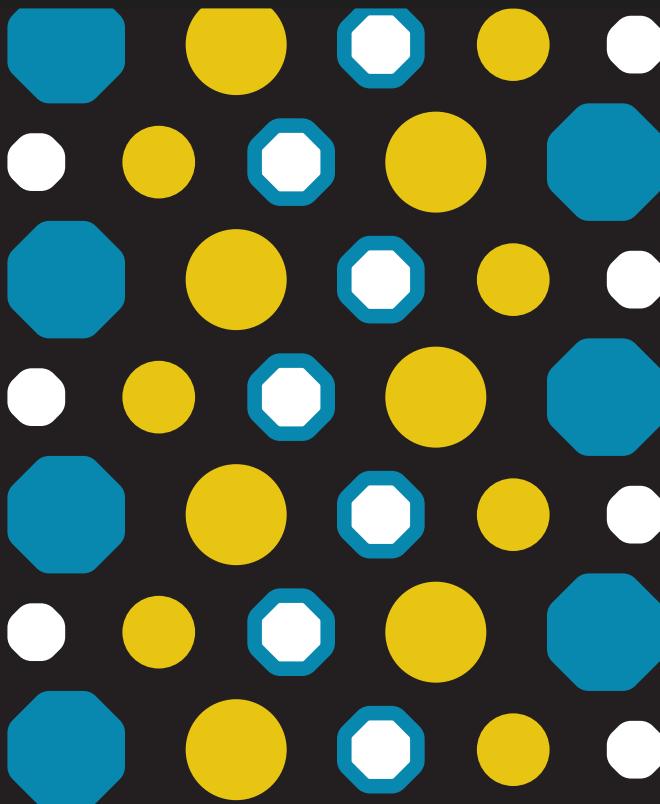


RETI DI CALCOLATORI

By Edoardo



RETI DI CALCOLATORI

wengxt

IP indirizzamento e indirizzamento

↓
IP stessa rete → invio diretto

IP rete diversa → invio al gateway → arrivo a destinazione

Web Service → si limita a mostrare contenuti da modificare manualmente
Application Server → prende informazioni in input e mostra contenuti in funzione di esso.

TCP

- Orientato alla comunicazione: stabilisce la connessione prima di trasmettere (handshake)
- Trasferimento affidabile tramite Ack
- Ricezione ordinata dei pacchetti tramite Ack e ritrasmissione
- Controllo di errore, flusso e congestione

UDP

A differenza di TCP non si preoccupa di stabilire la connessione e non offre garanzie sulla connessione o la sua stabilità

TCP invia flussi di byte mentre UDP invia datagrammi

Processi che sono in esecuzione su host diversi comunicano tramite meccanismi definiti dal protocollo (sequenze di comandi codificata) dello strato di applicazione

Client - Server Il client chiede e il server risponde. Solitamente viene chiesto un servizio che poi il server fornirà.

Tramite indirizzo IP raggiungo l'host ricevente e con il numero di porta il processo destinatario

Perdita Dati alcune app sono tolleranti (es audio) altre invece richiedono affidabilità totale
Banda dipende dal tipo di dato inviato.

A seconda di quanto Dondo uso o quota tolleranza di perdita ho deciso che protocollo (udp /tcp) usare es. banda e perdita costante \rightarrow udp

Jitter vsicazione del ritardo

TCP ha più PRO del UDP, ma in molti casi UDP è preferito in alcuni tipi di app.

HTTP

Una TCP. HTTP request dal client e HTTP response dal server

È un protocollo stateless ovvero non mantiene informazioni sulle richieste poste dal client

- Non Persistente chiude la connessione dopo lo ricezione di ogni elemento richiesto
 - Persistente la connessione è chiusa dopo un tempo di inattività

1.0 apre e chiude la connessione per ogni oggetto, 1.1 invece permette di lasciare attiva per un periodo di tempo.

REST e JSON

Si basa su `Http` per manipolare ed accedere a `www.e` creare servizi web

Il client chiede una risorsa e il server risponde inviandola SENZA ANCHE INFO DEL CLIENT (salvo eccezioni)

Ogni risorsa ha il proprio identificativo URL (equividente all'URI).

Incorpora tutti i metodi CRUD (Post, Get ...)

AUTENTICAZIONE

l'obiettivo è controllare l'accesso ai documenti nel server.

