DOMANDE RETI

1. crittografia moderna
2. come usare la chiave pubblica per cifrare un documento con la firma digitale
3. web caching (chiesta 2 volte)
4. Voice over ip
5. RPC remote procedure call
6. cos'è IDL, se io ho un server scritto in c e un client in c, posso definire l'interfaccia della procedura in c o devo per forza utilizzare un IDL? Non posso scriverla in c, int c del server e del client potrebbero avere un diverso formato di rappresentazione (esempio uno a 32 bit l'altro a 64), quindi devo usare IDL
7. semantiche di chiamata
8. ftp attivo e passivo. Problemi della violazione del layering? FTP in generale
9. risoluzione mail exchange?
10. posta elettronica (chiesta 2 volte)
11. Cos'è MIME
12. CDN
13. gestione stato tra client e server. Gestione stato www (chiesto 2 volte)? Cookie
14. architettura Client/Server del web, pagine statiche, dinamiche, attive, tre tier
15. *architettura distribuita vantaggi*
16. *eterogeneità protocolli testuali*
17. architetture di firewall
18. servizi da mettere nelle zone smilitarizzate (chiesta 2 volte)
19. Argomento a piacere (ha scelto firewall), (il ragazzo ha preso 23 quindi non solo a genta che ha preso un voto alto viene chiesto l'argomento a piacere)
20. Cos'è AJAX
21. Server stateful e stateless
22. Cos'è una funziona hash
23. Nella delega del DNS, ovvero dei nomi organizzati a livello gerarchico, fornisce dei vantaggi?
24. UTF-8
25. Semantiche RPC
26. Casi in cui si usa la comunicazione indiretta (quella con la mailbox)
27. Int del c potrebbe essere a 32 bit e 64 bit su macchine diverse
28. JSON e XML
29. Protocollo testuale: Possono esserci problemi di validazione con utf-8, in quali casi potremmo avere questo problema
30. Dinamicità pagine web
31. Cos'è il tempo di propagazione DNS (= tempo di aggiornamento delle informazioni quando vengono modificate, generalmente è di 24h)
32. Condivisione del tempo in una rete di calcolatori
33. Cos'è il nat. Modifico solo l'indirizzo IP nel nat?
34. Perché si usano solo TCP e UDP in internet? Cosa sono le middleboxes
35. Come faccio ad usare la crittografia a chiave segreta per cifrare un flusso di dati potenzialmente infinito
36. Naming
37. xmpp
38. Cosa sono telnet e rlogin
39. differenza tra ipv4 e ipv6

40 client server

Risposte:

Sicurezza informatica: crittografia

Crittografia moderna abbiamo 3 tipi:

-**chiave privata (simmetrici)**

-chiave pubblica (asimmetrici)

-chiave omomorfica (in fase di sviluppo)

I sistemi di crittografia sono formati quindi da chiave e algoritmo; dove la sicurezza dei sistemi crittografici è data dalla segretezza della chiave e dalla segretezza dell'algoritmo.

Perciò l'algoritmo può essere noto, ma è importante che venga aggiornato per renderlo il più robusto possibile, finché l'attaccante non conosce la chiave non riesce a privarsene.

L'altro concetto fondamentale della crittografia è la robustezza dell'algoritmo che è determinato dalla robustezza della chiave.

I numeri di bit della chiave, per la chiave pubblica sono circa 2040 bit; mentre per la chiave privata sono 392-256bit.

Nessun sistema è assolutamente sicuro; ma un sistema è detto potenzialmente sicuro se tempo di fare la crittoanalisi cioè lo studio dei metodi per ottenere il significato di informazioni cifrate senza avere accesso all'informazione segreta che è di solito richiesta per effettuare l'operazione.

Quindi se il tempo di fare la crittoanalisi è maggiore del tempo di vita delle informazioni e se il costo della crittoanalisi è maggiore al valore dell'informazione.

chiave privata: abbiamo il mittente e il destinatario possono cifrare con la stessa chiave privata; oppure utilizzare due chiavi diverse ma comunque collegate tra di loro.

La crittografia a chiave privata può essere utilizzata per trasferire un messaggio su un canale sicuro; se viene intercettato richiede la condivisione chiave tra mittente e destinatario, quindi non può essere letto;

anche per verificare l'integrità del messaggio: Message Integrity Code (MIC) cioè il checksum cifrato del messaggio;

può essere utilizzato anche per l'autenticazione ma qui abbiamo con la crittografia a chiave privata, abbiamo il problema della condivisione della chiave. Questo perchè la chiave deve essere condivisa tramite un canale sicuro.

L'autenticazione, per verificare che i due soggetti comunichino tra loro due si manderanno tra le due parti messaggi di sfida a vicenda, perchè solo loro due sono capaci di decifrare la chiave privata.

crittografia a **chiave pubblica:** abbiamo una chiave pubblica che è di pubblico dominio
Qui non abbiamo più il problema della condivisione della chiave; la chiave pubblica viene distribuita attraverso questo dominio di chiave pubbliche (PKI- Public Key Infrastructure)
però abbiamo un costo computazionale rispetto alla crittografia simmetrica;
la chiave privata appartiene solo a quell'utente;
l'utente cifra quella chiave pubblica, il destinatario decifra utilizzando quella chiave privata.
Questo tipo di crittografia può essere utilizzata sia per comunicare in canali sicuri, perchè il messaggio viene decifrato con la propria chiave pubblica; si possono immagazzinare i propri dati anche all'interno di un media, perchè le informazioni vengono cifrate con la chiave pubblica; può essere utilizzata anche per l'autenticazione: la chiave privata appartiene solamente all'utente -> il mittente cifra con un messaggio di sfida la chiave pubblica in modo che solo quel utente che ha la chiave pubblica può decifrare il messaggio.
E analogamente per il destinatario, il cui risponde cifrando la propria chiave privata e l'utente che ha inviato il messaggio per prima può (ipotizzare?) la chiave pubblica del destinatario per verificare l'autenticità del messaggio.
Ma può essere utilizzato per la paternità, come per esempio la firma digitale, visto che la chiave privata appartiene solo a quel utente può firmare il messaggio utilizzando la sua chiave privata, e il mittente può verificare la sua autenticità utilizzando la sua chiave pubblica.

La cifratura della chiave pubblica è significativamente più onerosa.
Se lei ha un documento molto grande che vuole firmare digitalmente esiste una soluzione per velocizzare la procedura di firma?
Si, si può utilizzare la funzione di hash-> effettuo una funzione di hash del documento e firmare il documento sulla funzione di hash, appunto per ottimizzare la firma digitale.

cos'è il web caching?
È una strategia che viene implementata sopratutto all'interno del web, così quando un client va a richiedere una particolare risorsa non dovrà andare a dialogare direttamente con il web server, o application server che magari saranno carichi per altre cose; di conseguenza potrà andare a reperire questa risorsa in maniera più veloce.
Abbiamo 4 tipi di caching all'interno della rete:

- il **browser caching**, dove memorizza all'interno della cache le pagine digitate
- **Transparent proxy,** che fa da web cache per l' intera LAN del client; in questo caso vengono memorizzati i contenuti pubblici e questi contenuti vengono inviati al browser in risposta per evitare di fare un'altra richiesta al server.
- **Reverse proxy**, è di fronte al server web che fa da web caching per l'intera application web. In questo caso vengono memorizzate pagine pubbliche dinamiche per evitare il re-rendering di esse, o anche può fare la conversione di pagine pubbliche a statiche per ottimizzare i tempi di

elaborazione e di risposta al progetto web;

cache server, dove vengono memorizzate pagine private dinamiche; in questo caso vengono salvati contenuti privati dove possono accedervi solo le persone autorizzate e quindi per evitare il re-rendering in porzioni di pagine private pubbliche che vengono chieste frequentemente, la generazione può essere onerosa.

Remote procedure call:

si occupa di inserire dei concetti convenienti per lo sviluppatore di applicazioni di rete. In particolare, fornisce dei servizi -> noi possiamo definire dei servizi come delle funzioni che sono implementate lato server e che sono invocabili remotamente. Quindi, io dal client con una sintassi, con una chiamata a funzione, io posso attraverso questo stato di middle, che si chiama **RPC**, questa chiamata lato client viene catturata dal mio supporto RPC, viene inviata questa chiamata al server, esso esegue la funzione che è stata invocata e rimanda indietro al client la risposta che verrà restituita come il valore di ritorno della funzione che ha invocato il client.

Come si realizza RPC

Cosa avviene dietro alle quinte? In realtà ho un Client che come abbiamo visto, chiama la funzione "moltiplica" localmente, e questa funzione moltiplica finge di essere la funzione moltiplica del Server.
E come fa a fingerlo? Lo fa grazie al fatto che nel lato client esiste una componente chiamata **stub** che ha il compito di fingere di essere la procedura server lato client.
Quindi, avrò uno **stub** specifico per ogni funzione che viene implementato nel lato server
Quindi, se ho una funzione moltiplica,dividi,somma,sottrai -> sono 4 funzioni e avrò uno **stub** per ognuna di queste funzioni.

Si ha uno stub lato Client e Server per ogni procedura.
Lo stub ha lo scopo di essere una componente proxy che finge di essere il server. >Gli fornisce al client un'interfaccia, chiamo la funzione moltiplica (dove non sto chiamando la procedura server, ma sto chiamando lo stub del client), lo stub prende i parametri chiamati dal client e ha il compito di impacchettarli in un messaggio di richiesta; di trasmettere al di sopra del RPC run-time, il messaggio di richiesta al server.
Al lato server l'RPC run-time mi darà questo messaggio a uno **stub del server** (esiste per ogni procedura server) ha il compito inverso allo **stub del client**; quello di prendere i messaggi di richiesta, di spacchettarli, trasformare i parametri e passarli alla funzione (procedura server interfaccia) in un formato nativo alla funzione procedura che eseguirà.
La procedura ritornerà un risultato; il risultato lo passerà allo **stub del server**, esso svolgerà la funzione opposta di quella svolta in precedenza, cioè impacchettare il risultato della procedura del messaggio di risposta, intellegibile al client e che verrà inviato attraverso questo meccanismo di RPC run-time al **stub** del client.
In questo caso, lo stub del client effettua lo spacchettamento di questo messaggio di risposta, prende il valore di ritorno che è contenuto, lo converte in un formato nativo e lo restituisce al client.
Questo meccanismo di RPC è basato su un layer di base che permette l'invocazione di funzioni basandosi su scambi di messaggi e delle procedure stub lato Client e Server che si occupano della conversione tra un formato comune e il formato specifico dei messaggi dei parametri valori di ritorno.

che cosa è IDL?

