate in base ai parametri dello user agent
- **Contenuto attivo**: la rappresentazione contiene uno script che farà richieste più specifiche in base alle caratteristiche dello user-agent

Campi di intestazione

Gli Header permettono alle parti di specificare meta-informazioni, sono molto importanti ed influiscono nella gestione della richiesta, nella rappresentazione scelta della risorsa, nel modo di trasferire la risorsa stessa. Ne esistono di standard ma è possibile definirne di nuovi:

- **Host** --> fornisce l'host e la porta relativi alla risorsa richiesta
- **Content-Length** --> definisce la lunghezza in byte del corpo della richiesta
- **Content-Type** --> definisce il tipo (internet media type) del corpo della richiesta
  - Gli internet media type (tipi MIME) specificano il tipo di una risorsa, registrati in un registro IANA
  - MIME --> serve anche per la codifica di binari in testuali per poi essere trasmessi, gestisce la complessità per trasferire i dati che non sarebbero trasmissibili normalmente.
- **Expect** --> Indica un insieme di comportamenti attesi da parte del client
- **Accept** --> lo user agent specifica quali media type sono accettabili
  - Accept-Charset --> indica la preferenza rispetto a un set di caratteri da usare per codificare testo nel contenuto di una risposta
- **Refer** --> Permette allo user agent di specificare dove è stato ottenuto l'URI per la risorsa richiesta

Cookie

HTTP è stateless, ovvero non c'è sato, la connessione viene chiusa alla fine di una riposta (1.0) o dopo un timeout (1.1). Se si vuole avere un meccanismo di stato bisogna utilizzare degli Header aggiuntivi. Queste informazioni di stato saranno poi riportate dallo user agent nelle richieste. Il Cookie rappresenta la metodologia per cui HTTP diventa `statefull`, attraverso l'header **Set-Cookie** in una qualsiasi riposta e alla relativa richiesta successiva con header **Cookie** si può impostare uno stato in una connessione HTTP. Oltre al valore del cookie stesso sono possibili parametri aggiuntivi

che aggiungono limitazioni all'utilizzo dei cookie. Gli attributi vengono memorizzati dallo user agent insieme al cookie stesso.

Questi sono detti token opachi --> ovvero che non vengono interpretati dal client

All'interno del Set-Cookie e del Cookie il token **SID** è quello che permette di identificare la sessione. Altri token presenti sono:

- Expires --> limite do validità del cookie
- Max-Age --> massimo tempo di vita in secondi
- Domain --> identifica il dominio per il quale impostare il cookie Se un server vuole chiudere una connessione manda un Set-Cookie con Expires antecedente o Max-Age minore o uguale a 0

Autenticazione HTTP

Il protocollo HTTP mette a disposizione un framework generico per l'autenticazione. Il protocollo si basa su un meccanismo di challenge-response tra server e client:

- il client prova ad accedere a una risorsa protetta
- il serve chiede le credenziali Nel momento in cui si inseriscono le credenziali si setta un header di nome **`AUTHORIZATION: basic`**. Nel momento in cui il server vede questo header, controlla se sono uguali i campi. Ogni volta che accedo alla risorsa il server si "ricorda" tramite l'header che viene inviato ogni volta. Questo campo è condiviso in BASE64, in verità sono in chiaro a livello HTTP