In questo modo sappiamo chi accede, ma non "lo sequenzia",
in cui si accede"

Cookie Il server risponde con un "cookie" ovvero un identity → identificativo salvato poi nel database sia del client che del server. Alle successive richieste il client invierà anche il cookie con cui il server capirà "chi è che chiede".

E possibile condizionare le GET specificando nell' header "If-modified-since <data>"

Nel caso il file non è stato modificato verrà preso dalla cache

Da qui deriva il web caching



FTP

Due canali: controllo e dati su porta 21 e 20

Il controllo avviene tramite scambio di messaggi di controllo "fuori banda"

Usa comandi simili ad http: RETR = GET ...

SMTP: protocollo per "consegnare" le mail sul server

POP3/IMAP: protocollo per "recuperare" le mail dal server

DNS

Host e router hanno indirizzo IP e nome, il DNS è un database distribuito per gestire queste corrispondenze, implementato come una gerarchia di **name server**

E' un protocollo applicativo usato da host, router e nome server che utilizza UDP per tradurre i nomi in indirizzi IP

Il DNS ha anche lo scopo di distribuire il carico tra server replicati garantendo onde di raggiungere quello più vicino

N.B. NESSUN nome server conosce tutte le corrispondenze nome-IP

Name Server Locale: è il NS di default offerto dall'ISP ed è quello che viene interrogato per primo

Name Server di riferimento: (per un host) per definizione è quello che è sempre in grado di tradurre il nome in IP (mapping)

Root Name Server: viene contattato da NSLocale che non riesce a risolvere uno nome. Il root NS interroga il NS di riferimento e invia la risposta al NSLocale. Nel mondo ci sono una dozzina di root NS

Quando un NS apprende una traduzione lo salva in cache (caching). Le traduzioni salvate localmente scadono dopo un certo periodo di tempo (solitamente 2 giorni).

I DNS memorizzano **Resource Record (RR)**

Formato RR: (nome, valore, tipo, ttl)

• Tipo = A → nome = nome host, valore = IP

• Tipo = NS → nome = dominio, valore = IP del NS di riferimento del dominio

• Tipo = CNAME → nome = alias di qualche nome reale ("cononico"), valore = nome cononico

• Tipo = MX → valore = nome mailserver associato a nome

PROGRAMMAZIONE AD EVENTI

Un singolo thread prende tutte le richieste, quando sono completate arriva il processo richiedente (callback) e invia il risultato

read("text.txt", function) → al termine della read di text.txt verrà eseguita la function.

Le callback possono essere omnidate.

SICUREZZA

Integrità, Disponibilità, riservatezza, autenticazione e Non ripudiazione

Disponibilità richiede prestazioni e robustezza.

Integrità comprende onde il recupero degli errori dovuti a problemi fisici della rete.

Non ripudiazione: un utente non può dire di non aver inviato un messaggio firmato (autenticazione).

Intruso passivo: legge i messaggi senza alterarli

Intruso Attivo: oltre a leggerli li modifica alterandone le informazioni. Può spacciarsi per il mittente autentico.

Possibili "Attacchi": IP sniffing, DDOS

CRITTOGRAFIA

A critta il PlainText e lo invia a B che lo decodifica: $D(C(m)) = m$

chiave segreta (simmetrica): chiave di decrittazione di A e B uguale

chiave pubblica (asimmetrica): chiavi differenti

D) La codifica è una funzione di una chiave (segreta). Se si conosce la chiave la decrittura è semplice, quasi impossibile nel caso contrario

La sicurezza è volutata in base ai risorse di tempo e costo per decrittare la codifica. La sicurezza di un protocollo dipende anche dalla dimensione del numero di possibili chiavi (# tentativi nel caso di brute force)

Con chiave simmetrica serve una chiave per ogni coppia di utenti che si scommettono dati/informazioni $\rightarrow O(n^2)$ chiavi

\Rightarrow Alternativa: K_{PRIV}

K_{PUB}



$$(m) \longrightarrow K_{PRIV}(m) \longrightarrow K_{PUB}(K_{PRIV}(m)) = m$$

$$K_{PRIV}(K_{PUB}(m)) \leftarrow K_{PUB}(m)$$

2) Nel caso assimmetrico ho $O(n)$ chiavi. Abbiamo una chiave segreta per ogni utente e una unica pubblica in comune per tutti

$$m \rightarrow K_{\text{segreta}} \rightarrow C(m) \rightarrow K_{\text{pubblica}} \rightarrow$$