Utilizzato per descrivere le interfacce degli oggetti e i tipi dei parametri, non è un linguaggio di programmazione (non serve per implementare gli oggetti o per realizzare client che accedano agli oggetti).

- Oltre ai tipi di dati primitivi, per cui fornisce già routine di serializzazione e di deserializzazione, XDR permette di gestire tipi di dati complessi
- In XDR, il formato delle strutture dati è definito attraverso un apposito linguaggio IDL (Interface Definition Language) simile al C
- Uso di *IDL compiler* per generare automaticamente partendo dalle procedure di codifica e decodifica dei dati complessi che sono noti al sistema (leggere e scrivere dati complessi)

Sun/ONC RPC, esempio di IDL di una struttura complicata, dove definisco una costante
ho delle strutture dati che si chiamano *namenode* -> è un puntatore namelist alla struttura
IDL compiler mi genererà automaticamente queste funzioni per dati nativi oppure complessi.
è un'implementazione C only, ma nonostante questo ho bisogno di un IDL che non è il C.
L'intel del Server e l'intel del Client sono lo stesso? No perchè uno può essere a 32 e uno a 64 bit.

semantiche di chiamata

Semantica may-be

E' la strategia più semplice. Questa strategia non si interessa nel prendere delle politiche per quanto riguarda i malfunzionamenti. Quindi sto implementando delle chiamate di procedura remota di tipo **may-be**(completamente best effort).
Se la chiamata fallisce e me ne accorgo lato Client perchè scade un time-out, non posso lato Client sapere cos'è successo.
Quando mando la richiesta e non mi arriva il messaggio di risposta non so se procedura è stata eseguita o il Server è crashato, in poche parole non so minimamente niente.
Inoltre, se la chiamata ha successo, la procedura remota potrebbe essere stata eseguita anche più di una volta (Client invia solo una richiesta, ma ci possono essere messaggi duplicati dal servizio di comunicazione sottostante). Ricordiamo che RPC spesso viene utilizzato sopra UDP ed esistono delle possibilità che il pacchetto venga duplicato durante il canale di comunicazione.
Semantica di semplice realizzazione ma solleva problemi di consistenza di stato sul Server, insicurezza sul Client se le procedure sono state invocate o no.

Semantica at-least-once

Se il **Client** stub non riceve risposta scaduto il time-out **riprovo a spedire il messaggio di richiesta** un numero di N volte.
Se la chiamata ha successo, la procedura è stata eseguita una o più volte (richieste e messaggi duplicati), da cui at-least-once (almeno una volta).
Se la chiamata fallisce il cliente può pensare a:

- malfunzionamento permanente di rete

- server crash

Questo tipo di semantica può andare bene nel caso di **procedure idempotenti**(procedure che non hanno side effect) , cioè che possono essere eseguite molte volte con lo stesso effetto sullo stato di interazione C/S.
Esempio, server stateless di NFS.77
Se non ho procedure idempotenti utilizzo un altro tipo di semantica.

Semantica at-most-once

Si assegna degli identificatori a tutte le richieste e una volta che è stata eseguita lato Server, esso si tiene traccia dell'identificativo e se riceve una successivo messaggio con lo stesso identificativo di richiesta lo **cestina/ignora**. Questo perché si evita di eseguire la stessa procedura.
Nel caso il Server ottenesse una nuovo messaggio con la richiesta precedente, scoprirebbe che ha mandato il messaggio di risposta ma il Client non ha ricevuto allora cosa fa?
Il messaggio di risposta che si era salvato in una cash viene rimandato al Client fino a quando non riceve un **acknowledge da parte del Client**.

Questo permette di ottenere una semantica di tipo at-most-once (al massimo una), in quanto è garantito che in ogni caso la procedura remota è:

- non eseguita
- eseguita solo parzialmente (in caso di server crash)
- eseguita una sola volta

Semantica exactly-once
Semantica difficile da realizzare
Il Server potrebbe implementare le procedure come transazioni atomiche, in modo che le procedure siano eseguite del tutto o per niente.

Questo permette di ottenere una semantica di tipo exactly-once (esattamente una), in quanto è garantito che in ogni caso la procedura remota è:

- non eseguita
- eseguita una sola volta

Questa semantica rappresenta il **caso ideale**, le implementazioni di RPC solitamente presentano semantica di tipo at-most-once.

FTP attivo e passivo. Problemi della violazione del layering?
ftp (file transfer protocol) stabilisce un collegamento con una macchina remota per il trasferimento (upload e download) di file. Come per telnet, ci sono problemi di sicurezza, legati alla trasmissione in chiaro delle password. Possibile uso di FTP over SSL o di alternative sicure come scp (trasferimento di file su canale ssh).

- Vari programmi applicativi, da linea di comando o con interfaccia grafica.

• Protocollo di comunicazione TCP
• Protocollo ftp con comandi (4 caratteri ASCII a 7 bit) e risposte (numeri a 3 cifre).
Esempi di comandi protocollo ftp:

- STOR local-file trasferisce un file locale sulla macchina remota
- RETR remote-file trasferisce un file remoto sul disco locale

Utente utilizza varie interfacce (ad es. linea comandi, con put (esegue STOR), get (esegue RETR), mget e mput, help, dir, ls, cd, lcd, … ftp usa due connessioni per ogni collegamento client/server, una di CONTROLLO (su porta 21) e una di DATI (su porta 20). Per questo, in alcuni testi si dice che le informazioni di controllo sono trasmesse fuori out-of-band (ma è situazione diversa da rlogin…).
Il Server mantiene lo stato della sessione del Client.
ftp attivo vs passivo -> chiede spesso all'esame
l'FTP in **modalità attiva** lavora in questo modo:

1. Il client inizia una connessione (generalmente sulla porta 21) verso il server e comunica la porta in cui si pone in ascolto (generalmente una porta random decisa dal client) per lo scambio dei dati;
2. Il server conferma la connessione (nel caso validando le credenziali)
3. Il server apre una connessione verso il client sulla porta comunicata
4. Il client conferma la connessione
5. Inizia la trasmissione dei dati

l'FTP in **modalità passiva** lavora in questo modo:

1. Il client inizia una connessione (generalmente sulla porta 21) verso il server;
2. Il server conferma la connessione (nel caso validando le credenziali) e comunica la porta in cui si pone in ascolto (generalmente una porta random in un preciso range impostato lato server) per lo scambio dei dati;
3. Il client apre una connessione verso il server sulla porta comunicata x la trasmissione dei dati
4. Il server conferma la connessione
5. Inizia il trasferimento dei dati

La differenza tra le due modalità di funzionamento del FTP sta nel fatto che la modalità passiva, quando ben gestita consente l'apertura e la chiusura di porte casuali, riducendo i rischi legati a possibili attacchi. L'utilizzo di un FTP passivo risolve i problemi legati al firewall nei client, ma d'altro canto si dovrebbe lasciare aperte molte porte non privilegiate. Quasi tutti i server FTP consentono di impostare un range di porte in cui il server può mettersi in ascolto.

Problemi di FTP

FTP presenta numerosi problemi, al punto che potrebbe forse essere indicato come esempio di come ***NON*** progettare servizi Internet.

In FTP attivo il Client comunica il proprio indirizzo IPv4 al Server come argomento del comando PORT, un errore di design davvero eclatante. Infatti, scrivere un indirizzo di livello 3 in un protocollo di livello 7

rappresenta una gravissima (!!!) violazione del principio di layering alla base del modello ISO/OSI con serie conseguenze per il protocollo applicativo. In seguito all'arrivo di IPv6, si è dovuto estendere il protocollo con due comandi (EPRT ed EPSV) che sostituissero i comandi PORT e PASV.

Inoltre, la violazione del layering causa problemi nel caso (praticamente certo) in cui il Client si trovi in un ambiente di rete in cui è vengono usati indirizzi IPv4 privati e NAT, poiché il Server riceve dal Client un IP privato che ovviamente non è raggiungibile.

Voice over ip(voIP)

si intende l'insieme dei protocolli di comunicazione di strato applicativo che rendono possibile tale tipo di comunicazione.

VoIP consente una comunicazione audio-video sistema real-time,unicast o multicast, su rete a pacchetto.

risoluzione mail exchange

Il funzionamento del sistema di posta elettronica dipende dal supporto offerto dalla risoluzione di nomi di tipo Mail eXchange (MX) del DNS.

In particolare, attraverso la risoluzione di nomi di tipo MX, il DNS permette a un Server SMTP di scoprire i nomi logici dei Server SMTP di un determinato dominio (ad esempio il dominio di un indirizzo di destinazione di un'e-mail).

Senza il supporto alla risoluzione MX, sarebbe impossibile scoprire quale sia il Server SMTP di un particolare dominio.

Si noti che i Server SMTP di un dominio sono tipicamente replicati. Più precisamente, abbiamo almeno 2 Server: 1 master e 1+ slave.

All'interno del dominio pippo.com ho un utente che ha il proprio user agent pre-configurato dove si installa il proprio user agent, bisogna configurarlo per dirgli che bisogna utilizzare questo SMTP server del proprio provider, e anche IMAP server o POP3 server del proprio provider.

SMTP Server A, id mittente , come fa a sapere chi è l'SMTP Server? Qual'è il server che si occupa di svolgere il ruolo di server SMTP per il dominio paperino.com?

Non lo può sapere, perché l'unica cosa che sa è il dominio in cui è definita la casella elettronica destinataria, @paperino.com.

Non c'è nell'email alcun riferimento al nome del server SMTP a cui consegnare la posta, so solo il nome della casella di posta, ma non so il nome del server in cui questa casella si tiene.

Per smistare questa posta, l'SMTP server del mittente deve scoprire il nome dell'SMTP server del destinatario. Non c'è un default.

