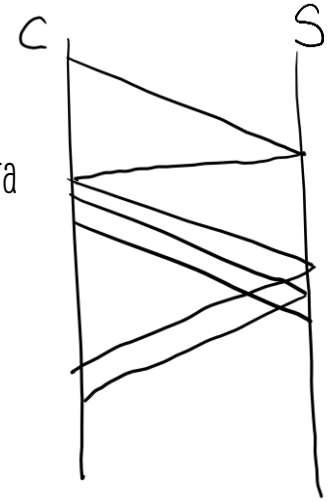
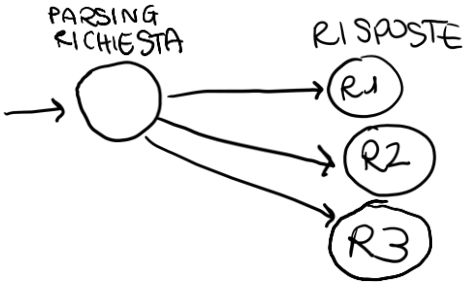
sets 15/11

HTTP 1.1 -> serie più vecchia in uso (RFC 2068-2616)
introduce le reti persistenti, guadagno prestazioni per server lontani dal client
se server e client sono vicini e la latenza è quindi bassa, il guadagno può non essere molto evidente; introdotti diversi miglioramenti
uno di questi è la pipeline:
il client apre una connessione con il server e invia una o più richieste su questa connessione e riceve delle risposte dal server su questa stessa connessione.
anche mantenendo connessioni persistenti senza l'uso della pipeline succede che il client manda la richiesta e aspetta la risposta del server; poi successivamente può inviare una seconda richiesta
PIPELINE -> il client non aspetta di ricevere una risposta prima di inviarne un'altra, se ha più risorse da richiedere le richiede senza aspettare risposte



pipeline è facile da implementare se il server usa un singolo thread per mandare le risposte -> legge e da una risposta per volta



il server legge quindi la prima richiesta e invia la prima risposta, legge poi la seconda richiesta e manderà la seconda risposta ecc.
il tempo necessario per produrre le risposte dipende dalle richieste: ci metterò poco tempo se il file è corto, potrei avere un ritardo se il file è troppo grande.
come possiamo velocizzare il funzionamento del server? attraverso questi modi:

- ottimizzare il parsing delle richieste
- implementare un meccanismo di caching per risposte frequenti
- limitare il numero massimo di thread attivi per evitare il sovraccarico
- migliorare il tempo di esecuzione delle operazioni di I/O

se i client sono più di uno, per esempio 2. avrò due thread che faranno il parsing delle richieste e poi questi lanceranno i vari thread di risposta. se pongo un limite massimo di thread attivi l'efficienza dipenderà dal numero di client e di capacità del sistema.

HTTP 2.0 (RFC 7540) -> 2015
Lo scopo di questa versione è quello di rimuovere la sincronizzazione che c'è nella pipeline.
Usa il protocollo TCP per scambiare i dati. Divide la stringa in due se è troppo lunga, e ne manda prima un pezzo e poi un altro. È un tentativo non riuscito di riprogettare il protocollo TCP basandolo su un'idea di trasporto di tipo datagram (secondo il prof).

HTTP 3.0 (RFC 9114) -> 2022
Vuole abbandonare l'uso del trasporto TCP a favore del trasporto UDP
Usare l'UDP come protocollo di trasporto permette di evitare il 3-way handshake
Lo svantaggio di UDP è quello di non essere affidabile -> l'affidabilità deve essere recuperata in qualche modo

Il protocollo TCP prevede un meccanismo di controllo di flusso basato sulla conoscenza da parte del mittente della dimensione del buffer di ricezione. Se il mittente manda tanti dati, fa in modo di limitare la quantità di dati per non superare la dimensione del buffer di ricezione ed evitare la perdita di dati
Il meccanismo di controllo di congestione rimedia alle perdite dovute alla insufficiente dimensione dei buffer, ma senza sapere la dimensione dei buffer. Riconosce la congestione dopo che è avvenuta
UDP non ha questi controlli, quindi non rallenta e i messaggi arrivano velocemente