RSA:

- 1) due numeri grandi e primi p e q
- 2) $n = pq \geq (p-1)(q-1)$
- 3) $e < n / \text{red}(e, \varphi) = 1$ $e = \text{encryption}$
- 4) $d / e \cdot d \bmod \varphi = 1$ $d = \text{decryption}$
- 5) $(n, e) = \text{chiave pubblica}$ $(n, d) = \text{chiave privata}$

\rightarrow Cifra m ($< n$)

$$\begin{aligned} C &= m^e \bmod n \quad (\text{Pub}) \\ m &= C^d \bmod n \quad (\text{Priv}) \end{aligned}$$

RSA è sicura poiché è facile calcolare $f(\text{cifra})$ ma non $f^{-1}(\text{decifra})$ perché sarebbe un problema troppo difficile computazionalmente o a forza bruta

FIRMA DIGITALE

Sequenza di bit che dipende dal documento e dalla persona che firma.
La firma deve poter essere verificata, non falsificabile e non ripudiable
 $\text{doc} \rightarrow \text{RSA} \rightarrow \text{FIRMA doc}$

↑
chiave segreta

Lo firma viene poi decodificata con la chiave pubblica

Il procedimento è lungo per documenti grandi quindi si "firma" una sua parte (impronta digitale)

Avere documenti con lo stesso impronta è difficile, ma possibile (per tentativi)

$2^{\frac{m}{2}}$ con $m = \text{documento}$ è il numero di documenti da creare per avere la probabilità di avere due impronte uguali maggiore del 50%

OAuth

Autenticazione dell'utente e garantire l'accesso a terzi (app) alle info dell'utente autenticato

FLUSSO:

- 1) Il client chiede il permesso di accedere a info protette (scope)
Il server che le detiene (se diamo il permesso) restituisce un codice di autorizzazione (es. login Google → noi stessi)
- 2) Il client invia il codice al server di autorizzazione (es. Google) che risponde (se il codice è corretto) con un token d'accesso al servizio richiesto
- 3) Il client invia il token al servizio che risponde con le info richieste nel caso il token sia valido

NB. E' possibile che il "Resource Owner" ci fornisca direttamente il token da poter usare per il servizio richiesto (*implicit grant*)

P2P

Paradigma per le applicazioni distribuite. Le entità che ne fanno parte condividono le risorse e contribuiscono insieme per offrire il servizio.
Le entità non sono più client o server, ma svolgono contemporaneamente i due ruoli: sono dette **peer** e agiscono come **servizi**.
Gli indirizzi non sono più noti a priori poiché cambiano in base alla rete (Overlay) che i peer usano.
L'**Overlay Network** è una rete virtuale "sopra" la rete fisica in cui si farà la ricerca per identificare il "node" richiesto per il servizio.

CHORD

Protocollo che tramite una chiave ritorna l'indirizzo in cui si trova la risorsa
Utilizza **Distributed Hash Table** ovvero tabelle con le coppie chiave: valore
I nodi sono organizzati in "cerchi" e ognuno ha una sequenza di m-bit (node: hash IP, key: hash key)

Nella ricerca della risorsa conosciamo la funzione di hashing e quindi il nodo a cui mi devo riferire. Nel caso in cui l' n -esimo nodo è offline posso automaticamente al successore poiché ogni nodo conosce il suo successore.

BLOCKCHAIN

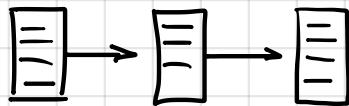
Un blockchain è un "ledger distribuito" che tiene traccia delle transazioni.
Ad ogni transazione viene inviato un messaggio broadcast a tutte le entità per aggiornare tutti i ledgers.

Per mantenere l'autenticità viene usata una firma digitale che combina ad ogni transazione (tutta stessa entità).

Ogni transazione contiene anche le transazioni precedenti che garantiscono la disponibilità dell'output outputs \leq inputs

La blockchain garantisce l'ordine delle transazioni: se trovo due blocchi con lo stesso transazione cancello il secondo.