Quindi, per scoprire quale è l'SMTP server lo fa attraverso un apposita interrogazione DNS, con risoluzione di tipo MailExchange → attenzione!! Non è come: dal nome host mi ottiene il corrispondente indirizzo ipv4; il mailexchange non parte da un nome di host ma da un nome di dominio. Io mando una richiesta di nome di dominio; mailexchange paperino.com significa mi mandi indietro il nome di tutti i server SMTP, dove ce ne devono essere almeno due: master e un slave, del dominio paperino.com.

Mi arriva indietro il nome, ma prima di utilizzarlo devo fare una seconda risoluzione di tipo A o AAA, oppure entrambe, per avere gli ip corrispondenti alla lista di questi server che mi sono stati restituiti.

Quindi, devo avere 2 tipi di risoluzioni: MX e A/AAA. Solo allora, dopo che ho i nomi e gli indirizzi ip che posso connettermi e trasferire l'email dal mittente al destinatario.

L'email viene messa nel database che è la casella di posta e il destinatario può accedere in maniera completamente asincrono rispetto alla procedura di trasmissione/invio della mail

Posta elettronica

il servizio di posta elettronica si basa su

• **User Agent (UA)**, Client che si interfacciano con l'utente

• **Mail Transfer Agent (MTA)**, i server di posta

E-mail parte da UA mittente e arriva a UA destinazione attraverso la rete dei MTA.

Utilizzo del protocollo di trasferimento **SMTP** (Simple Mail Transfer Protocol) per la consegna della posta alla casella del destinatario (modello push):

• Mail Transfer Agent usano SMTP

• User Agent in trasmissione, quando manda la propria posta manda al proprio server SMTP del proprio provider, in trasmissione utilizza SMTP per inizare il processo rooting della mail connettendosi con il server del proprio Internet service provider e consegnando la mail a questo server SMTP che poi si sconnetterà dall'utente, prenderà in carico l'email e proverà a consegnarla a destinazione per diversi giorni fino a quando non verificherà che tutto è andato bene.

Nel caso non riesce a inviarla a destinazione a un certo punto manderà un messaggio di errore all'utente.

Utilizzo di protocolli specifici, come POP3 o IMAP (o, ancor meglio, delle loro versioni «sicure» POP3s e IMAPS), per la ricezione della posta, ovverosia per il trasferimento della posta dalla casella di posta dell'utente allo UA che ne permette la consultazione (**modello pull**).

Il formato dei messaggi è RFC 822, sono messaggi che sono formati da un header e un body.

Gli header contengono tutte le informazioni classiche di un messaggio di posta, che compongono l'intestazione; *from* (indirizzo mittente), *to* (mailbox destinatario → anche più destinatari) , *date, subject* (soggetto del messaggio) , *cc* (copia destinatari).

I body invece contengono il corpo del messaggio → testo dell'email, in formato ASCII

MIME

Multipurpose Internet Mail Extensions è un protocollo che estende RFC 822 per consentire invio di dati multimediali (audio, immagini, video, documenti word, etc.)

MIME ha 3 parti fondamentali:

1) Righe di intestazione (integrano header di RFC822) per definire il tipo di dato.

2) Specifica il tipo di dato presente nella mail

3) Indica la codifica usata per i vari dati, in modo che possano essere trasmessi in un e-mail (che usa SOLO caratteri ASCII).Eventuali conversioni di formati binari in ASCII (ad esempio, tramite codifica base64).

CDN

Content Delivery Network: serve per distribuire su grande scala il contenuto statico o che cambia raramente. Es. patch per un qualsiasi programma.

Sono funzioni che in automatico da un server centrale distribuiscono su copie localizzate di questo server centrale editati, in cui gli utenti possono accedere, e sono fisicamente vicini a loro; per evitare di scaricare grossi contenuti statici da un unico server centrale, che potrebbe essere anche da una parte del mondo lontano e avere di conseguenza un enorme latenza sul download.

Senza contare che se tutti si connettessero sul server centrale il carico rallenterebbe maggiormente il trasferimento dei dati; invece viene suddiviso su tutti i server locali e l'utente può connettersi a quello più vicino a lui.

È automatica la connessione dell'utente?

Si, è gestito tutto in automatico. L'utente non deve decidere a quale server connettersi.

Mi parli della gestione dello stato tra un client e un server

La gestione dello stato tra un clien e un server può essere di 2 tipi: **stateless** e **statefull**

stateless → lo stato, l'interazione tra client e server non viene salvato sul server; di conseguenza ogni nuova richiesta non dipende dalla precedente

statefull → salva lo stato d'interazione tra client e server; di conseguenza ogni nuova richiesta da parte del client dipende dalla precedente

=> non ce n'è uno buono e uno cattivo, se a seconda dell'applicazione che voglio fare posso usare uno o l'altro.

Nel web questo potrebbe avere dei problemi..?

Il web è basato su HTTP che è **stateless** → se voglio mantenere lo stato d'interazione tra client e server devo mantenerlo a livello applicativo nel mio application server.

E come faccio a gestire questo stato?

Lo faccio con i **cookie** → Esistono soluzioni **client side** o **server side**

La soluzione **client-side** è una delle soluzioni più usate, cioè i cookie. Dove viene creato questo token in base alla connessione, in cui viene fornito insieme ad ogni richiesta HTTP tra client e server per mantenere traccia di sessione tra client e server.

Architettura client-server web

Al giorno d'oggi il web si sta evolvendo e si è passati dalle prime architetture del web client/server a delle architetture più complesse che permettessero la gestione di pagine non solo statiche ma anche di pagine dinamiche e attive.

Nasce così un architettura detta **estesa** proprio perché ci sono delle componenti aggiuntive.

Il web browser che rappresenta il nostri client invia delle richieste a un server web;

server web ha il compito di rispondere a queste richieste attraverso delle pagine statiche , tutti i contenuti statici relativi anche a pagine dinamiche; una pagina dinamica può contenere dei contenuti

statici (es. immagini). I contenuti statici sono all'interno di un file system nel quale un utente può caricare tutti questi contenuti statici attraverso ad esempio il servizio FTP.

Se il server web non riesce a rispondere a questa richiesta del browser perché necessita non solo di una pagina statica, ma bensì di un contenuto dinamico, allora questa richiesta viene inoltrata a un application web che è contenuta all'interno di un application server e fa parte 2 livello che è l'**application tier**.

Il primo tier è quello del server web, chiamato **presentation tier;** il secondo è **presentation tier** e il terzo è il **data tier.**

Il data tier contiene un database la quale l'application web si servirà per prelevare i contenuti necessari alla generazione della pagina dinamica.

Esistono delle interfacce che permettono la condivisione tra un livello e l'altro;

l'interfaccia tra il web browser e il server web è un'interfaccia di tipo HTTP;

l'interfaccia tra il databse e l'application server è di tipo SQL.

Queste due interfacce, sono interfacce fisse/prestabilite che non cambiano al variare dell'architettura.

Mentre l'interfaccia tra il web server e l'application web è un'interfaccia che può variare a seconda dell'architettura stessa dell'application web.

Inizialmente, si utilizzavano le CGI, proprio perché al web inizialmente serviva solo contenuti statici, quindi il server doveva eseguire solamente dei semplici script.

Poi, con l'evoluzione del web si iniziano a creare dei veri e propri framework, ovvero delle piattaforme dove il server deve andare a inserire del codice per generare la pagina.

Quindi, si è preferito passare dalle CGI a un'interfaccia di tipo HTTP. Questo porterebbe a pensare di eliminare o spostare il server web dopo l'application web. Questo però è un errore in quanto: innanziatutto il server web serve per servire contenuti statici nell'immediato, quindi per creare una risposta più velocemnte alla richiesta del browser ; poi l'implementazione del HTTP fatta dal server web è un'implementazione sicura, quindi il server web è stato implementato apposta per garantire questa sicurezza e l'application web invece no.

Questa architettura estesa è stata generata per creare appunto anche delle pagine dinamiche/attive.

Una pagina dinamica a differenza di una statica, non è identica ad ogni richiesta ma viene creata ad ogni momento dal server web in richiesta a una risposta del browser e può cambiare da richiesta a richiesta.

Le pagine attive si differenziano dalle pagine dinamiche, perché non vengono generate del tutto dal server bensì è il browser che si occupa di completare la richiesta; Quindi il compito del server è molto limitato rispetto alla creazione di pagine dinamiche o statiche, e c'è un lavoro più complesso da parte del browser, del client. Le pagine attive sono il soggetto principale del web 2.0, che è l'evoluzione del web tradizionale, dove non si ha più un web composto da siti, ma di servizi;

si ha una distribuzione di contenuti che non vengono più solo forniti dal web verso l'utente ma c'è un'interazione completa fra l'utente e il web stesso.

Quali sono i vantaggi di avere un'architettura distribuita nel DNS? Perchè non abbiamo un database centralizzato ma lo abbiamo distribuito?

Non abbiamo un database centralizzato innanzitutto perché un singolo point of failure, cioè in caso di guasto andreabbe in crash tutto il sistema; se invece ho un sistema decentralizzato in caso di guasto di uno dei punti, il mio sistema può continuare a funzionare.

Inoltre, utilizziamo un sistema di tipo decentralizzato anche per migliorare le prestazioni, quindi servizio e richieste in modo più veloce.

Il DNS si basa su una struttura di tipo gerarchica nel quale abbiamo domini di alto livello e da questi dipenderanno dei successivi domini in modo appunto gerarchico, ma amministivamente indipendenti.

Dal punto di vista della gestione amministrativa invece?

Sono indipendenti tra loro però c'è un concetto di delega. I domini di più alto livello delegano i domini di sotto livello.

Quali sono i vantaggi di un UTF-8 contro UTF-16 o un UTF-32?

Utf-8 è un encoding completamente compatibile con la codifica ASCII, proprio perché i primi caratteri dell'encoding corrispondono ai primi caratteri del codice ASCII. Questo permette una retrocompatibilità con il passato che era il punto fondamentale il quale si voleva arrivare per permettere un funzionamento più eterogeneo. Utf-8 utilizza un encoding a lunghezza variabile, come utf-16.

Mentre utf-32 utilizza un encoding a 4 byte fissi.

Utf-8 è lo standard principale per xml e json, e viene utilizzato per rendere i sistemi distribuiti eterogenei, infatti nel caso in cui abbiamo un sistema locale che utilizza un altro tipo di codifica viene effettuata la transcodifica, ovvero viene convertita la codifica locale in codifica utf-8, e viceversa dall'altra parte della comunicazione, rendendo la comunicazione eterogenea.

Utf-16 invece, è lo standard di Java e Windows

Utf-32, non viene più tanto utilizzato proprio perché ha una struttura complessa e non è compatibile e retro-compatibile con le altre codifiche.

Che cosa è utf8?

Utf-8 è un protocollo testuale che rappresenta l'evoluzione del utf-16 e utf-32
In utf-8 usa gruppi di byte per rappresentare i caratteri unicode, ed è particolarmente utile per il trasferimento tramite la posta elettronica a 8 bit.

Utf-8 è di lunghezza variabile per natura; associa a ciascun carattere una sequenza di byre di lunghezza variabile che va da 1 a 4.

Dato che utf-8 usa solo byte per rappresentare i caratteri ASCII, riesco a utilizzare stringhe null-terminated

I vantaggi e gli svantaggi dell'utf-8 sono:

vantaggi:

- Il vantaggio più ovvio di qualsiasi codifica UTF è che permette di rappresentare tutti i caratteri, a differenza di codifiche più vecchie.

- Alcuni caratteri di Unicode (per esempio l'alfabeto latino) occupano in UTF-8 un solo byte, altri richiedono fino a quattro byte.
- Una sequenza di byte che codifica un carattere non può apparire come parte di una sequenza più lunga che codifica un altro carattere, come succedeva per codifiche a lunghezza variabile meno recenti
- Il primo byte di una sequenza è sufficiente a determinarne la lunghezza (è sufficiente contare il numero di bit più significativi con valore uno). Questo rende molto semplice estrarre una sotto-stringa da una stringa più lunga, senza bisogno di decodificare la sequenza di byte UTF-8.

- La maggior parte dei software esistenti (inclusi i sistemi operativi) sono stati scritti senza tener conto di Unicode, perché l'uso di questo andrebbe a creare problemi di compatibilità. Per esempio la libreria standard del C marca la fine di una stringa con un byte nullo (0x00). Il primo byte verrebbe trattato come il marcatore di fine stringa, e il secondo insieme a quelli successivi verrebbero ignorati.
Quindi, UTF-8 è pensato in modo che nessuno dei byte codificati possa assumere uno dei valori speciali del codice ASCII, evitando questi problemi e simili.
•UTF-8 è la codifica predefinita per il formato XML.

Svantaggi:

- UTF-8 usa sequenze di lunghezza variabile, cioè singoli caratteri possono venire rappresentati con sequenze di byte di lunghezze diverse.

Le alternative: UTF-32 e UTF-16

UTF-32 utilizza sempre sequenze di numeri a 32 bit, ovvero 4 byte. La semplicità della sua struttura migliora la leggibilità del formato. Nelle lingue che utilizzano principalmente l'alfabeto latino e quindi solo i primi 128 caratteri, questa codifica occupa molta più memoria del necessario (4 byte invece di 1).

UTF-16 si è affermato come formato di visualizzazione in sistemi operativi come Apple macOS e Microsoft Windows e viene utilizzato anche in molti framework di sviluppo software. È una delle codifiche UTF più vecchie ancora in uso. La sua struttura è particolarmente adatta per la codifica di caratteri in lingua non latina, perché occupa poco spazio in memoria. La maggior parte dei caratteri può essere rappresentata con 2 byte (16 bit). Solo in caso di caratteri rari la lunghezza può raddoppiare fino a 4 byte.

Quali sono le architetture dei principali firewall?

Abbiamo 3 tipologie di firewall principali:

Packet filtering firewall

Stateful inspection firewall

Application proxy firewall (o application-level gateway)

Questi 3 tipi di firewall, poi possono essere applicati nell'architettura generando diversi tipi di architetture firewall.

La prima architettura è: **screening router firewall**, nella quale abbiamo una router principalmente che attraverso un **package filtering firewall**, filtra tutti i pacchetti che dall'esterno vogliano arrivare alla rete interna, e viceversa. Questa di solito è una soluzione applicata per piccole reti, come quella di casa.

single bastion è un'architettura dove si ha un firewall posto tra la rete interna e la rete esterna, ed è appunto una macchina che permette la divisione tra la rete esterna e quella interna garantendo la sicurezza tra le due parti

double bastion permette di creare delle zone smilitarizzate all'interno dei quali abbiamo dei dispositivi sicuri in quanto sono protetti da dei firewall; tra questi dispositivi possiamo trovare anche la application proxy firewall che a differenza degli altri permette di filtrare un determinato servizio

distributed firewall nel quale abbiamo due zone smilitarizzate, du cui una smilitarizzata interna e una esterna. Per quella esterna avremo applicazioni e servizi accessibili dall'esterno, mentre per quella interna avremo applicazioni e servizi accessibili dalla rete interna (intranet).

Di solito questa soluzione è resa gestibile da strumenti di configurazione sofisticati che automatizzano la gestione dei vari firewall installati nella rete.

Argomento a piacere: firewall

Il firewall è un dispositivo hardware/software di difesa perimetrale di una rete, e svolge funzioni di collegamento tra due o più segmenti di rete, fornendo una protezione della rete stessa.

Un firewall è un sistema costituito da molti componenti che:

- rappresenta l'unico punto di contatto della rete interna con l'esterna
- filtra e controlla tutto il traffico tra le due reti
- concentra i meccanismi di sicurezza
- impone la politica di sicurezza della organizzazione
- nasconde informazioni della rete interna
- registra eventi (logging) ed elabora statistiche sul traffico di rete (auditing)

Quindi, sarebbe una protezione della rete interna per non farla vedere in quella esterna.

Quindi, si occultano dei dati che possono essere sensibili della rete interna a quella esterna che è generalmente poco sicura.

Quella interna, essendo generalmente reti LAN, sono spesso più sicure e non hanno problemi.

Questo meccanismo non è perfetto, essendo una porta/muro, non protegge se facciamo entrare un qualcosa di malevolo all'interno; non potegge da virus, di accessi secondari.

Ed essendo l'unico punto di snodo può facilmente presentare il codidetto collo di bottiglia o dare un sacco di problemi se l'architettura del sistema non è compatibile o comunque perfettamente compatibile con la progettazione del firewall.

Il firewall per essere progettato ha alcuni requisiti da soddisfare, come per esempio:

- requisiti di autenticazione e/o di autorizzazione

- requisiti di essere uniforme al sistema in cui deve essere inserito/progettato

- bisogna fare anche un elenco dei servizi a cui posso far accedere la mia rete, o viceversa posso aprire al patch a cui voglio fare accesso senza che il firewall dia dei problemi.

I requisiti di autenticazione rispondono fondamentalmente a 3 frasi che sono: qualcosa che ho, qualcosa che sono e qualcosa che possiedo.

Qualcosa che so risponde alla maggior parte dei casi a una password o a un pin, o comunque a una nozione che io conosco

Qualcosa che ho si riferisce principalmente ad un oggetto fisico che io devo utilizzare: es. smart key, smart card..

Qualcosa che sono indica proprio qualcosa che possiedo io come persona, quindi un qualsiasi dato biometrico → scansione della retina, impronta digitale …

Per quanto riguarda i requisiti di autorizzazione possiamo basarci su due concetti totalmente opposti, ma che danno un threadoff su quello che può essere il livello di sicurezza o il livello di facilità di gestione;

ovviamente scegliando uno dei due l'altro cala tantissimo.

Nel primo caso, avendo un principio molto più sicuro ma molto più difficile da gestire, per tutte le eventuali eccezioni che possono esserci, è quello di vietare tutto ciò che non è espressamente richiesto.

Si può avere facilmente un problema, perché ad esempio se ci sono servizi che sfruttano piccole differenze sottili l'uno dall'altro, non essendo quello specifico permesso che è stato dato, è molto probabile che il firewall lo blocchi e non gli dà il permesso di passare/continuare.

D'altro canto, si può utilizzare la tecnica inversa, quindi: permettere tutto ciò che non è espressamente vietato → ma qui essendo più facile da gestire, il problema ricade sulla sicurezza. Piccole eccezioni malevoli che possono entrare nella rete interna, può entrare perché era stato vietato una precisa condizione.

Una volta scelta il tipo di autenticazione e autorizzazione, bisogna scegliere il tipo di firewall.

Esistono 3 tipi di firewall, e agiscono in 3 livelli differenti.

1) **packet filtering** che agisce a livello rete → analizza ogni pacchetto che lo attraversa singolarmente, senza tenere conto dei pacchetti che lo hanno preceduto. In tale analisi vengono considerate solo alcune informazioni contenute nell'header del pacchetto, in particolare l'indirizzo IP della sorgente, l'indirizzo IP della destinazione, la porta della sorgente, la porta della destinazione e il protocollo di trasporto. Su questi parametri vengono costruite le regole che formalizzano la policy del firewall e che stabiliscono quali pacchetti lasciar passare e quali bloccare.Questo tipo di filtraggio è semplice e leggero ma non garantisce un'elevata sicurezza. Infatti risulta vulnerabile ad attacchi di tipo IP spoofing in quanto non riesce a distinguere se un pacchetto appartenga o no ad una connessione attiva

2) **Statefull inspection firewall** tiene traccia delle connessioni TCP aperte e controlla che il traffico in entrata o in uscita, appartenga a quella connessione TCP specifica che abbiamo aperto;

Questo tipo di livello svolge lo stesso tipo di filtraggio dei packet filter firewall e in più tiene traccia delle connessioni e del loro stato, quindi abbiamo una maggiore protezione rispetto al packet filtering firewall

3) **Application proxy firewall** agisce a livello applicazione, filtra tutto il traffico di una singola applicazione sulla base della conoscenza del suo protocollo.

Quindi, può filtrare servizi di applicazioni specfici; Questo tipo di firewall è in grado di rilevare i tentativi di intrusione attraverso lo sfruttamento di un protocollo e di realizzare le funzionalità di logging e reporting in modo migliore rispetto ai firewall precedenti.

Quindi risolve i problemi che erano stati introdotti dai firewall precedenti.

Una volta che si è deciso il firewall che si vuole utilizzare, bisogna decidere anche l'architettura con cui comporlo e la sua eventuale posizione, a seconda delle eseginze:

Personal firewall, per proteggere le comunicazioni di un singolo utente, sui computer di essi.

Hosted-based firewall per proteggere le comunicazioni di una singola macchina, su ciascun server.

Bastion host macchina sicura dedicata al controllo del sistema e del suo traffico, e questo è iil punto più sicuro della rete che tipicamente ospita strumenti sofisticati come per esempio **application-level-gateway.**

Quest'ultima è la soluzione più indicata nel caso si abbia dati estremamente sensibili e una mole di dati molto grande che vogliamo proteggere.

Per il primo posizionamento, possiamo utilizzare lo **screening router** che è un'architettura firewall, dove protegge ambienti di dimensioni piccole come per esempio un ambiente di un ufficio..

single bastion è un'architettura dove si ha un firewall posto tra la rete interna e la rete esterna, ed è appunto una macchina che permette la divisione tra la rete esterna e quella interna garantendo la sicurezza tra le due parti

double bastion permette di creare delle zone smilitarizzate all'interno dei quali abbiamo dei dispositivi sicuri in quanto sono protetti da dei firewall; tra questi dispositivi possiamo trovare anche la application proxy firewall che a differenza degli altri permette di filtrare un determinato servizio

distributed firewall nel quale abbiamo due zone smilitarizzate, du cui una smilitarizzata interna e una esterna. Per quella esterna avremo applicazioni e servizi accessibili dall'esterno, mentre per quella interna avremo applicazioni e servizi accessibili dalla rete interna (intranet).

Di solito questa soluzione è resa gestibile da strumenti di configurazione sofisticati che automatizzano la gestione dei vari firewall installati nella rete.

Che cosa è la zona demilitarizzata?

è una sottorete fisica o logica che contiene ed espone dei servizi ad una rete esterna non ritenuta sicura, come ad esempio Internet. Lo scopo di una zona demilitarizzata è di proteggere la rete LAN di un'organizzazione.

I server Web potrebbero dover comunicare con un database interno per fornire alcuni servizi specializzati. Quindi bisognerebbe configurare una zona militarizzata altamente monitorata che comprende per lo più server Web e server simili che si interfacciano con il mondo esterno, cioè Internet, in maniera tale da proteggere i server di database ponendoli in una zona separata dalla zona demilitarizzata dato che è pubblicamente accessibile. Quindi si può essere utilizzato un firewall dell'applicazione che funge da mezzo di comunicazione tra il server del database e il server web, così da fornire un'ulteriore livello di sicurezza, anche se è più complesso. Questo tipo di configurazione ha però uno svantaggio dovuto al fatto che gli utenti della workstation non possono usare directory di rete e apportare modifiche direttamente a pagine Web o altri file e quindi costretti ad usare protocolli applicativi come FTP per caricare o scaricare le pagine.

Che cosa è FTP?

ftp (file transfer protocol) stabilisce un collegamento con una macchina remota per il trasferimento (upload e download) di file.Si tratta di un protocollo basato sull'architettura client/server, ed è un protocollo di un servizio a livello applicativo il quale si basa sulla connessione TCP. Inoltre, si basa su due comandi specifici: **storage** e **retrive.**

La **storage** serve per allocare in una macchina remota un file che abbiamo su disco locale

La **retrive** serve per prelevare dalla macchina remota per poi inserirlo sul disco locale.

Si basa sull'utilizzo su due canali di connessione:

- il canale **comandi o controllo** in cui vengono date tutte le informazioni apposta per instaurare una connessione per le azioni da svolgere (porta 21 della macchina server)

- la connessione di tipo **dati** in cui vengono trasferiti i dati, avvengono generalmente sulla porta 20, ma spesso non è così perché l'implementazione FTP può essere fatta in 2 modi diversi.

Innanzitutto bisogna specificare che il servizio ftp adotta un processo master che si mette in attesa di ricezioni di connessioni chiamate **FTP Demon**, cioè un processo demone. Genera un processo **slave** per ogni connessione che si collega.

In base al sistema in cui opera, può esserci o un singolo processo che collega entrambe le connessioni (quella di controllo e quella di dati), oppure esiste un processo singolo per ogni connessione.

L'implementazione di FTP può essere di 2 tipi:

FTP attivo → la macchina client si connette alla porta controllo della macchina server, e gli fornisca il proprio indirizzo IP insieme alla porta su cui si deve connettere. In modalità attiva, il client FTP si connette alla porta 21 del server FTP da una porta non privilegiata a caso;

La porta di comando del client contatta la porta di comando del server e fornisce la sua porta dati.

Il server fornisce un riconoscimento alla porta di comando del client.

Il server stabilisce una connessione tra la sua porta dati e la porta dati del client.

Alla fine, il client invia una conferma al server.

L'FTP attivo deve essere utilizzato quando il server FTP, che sta tentando di connettersi, non supporta connessioni FTP passive o se il server FTP si trova dietro un firewall / router / dispositivo NAT.

FTP passivo → La modalità FTP passiva è stata sviluppata per risolvere i problemi di connessione della modalità attiva. Questa è la comunicazione tra il client FTP e il server in modalità passiva.

• Il client contatta la porta di comando del server e invia il comando PASV per indicare che questa è una connessione passiva.

• Quindi il server dà la sua porta dati di ascolto al client.

• Quindi il client effettua una connessione dati tra il server e se stessa utilizzando la porta specificata. (la porta è data dal server)

• Alla fine, il server invia una conferma al client.

L'FTP passivo dovrebbe essere utilizzato sempre a meno che non si sia verificato un errore o se la connessione FTP utilizza porte FTP non standard.

Qual'è la differenza tra FTP attivo o passivo?

1) La modalità attiva offre maggiore sicurezza al server FTP. Ma in modalità passiva no. La modalità passiva viene utilizzata quando le connessioni FTP sono bloccate dai firewall.

2) L'FTP attivo potrebbe causare problemi a causa dei firewall. Ma l'FTP passivo non ha problemi di connessione dei firewall.

3) In modalità attiva, il client stabilisce il canale di comando e il server stabilisce il canale dati, ma nell'FTP passivo, entrambe le connessioni vengono stabilite dal client.

4) La maggior parte della modalità predefinita del browser web è passiva. La modalità attiva non viene utilizzata come modalità predefinita di un browser.

Che cosa è AJAX?

Ajax è l'abbreviazione di Asynchronous JavaScript and XML;

È un servizio per creare pagine attive interattive, basandosi su uno scambio di dati in background fra web browser e server,consentendo così l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

Che cosa è una funzione hash?

Sono delle funzioni che servono per garantire una maggiore sicurezza nei documenti;

sono delle particolari funzioni che prendono in input un messaggio di una qualsiasi lunghezza, arbitraria, e restituiscono un output di lunghezza fissa, molto più piccola rispetto a quella originaria.

Una particolarità delle funzioni hash è quella di eseguire questo calcolo in maniera velocissima e dato un messaggio in input risulterà: se il messaggio è sempre lo stesso il medesimo hash, ma se cambia solo una piccolissima parte come ad esempio una virgola, l'hash del messaggio sarà notevolmente diverso.

Dall'hash è quasi impossibile risalire al messaggio vero e proprio.

L'hash viene utilizzato in molti ambiti:

- si può garantire l'integrità del documento, perché potrei hashare direttamente il documento e inviarlo; ed ecco che quando il ricevente lo riceverà potrà tranquillamente hasharlo di nuovo e vedere se i due hash coincidono

- possono essere utilizzati anche per garantire una maggiore sicurezza; per esempio se vado a salvare dei dati sensibili su un supporto che non viene considerato sicuro → basti pensare alle password, quando ci logghiamo la password non viene condivisa direttamente dal server perché potrebbe essere troppo insicuro, ma si effettuerà l'hash e il server controllerà che i due hash coincidono, con quello salvato nel database e quello che è stato appena inviato.

Naming, DNS e Name Server

Dato che la rete è molto vasta, bisogna utilizzare una procedura di **naming**, cioè associare un nome logico a un server, un indirizzo che corrisponde alla locazione fisica all'interno di un server e una route che serve per localizzare la risorsa

Indirizzo fisico è l'indirizzo IP

da un nome reperisco il sistema, ma c'è anche il percorso per arrivarci

Il nome specifica quale oggetto (entità) si riferisce, denota l'entità:

• nome logico a livello utente

• identificatore come nome (interno) definito dal sistema

L'indirizzo specifica dove l'oggetto risiede.

La route specifica come raggiungere l'oggetto.

Questa associazione avviene tramite una funzione di corrispindenza, che risolve il problema di associare

il nome logico alla risorsa fisica.
la traduzione da nome a indirizzo è come un hashtable, mapping; Esistono due mappe fondamentali;
la mappa fra indirizzi e percorsi mi viene già data da IP
è una mappa che devo consultare nella risoluzione del nome
=> dal nome ottengo l'indirizzo IP, e da esso riesco a utilizzare la risorsa
Binding è l' associazione tra un nome e una risorsa
Quindi, quando invochiamo una funzione (oriented programing, classi derivate ecc) e si ha un oggetto/riferimento a un oggetto come esempio l'interfaccia, si chiama un metodo e esso viene risolto al momento della chiamata, e si va a vedere se è associato alla classe base
quindi l'associazione tra nome e funzione si chiama **binding**
Due tipologie di binding:
binding statico può non essere ideale perchè mi associa a tutto il tempo di vita dell'applicazione → è il binding che viene effettuato una sola volta
binding dinamico → viene effettuato ogni volta per aggiornare le risorse, e questo mi dà la possibilità di conoscere quali sono le risorse che sono disponibili e quali no.
In costanti dinamici si utilizza un binding dinamici, ha maggiori performance.

A livello ideale, il binding dinamico bisognerebbe farlo ogni volta, ma questo non accade perché si possono avere problemi computazionali, cioè si avrebbe troppa elaborazione e quindi viene effettuato a intervalli di tempo.
Per riuscere a mantenere in memoria questi binding c'è il bisogno di utilizzare un database → ma utilizzare un databse unico e centralizzato non è per ulla efficiente e si è rischio di **single point failure,** quindi non posso accedere più alle mie risorse, c'è una difficoltà di consultazione del mio databse e il problema più grosso è quello di malfunzionamenti => quindi il problema più grosso è la manutenzione

La soluzione a questo è l'utilizzo del **DNS (**Domain Name System), prevede un sistema decentralizzato dei nomi, infatti viene utilizzato per assegnare nomi ai nodi della rete(host).

Il servizio è realizzato attraverso un databse distribuito, costruito dai server DNS.

La radice nominata ci sono solo le informazioni dei top level domains, sono cosiddetti livelli di 1 livello che sono gestiti da grandi organizzazione
Ognuno dei domini e sotto domini ha almeno 2 server che hanno tutte le informazioni che appartengono a quel dominio; in ogni nodo del grafo ci sono le sue informazioni, che sono le informazioni dei domini sottostanti, nomi e ip dei server dei sottodomini.
al terzo livello ci sono le organizzazioni vere e proprie
Ogni dominio può essere **relativo** o **assoluto**
Ogni dominio deve fare riferimento al dominio che lo contiene. Esiste il **concetto di delega**: un dominio delega i sottodomini a gestori sottostanti (che se ne assumono responsabilità e autorità).
es. Se il dns di facebook non funzionava era loro autorità sistemarlo.
All'interno del DNS abbiamo diversi **record type,** come per esempio:
- **A** per la risoluzione da nome host a indirizzo ipv4
- **AAAA** per la risoluzione da nome host a indirizzo ipv6

- **MX** per risoluzione da nome dominio a nome del server di posta elettronica.

Il DNS che è un protocollo di trasposrto, usa un semplice protocollo richiesta-risposta basato su UDP.
Viene scelto UDP per diverse ragioni, le quali sono:
- non è necessarria l'affidabilità > questo perché il protocollo è talmente semplice che è facile realizzare una forma di affidabilità a livello applicativo
- non è necessario la **congestion control** → questo perché il protocollo occupa un traffico talmente limitato che è molto improbabile che esso diventi una causa di congestioni
- è importante minimizzare la latenza nella risoluzione dei nomi → quindi il setup di una connessione TCP rallenterebbe eccessivamente le procedure di risoluzione dei nomi.

Il concetto di organizzazione gerarchica e di delega, fornisce dei vantaggi da un punto di vista amministrativo?

Si. I DNS implementano uno spazio gerarchico dei nomi, per permettere che parti di uno spazio dei nomi, conosciute come "zone", possano essere delegate da un **name server** ad un altro name server che si trova più in basso nella gerarchia.

Una "zona" DNS è una parte dello spazio dei nomi, costituita da un dominio e i suoi sottodomini che non sono a loro volta delegati, che è sotto una stessa gestione amministrativa e quindi è gestita da uno o più server. Per ragioni di ridondanza, ciascuna zona è "replicata" su più *server*, e di conseguenza la delega è costituita da più r*ecord name server*, che indicano che ciascuno dei server indicati contiene le informazioni per quella zona.

Architettura delle applicazioni distibuite → modello client/server

Le applicazioni distribuite prevedono un'interazione tra due elementi: mittente e destinatario.

Rispetto a questo tipo di comunicazione, ci sono due caratteristiche che rappresentano diverse tipologie che sono: la designazione dei processi e la sincronizzazione dei processi che hanno tra di loro.

Per quanto riguarda la designazione dei processi, si può parlare di **schemi diretti simmetrici, schemi diretti assimetrici e schemi indiretti**.

I **schemi diretti simmetrici** prevedono che si nomino esplicitamente la vicenda e di conseguenza nelle send e receive vengono nominate esplicitamente anche rispettivamente il mittente e il destinatario.

Nei **schemi diretti assimetrici** invece il mittente nomina esplicitamente il destinatario, ma il destinatario non specifica da quali mittenti riceverà la comunicazione.

Quindi tiene aperta una linea di comunicazione generica su cui alla porta si agganceranno i vari mittenti che richiedono il servizio.

I **schemi indiretti** prevedono che la ciìomunicazione non avvenga direttamente tra il mittente e il destinatario, ma ci sia un soggetto intermedio che si chiama **mailbox** che fa da intermediario nella comunicazione. Quindi il mittente non richiede la comunicazione direttamente al destinatario ma deposita la sua richiesta all'interno della mailbox e il destinatario non comunica direttamente con il mittente ma verifica periodicamente se nella mailbox si sono depositate richieste.

E questo avviene anche nel caso delle risposte → il destinatario deposita l'esito dell'operazione direttamente nella mailbox dove verrà verificata dal cliente.

Secondo lei, questo tipo di comunicazione tramite mailbox in che occasioni potrebbe essere utile?

Può essere utile per motivi di sicurezza, in cui non è opportuno tenere aperta una comunicazione diretta oppure anche nel caso di comunicazioni asincrone in cui è opportuno che il mittente o il destinatario o entrambi, continuino a fare le loro operazioni, quindi eseguire altri processi e non siano costantemente in attesa dell'interazione con l'altra parte.

Poi a livello di sincronizzazione due processi possono comunicare in modalità asincrona o sincrona.

La modalità **sincrona** prevede che i processi siano bloccanti, ovvero quando il mittente manda una richiesta al destinatario, il mittente rimnga bloccato finché la richiesta non ha avuto una conclusione che gli è stata rimandata, cioè di successo o fallimento

Nella modalità **asincrona** invece il mittente e il destinatario continuano ad eseguire le loro operazioni e non rimangono bloccati in attesa dell'operazione.

La modalità asincrona è più semplice da realizzare a livello di programmazione, però presenta dei problemi per quanto riguarda la consistenza del server, perché appunto non si può avere la garanzia che la comunicazione sia andata a buon fine, soltanto guardando lo stato del mittente e del destinatario.

Nel caso in cui ci siano server occupati con un numero elevato di operazioni che devono essere eseguite, la programmazione si complica ancora di più in quanto bisogna introdurre un **buffer**.

Un buffer è un array che prevede la memorizzazione sequenziale delle richieste che vengono memorizzate a livello temporaneo. Poi vengono inoltrate al destinatario per essere eseguite.

Infine, le applicazioni distribuite possono essere **con o senza connessione**.

Nel caso siano **con connessione** la connessione viene stabilita fin da subito tra mittente e destinatario.

Nel caso invece siano **senza connessione**, la programmazione diventa più onerosa a livello di dati (pacchetti di dati), in quanto ogni pacchetto di informazione deve contenere oltre al hoveread anche l'indirizzo del destinatario. (es. sistema postale)

Poi, la comunicazione può essere **affidabile** o **non affidabile.**

Affidabile → Nel caso in cui il supporto garantisca la consegna dei messaggi. In caso di perdita di messaggi o guasti, il supporto tipicamente ritrasmette (anche più volte) il messaggio.

Ovviamente non può risolvere guasti permanenti (per esempio di rete). Alto costo di realizzazione (basse prestazioni) Vs. semplicità di programmazione.

Comunicazione Non Affidabile → In questo caso il supporto invia i messaggi senza verificarne la consegna. Alte prestazioni Vs. difficoltà di programmazione.
Un esempio di queste due comunicazioni è la differenza tra TCP e UDP.

Per quanto riguarda invece il modello client/server → è un'evoluzione di questo modello generico, dove il client fa da mittente, quindi è la macchina che chiede al server di eseguire una richiesta.

Invece il server è una macchina centralizzata che si occupa di eseguire dei compiti specifici.

Quindi lo schema di comunicazione è **molti a uno** → questo perché vari client possono comunicare con lo stesso server.

La programmazione del server è molto più complicata rispetto a quella del client perché deve poter gestire molte connessioni; e sopratutto il server deve essere avviato all'avvio della macchina.

La comunicazione tra client e server deve avvenire o attraverso il **modello pull** o attraverso il **modello**

push.

• **Modello pull**. Il Client richiede l'informazione al Server e in genere si blocca e aspetta la risposta (simile a chiamata di procedura). Modello più utilizzato. Quindi, di conseguenza è una modalità sincrona

• **Modello push**. Il Client segnala il proprio interesse e poi fa altro. È compito del Server di inviare l'informazione se e quando disponibile. Il Server effettua l'invio (push) delle informazioni(applicazioni della posta). → modello asincrono.
Il modello push può anche essere emulato da un'operazione di **polling** non bloccante lato Client.
Succede che il Client ogni tot secondi richiede al Server servizio fino all'ottenimento della risposta. Il modello pull è Server-driven e sincrono (bloccante) mentre il modello polling è Client-driven e asincrono (non bloccante) e il modello push è Server-driven e asincrono.
Per quanto riguarda le richieste multiple di servizio l'interazione tra client e server può essere **statefull o stateless.**

Cosa sono JSON e XML?
Sono due linguaggi;
json viene utilizzato per archiviare le informazioni in modo organizzato e di facile accesso. La sua forma completa è JavaScript Object Notation, ed offre una raccolta di dati leggibili dall'uomo a cui è possibile accedere logicamente.
JSON
● JSON è un formato di rappresentazione dati particolarmente leggero e molto utilizzato sul Web
    - Molto più compatto e significativamente più performante e facile da processare rispetto a XML
● Pensato per applicazioni Web 2.0 e per la manipolazione dei dati in JavaScript
● Compatibile con hash in linguaggio JavaScript
Xml è un linguaggio di marcatura progettato per memorizzare i dati. È comunemente usato per il trsferimento di dati.
XML è un linguaggio di descrizione specializzabile per settori specifici (ovverosia un metalinguaggio)
- Standardizzato da W3C, è molto utilizzato anche al di fuori del Web
- Tecnologia sviluppata in ottica machine-oriented, per facilitare la generazione automatica di codice che valida e/o manipola tipi di dati strutturati rigorosamente definiti
- XML è usato sia per la rappresentazione di dati e messaggi scambiati che per la definizione del loro formato

Protocollo testuale: Possono esserci problemi di validazione con utf-8, in quali casi potremmo avere questo problema
Attenzione! Poiché in UTF-8 i caratteri hanno dimensione variabile, si deve fare particolare attenzione a verificare che un buffer di memoria non contenga dei caratteri incompleti prima di utilizzarlo!
=> quindi prima li devo validare

- Con una write da un file o da una comunicazione tra processi (IPC- Inter process communication) potrei leggere in un buffer solo una parte di una stringa UTF-8, che non contiene tutti i byte che

codificano l'ultimo carattere ricevuto!

● Attenzione! Oltre a verificare che un buffer non contenga caratteri incompleti bisogna verificare che i dati rappresentati siano effettivamente validi!

=> devo passare per una libreria di validazione

Il C non si è allineato a UTF-8; supporta delle codifiche multibyte

Transcodifica

– In Java si usano le funzioni di transcodifica fornite da InputStreamReader e OutputStreamWriter (il secondo parametro del costruttore specifica la codifica "esterna").

=> classi fondamentali per la codifica

– In Unix/C lo standard è rappresentato dalla funzione *iconv* fornita dalle librerie di sistema, che però è piuttosto difficile da utilizzare.

=> si consiglia di utilizzare librerie più semplici come:

- utf8proc
- libunistring -> più semplice
- glib

Ulteriore modo per inizializzare il buffer a 0:

char buffer[..] = {0} -> preferibile questo ma memset si può utilizzare quando si vuole

Che cosa è NAT?

NAT è una soluzione che trasforma automaticamente indirizzi privati in indirizzi globali. Ed è l'unica cosa che fa andare in piedi l'internet IPv4.

Ha la possibilità di connettere a Internet reti con un numero elevato di indirizzi privati utilizzando un numero relativamente basso di indirizzi pubblici.

Il suo funzionamento è che ho una rete all'interno della quale uso degli indirizzi privati, e nel router a confine di questa rete, il gateway adsl di casa nostra ad esempio, ho un processo NAT che è in esecuzione sul mio router. Esso sa che c'è una rete interna ed esterna, su quella interna ci sono degli indirizzi privati che non possono comunicare con la rete esterna. Quindi il router connesso alla rete esterna ha un indirizzo IP pubblico e modifica tutti i pacchetti che arrivano da indirizzi IP privati verso l'esterno e modifica l'indirizzo sorgente e porta, fingendo di aver inviato lui i pacchetti. Dunque mette il proprio indirizzo IP e un numero di porta finta sopra a questi pacchetti, e quando gli ritorneranno indietro prenderà l'indirizzo di destinazione, lo modificherà e lo manderà all'interno.

Svantaggi:

-- Problemi nel caso si voglia rendere accessibili all'esterno servizi che girano nella rete con indirizzi privati

– Necessità di configurazioni ad hoc per server

– Uso di protocolli per consentire al traffico esterno di passare attraverso il NAT (STUN, TURN, ICE, ecc.)

-- Problemi con alcuni tipi di servizi. Ad esempio, per abilitare FTP attivo è necessario un NAT steteful con uno specifico modulo di protocol translation

Nella Internet moderna, esistono molti strumenti, denominati “middleboxes”, che, come NAT, manipolano il traffico per vari motivi e possono rappresentare un problema per le applicazioni. Cioè questi middleboxes assumono che possono passare solo traffico UDP e TCP, non potendo utilizzare altri protocolli diversi come SCTP

Dinamicità pagine web

Le prime pagine apparse sul Web erano scritte in HTML ed erano statiche e immutabili.
Che cosa succedeva? Praticamente c’era un web master che utilizzava un editor HTML per scrivere una pagina in HTML.
Il web master generava/gestiva sulla propria workstation locale generava il proprio sito, esportava l’HTML e lo caricava sul web server. E il web server prendeva l’HTML generati e li caricava tipicamente attraverso FTTP.
Quindi, si utilizzava FTTP per caricare questi documenti HTML generati da from page; e ogni volta che cambiava qualcosa bisognava rigenerare gli HTML e ricaricarli sul web attraverso FTTP.
Quindi, sul web server si trovavano semplicemente delle **pagine statiche,** che erano rappresentati da dei file sul disco fisso, sul file system sul server. Il server era assolutamente semplicissimo: prendeva una richiesta e le richieste che si potevano avere erano “mandami un file HTML o un file css”
Una volta effettuato il parsing della richiesta, e una volta determinato il percorso all’interno del file system del file HTML che deve servire; servire il file HTML è una riga di codice -> esiste una system call (non è standard, ma i più grandi so la usano) che si chiama **sendfile** che prende tutto il contenuto del file e lo sbatte su una socket.
Evoluzione del Web verso una maggiore dinamicità.
Per i siti web in cui si hanno delle pagine che cambiano velocemente non si possono utilizzare pagine statiche ma pagine dinamiche
Tipi di pagine:
Pagine statiche (fanno parte del web1.0) Sono documenti Web il cui contenuto è definito al momento della loro creazione e non cambia più. Qualunque richiesta di pagina statica fornisce sempre la stessa risposta.
Pagine dinamiche. Sono documenti Web che vengono creati dal Web Server solo quando un browser li richiede. In tale caso il Server Web attiva un’applicazione che crea il documento in modo dinamico e lo invia al browser. Il contenuto di un documento dinamico può variare di richiesta in richiesta.

Un terzo livello di complessità/dinamicità:
Pagine attive (fa parte del web2.0) Una pagina attiva ha un contenuto che NON è completamente specificato dal Server. È un pezzo di programma che viene mandato dal Server al browser, dove viene eseguito localmente. Una pagina attiva può interagire con l’utente e con altre risorse ed entità, per cui il suo contenuto non è mai statico.

pagine statiche:

vantaggi: semplicità (della pagina,server) che si riflette anche in performance (la facilità di effettuare il cache di queste informazioni); e affidabilità.

- Estrema performance perchè le pagine possono essere servite lato server (una system call) oppure attraverso una cache
- affidabilità, perchè un minimo di componenti è essenziale -> quindi server semplicissimo, browser semplicissimo => massima affidabilità

svantaggi: sono pagine poco flessibili, per ogni cambiamento bisogna modificare
il web master deve cambiarle in locale e caricarle sul server

pagine dinamiche:

vantaggi: sono adatte per informazioni con elevata frequenza di cambiamento (es. siti di e-commerce o CMS= Content Manage System -> sw che sono pensati per la realizzazione di siti come unife.it o corriere.it; dove io ho l'utente che legge l'informazione, ma c'è un'interfaccia di amministrazione o editing, dove io posso entrare con username e password in un portale di unife e inserire eventuali informazioni).

Il vantaggio è che nonostante queste pagine dinamiche siano date per creare un sacco di applicazioni, per il browser sono come delle pagine statiche; quindi non ho bisogno di un browser sofisticato.

svantaggi: ho una computazione sul server che richiede dei server potenti;
ho delle prestazioni che sono più basse rispetto alle pagine statiche, perchè ho un tempo di elaborazione e in molti casi non posso effettuare il caching delle pagine dinamiche;
a differenza delle pagine statiche che hanno bisogno di un web master per caricare le pagine, qui ho bisogno di un programmatore con delle buone capacità perchè realizzare delle applicazioni web basate sulle pagine dinamiche è molto difficile, non in quanto realizzare delle applicazioni web basate sulle pagine attive (rappresenta una complessità di 2 ordine superiori) ma comunque non è una passeggiata, c'è bisogno di realizzare delle applicazioni web performanti su pagine dinamiche.
Lo svantaggio principale delle applicazioni con pagine dinamiche è che un'interazione tra l'utente e la pagina è limitata -> è del tipo "clicca e aspetta"; quindi l'idea sarebbe quella di generare le pagine in fretta, perchè questo "clicca e aspetta" distrugge la user experience

pagine attive:

vantaggi: *l'interazione con l'utente è continua* = by design -> quindi non ho questa spiacevole esperienza che la pagina si frizza;
ho un minor carico sul server perchè la computazione non è tutta sul server ma è principalmente sul client -> si effettua una flooding della computazione dal server al client;
Inoltre, le pagine attive sono pagine molto complesse ma esistono modelli du programmazione innovativi e interessanti (es. web components, declarative..)
svantaggi: c'è bisogno di browser più sofisticati e di dispositivi client più potenti;
ma questo è un finto bisogno.. perchè la maggior parte dei browser di oggi è basato su chromium che è il motore alla base di chrome;
ma nonostante la qualità dei browser queste pagine sono di una complessità veramente alta
Richiedono ottime capacità di programmazione
Problemi di portabilità (coinvolgono tutti Client)
Problemi di sicurezza (eseguono sui Client) se ho degli utenti malintenzionati, possono rompere la mia

applicazione -> acquisire token in maniera tale che si prendono i privilegi della mia applicazione. Ovviamente, questa cosa può succedere anche con le pagine dinamiche, ma ho una superficie di attacco maggiore nella applicazioni su pagine attive

Soluzione di elevatissima complessità

Telnet e rlogin →servizi di terminale remoto

telnet e rlogin permettono il collegamento (sessione login) con macchina remota.

Funziona che noi dalla macchina locale ci colleghiamo alla macchina remota e vediamo l'interfaccia locale trasformato in un terminale del sistema remoto.

Quindi il terminale locale diventa un terminale del sistema remoto : a noi può sembrare di lavorare localmente ma in realtà stiamo lavorando da remoto.

telnet standard in TCP/IP (Internet) (gestisce eterogeneità di S.O., HW, etc.)

rlogin per i sistemi UNIX (remote login) → in sistemi UNIX c'è il concetto di terminale interno al sistema operativo ; nel kernel ci sono sistemi di gestione unix, significa che ci sono strutture/concetti per supportare il terminale. Questo permette di dare per scontato che tante funzionalità sono dentro al sistema operativo e non devo implementarle(molto più semplice).

I programmi applicativi telnet e rlogin hanno interfaccia verso utente da linea di comando (sono comandi in Unix).

Il protocollo di comunicazione è TCP/IP

telnet:

Una possibile implementazione del telnet su una macchina Unix in cui ha il concetto di pseudo terminale, può essere quello in cui il demone telnet crea una shell di login. Su essa possiamo lanciare dei vari comandi dall'applicazione.

La shell di login è agganciato al telnet attraverso il pseudo terminale che può essere una funzione del sistema operativo come in rlogin oppure è una funzione del pseudo terminale che deve essere fornito dal Server telnet.

Client stabilisce una connessione TCP con Server (porta 23), quindi accetta caratteri da utente e li manda al Server e contemporaneamente accetta caratteri del server e li visualizza sul terminale utente.

Server accetta la richiesta di connessione da Client e inoltra dati da connessione TCP al sistema locale.

Caratteristiche principali del protocollo applicativo di telnet:

• gestione eterogeneità tramite interfaccia standard (NVT)

• Client e Server negoziano le opzioni del collegamento (ASCII a 7 bit o a 8 bit)

• comunicazione simmetrica

Caratteristiche rlogin

• conosce l'ambiente di partenza e quello di arrivo, ha nozione di stdin, stdout e stderr (collegati al client mediante TCP).

• utilizzo di una sola connessione TCP/IP

• esporta l'ambiente del client (es. il tipo di terminale - TERM) verso il server

• **flow control:** il client rlogin tratta localmente i caratteri di controllo del terminale (ctrl-S e ctrl-Q fermano e fan ripartire l'output del terminale, senza attendere l'invio del carattere al server)

out-of-band signaling per i comandi dal server al client (es. flush output per scartare dei dati, comandi per il resize della finestra e per la gestione del flow control). **out-of-band** signaling implementato con TCP urgent mode.

in-band signaling per i comandi dal client al server (spedizione dimensione finestra).

xmpp

XMPP è un protocollo che permette la comunicazione real-time di messaggi e di informazioni di stato (assente, occupato, ecc.).

I client si registrano presso server locali.

Routing di messaggi e architettura simile a SMTP.

XMPP è un protocollo di comunicazione basato su XML (Extensible Markup Language). Il set standard XML codifica o traduce i documenti informatici in formato leggibile dalla macchina. XMPP è utilizzato per MOM (Message-oriented middleware), che sono programmi software progettati per l'invio e la ricezione di messaggi tra sistemi diversi.

TCP e UDP

La User Datagram Protocol (UDP) è uno dei protocolli di base utilizzati per la trasmissione di dati su Internet, con l'altro è Transmission Control Protocol (TCP).

Velocità

UDP, in media, può essere un protocollo molto più veloce per la trasmissione di informazioni di TCP, perché utilizza un totale di 8 byte per pacchetto di dati da trasmettere informazioni rispetto a TCP di 20.

senza connessione

Questa intestazione più snella viene perché UDP è, a differenza di TCP, senza connessione. Con il protocollo TCP, i dati è garantita per arrivare nell'ordine corretto ed è controllato per errori all'arrivo, mentre UDP non fornisce questa protezione: i pacchetti possono essere persi o danneggiati.

usi

Mentre il protocollo TCP viene utilizzato per la navigazione web, UDP è più spesso utilizzato per Voice over IP e giochi, dove la velocità è assolutamente fondamentale e i piccoli errori che possono apparire in UDP non sono così importanti.

Perché si usano solo TCP e UDP in internet?

TCP e UDP sono protocolli utilizzati per l'invio di bit di dati, noti come **pacchetti, su Internet**.

Essi sono sopra il protocollo internet IP quindi, se si sta inviando un pacchetto tramite TCP o UDP, quel pacchetto viene inviato sicuramente a un indirizzo IP.

TCP e UDP non sono i soli protocolli che lavorano su IP, tuttavia sono quelli più ampiamente utilizzati.

TCP è acronimo di Transmission Control Protocol ed è il protocollo più comunemente usato su Internet.

Quando si carica una pagina web, il computer invia pacchetti TCP all'indirizzo del server web, chiedendo di farci vedere quella pagina web per voi.

Il web server risponde inviando un flusso di pacchetti TCP, che il browser web mette insieme per formare la pagina web e mostrarla sullo schermo.

Quando si clicca un link, si accede a un sito o si invia un commento, il browser invia pacchetti TCP al server e il server risponde con altri pacchetti TCP.

Il protocollo TCP garantisce che il destinatario riceva i pacchetti.

Il destinatario (ad esempio il web server) invia la conferma di ricezione al mittente (il nostro computer).

Se il mittente non riceve conferma, rispedisce i pacchetti, e smette solo dopo un certo periodo di tempo se il destinatario non risponde perchè offline.

I pacchetti vengono inoltre controllati per eventuali errori.

Il TCP è molto affidabile e i pacchetti sono tracciati in modo che nessun dato venga perso o danneggiato durante il transito.

Questo è il motivo per cui i download di file non vengono danneggiati anche se si utilizza una rete lenta o che si interrompe spesso.

UDP è l'acronimo di User Datagram Protocol.

Un datagramma è uguale a un pacchetto di informazioni quindi il protocollo UDP funziona in modo simile a quello TCP, con una differenza, non controlla gli errori.

Quando si utilizza UDP, i pacchetti vengono inviati al destinatario velocemente senza attendere e senza assicurarsi che il destinatario li abbia ricevuti, continuando a inviare pacchetti.

Se il destinatario perdesse alcuni pacchetti UDP, non ha alcun modo di chiederli di nuovo.

In pratica una comunicazione **UDP non dà alcuna garanzia di ricezione dei dati**.

Il vantaggio è che **i computer possono comunicare tra loro più rapidamente**.

UDP viene utilizzato quando la velocità di rete è elevata e può essere superflue il controllo di errori.

Ad esempio, **UDP è spesso utilizzato per li video in diretta in streaming e per i giochi online**.

Un video in streaming in diretta è un flusso di dati continuo che viene inviato al computer.

Se si perde qualche fotogramma, esso viene saltato e di certo non sarà possibile chiedere di vederlo dopo.

I flussi di streaming UDP si differenziano rispetto a quelli TCP proprio perchè i pezzi di video non ricevuti vengono saltati.

Se si perde la connessione per alcuni secondi, il video si blocca per un attimo e poi passa al punto di ripresa saltando i pacchetti persi.

Se si verifica una minore perdita di pacchetti, il video o l'audio possono essere distorti per qualche istante e tornare buoni subito dopo.

Con i giochi online la storia è simile, se si perde qualche pacchetto UDP, i giocatori passano da un punto a un altro senza che si veda movimento.

Quello che conta è rimanere attuali nel gioco, senza guardare al passato e a ciò che eventualmente è stato perso.

Saltando la correzione di errori che farebbe TCP, si velocizza la connessione del gioco e si riduce la latenza.

Conclusione

Se un'applicazione utilizza il protocollo TCP o UDP dipende dal suo sviluppatore e non si può cambiare.

La maggior parte dei programmi vogliono la correzione degli errori e preferiscono la robustezza del protocollo TCP, mentre alcune applicazioni hanno bisogno di velocità e si affidano a UDP.

Con un programma come Wireshark si possono vedere i vari pacchetti che viaggiano avanti e indietro sul computer.

Se si sta configurando un router o un firewall per aprire certe porta, se non si è sicuri se un'applicazione utilizza TCP o UDP, si può scegliere di aprire "**entrambi**" per applicare la stessa regola sia al traffico TCP e UDP.

differenza tra ipv4 e ipv6

Un indirizzo IPv4 è composto da 32 bit e suddiviso in una parte che identifica la Local Area Network (LAN) in cui esso si trova (network ID) e in una parte di host (host ID) che identifica il particolare nodo all'interno di quella rete

- La lunghezza del network ID è variabile e quindi permette il subnetting
- Supporto al subnetting, cioè posso creare delle sottoreti all'interno di una rete un pò più grande assegnandogli alla sottorete una porzione di network ID più lunga rispetto a quello della rete principale
- Rappresentazione di indirizzi IPv4 segue la cosiddetta "dotted notation": 192.168.0.1/24

E' rappresentato da indirizzi da 4 byte, ogni byte va da 0 a 255 partendo dal byte più significativo a sinistra.

/24 identifica il numero di bit che compongono la network ID. In questo i 24 bit sono i primi 3 byte cioè 192.168.0 e rappresentano il codice della LAN dove si trova il mio nodo ed ha come host 1 all'interno della LAN

Il numero massimo di dispositivi con indirizzo IPv4 è di circa 4 miliardi (2^32). In realtà non tutti gli indirizzi sono utilizzabili, visto che alcuni indirizzi (~18 milioni) sono riservati per l'uso in reti private:

– 10.0.0.0/8 (255.0.0.0)

– 172.16.0.0/12 (255.240.0.0)

– 192.168.0.0/16 (255.255.0.0)

In più altri indirizzi sono riservati per multicast, routing. In generale c'è limite massimo di efficienza di assegnazione degli indirizzi.

Problemi di IPv4

IPv4 ha un numero molto limitato di indirizzi e sono esauriti ovunque. L'esaurimento degli indirizzi significa che il local registry non è più in grado di assegnare degli indirizzi IPv4.
Quindi per attaccare nuovi dispositivi all'interno di Internet? Non possiamo utilizzare indirizzi pubblici ma quelli privati utilizzando il NAT (collegato reti privati all'internet pubblica) oppure fare la migrazione a IPv6 ma non è una soluzione banale

IPv6

IPv6 è la nuova versione di IP, specificatamente progettata per superare i limiti di scalabilità di IPv4, in particolare per quanto riguardo la dimensione per lo spazio degli indirizzi.
IPv6 adotta indirizzi di 128 bit e questo consente di allocare (in teoria) 2^128 cioè 3,4 * 1038 indirizzi (un numero di indirizzi per metro quadro di superficie della Terra maggiore del numero di Avogadro).
Ha numerosi vantaggi rispetto a IPv4;
– Adozione di un formato di frame più semplice (e quindi veloce da processare per i router)
– Indirizzi con diversi scope: link-local, site-local (deprecato) e global
– Ripensamento di alcuni protocolli (es. ARP) in favore di soluzioni più robuste
– Estensioni per connettere dispositivi low-power

La differenza tra questi due formati di indirizzi sta nel fatto che gli indirizzi IPv6 sono più lunghi e formattati in modo diverso, quindi esistono più configurazioni di indirizzi IPv6 univoci possibili. IPv4 è un sistema a 32 bit che utilizza una stringa di numeri separati da punti, mentre IPv6 è un sistema a 128 bit che usa sequenze alfanumeriche separate da due punti.