Questo permette di evitare di fare più spese con la stessa moneta (bitcoin)



QUALITÀ SERVIZIO

- 1) Modifiche a livello applicativo e di trasporto
- 2) Modifiche allo stato di rete

- Real-time Unidirezionale (es. Streaming video): pochi vincoli sui ritardi
- Real-time Interattivo (es. Videoconferenza): vincoli sui ritardi più stretti.

Soluzioni:

1) Sovraccarico dimensionamento: maggiore banda

2) Modifiche ai protocolli:

- riservare risorse per specifiche funzioni

- stabilire accordi con l'operatore del servizio (Service Level Agreement, SLA)

3) Comprensione

STREAMING

applicazione helper: mezzo con il quale "uso" i dati inviati in streaming (per video → il player del browser)

RTSP: protocollo per i "comandi" sul file (es. play, pausa ecc.)

RTP / RTCP (2)

RTP ridefinisce uno standard per i pacchetti multimediali, scolabile. RTP invia pacchetti via socket UDP e sta al programmatore estrarre le informazioni

RTCP funziona insieme a RTP e trasmette informazioni per il monitoraggio e il miglioramento delle performance: per esempio diminuire la connessione in ingresso da una data sorgente se lo sorgente in questione è già al massimo della velocità.

(Con RTCP il trasmettitore invia al ricevitore info per lo sincronizzazione del flusso (UDP non ha un flusso stabilito))

Per non uscire meno dall'ottimizzazione i pacchetti RTCP possono occupare al massimo il 5% di banda per ogni sessione

H.323 e SIP sono due protocolli che "simulano" chiamate audio-video su internet

→ Compi RTP e funzioni!

1)

- MARCHIATURA: differenzia i pacchetti in base alla loro specifica

- PROTEZIONE: isolza flussi tra applicazioni

- CALL ADMISSION: un'applicazione può essere rifiutata a priori se chiude troppo banda non disponibile

Una politica di call admission permette anche di sostituire pacchetti o borsa priorità con pacchetti ad alta priorità invia di scortarli a priori se la coda è piena

Per valutare la richiesta di risorse delle applicazioni ci si basa sul FLUSSO MEDIO, FLUSSO MASSIMO e la DIMENSIONE DEL BURST (picchi)

T Spec: caratteristiche del flusso

R Spec: caratteristiche richieste dal servizio

I parametri di T Spec sono inseriti nel **Token Bucket**:

• Token Rate r

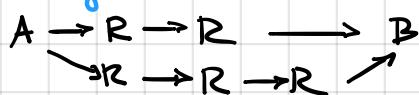
Ad ogni pacchetto inviato rimuovo un token, finiti i token stop.

Il bucket si "riempie" di r token ogni secondo e contiene al massimo b tokens
 $(b + r \cdot T)$

A differenza di best effort i pacchetti "in ecceso" che non entrano nel buffer sono scartati (come su best effort), ma solo se l'applicazione invia un flusso medio maggiore di quello dichiarato in connessione (+ picco massimo) e non a prescindere

In alternativa al call admission vi sono applicazioni collettive.

Integrated Server



Da A a B per ogni router si verifica la call admission: se il router garantisce il "possesso" dei pacchetti entro un certo ritardo (di RSpec): fino a B si "costruisce" un circuito virtuale da A a B.

La call admission può essere eseguita sulla qualità del servizio (QoS) o su CARICO CONTROLLATO. Il secondo caso ha garanzie più forti e si basa sul fatto che i router inviano i pacchetti con un carico, oppunto, controllato: i ritardi saranno maggiori e potrebbe capitare di perdere pacchetti.

CLOUD COMPUTING

Risorse computazionali in un unico pool da dare come servizio agli utenti
ON DEMAND

Alla base del cloud computing c'è la virtualizzazione: hardware, software, storage e risorse di rete

- Storage Distribuito

- API

Il cloud computing ha le caratteristiche di essere modulare e quindi scalabile

OPENSTACK

Piattaforma open-source per il cloud computing

1) Request Provisioning dall' user interface

- Login su HORIZON (dashboard)
- Configurazione VM
- Launch

I parametri sono inviati tramite POST al server back-end: se non sono autenticati possiamo trarre Keyston per il token 2)

Keystone è una sorta di database che contiene utenti, gruppi, token, domini, ruoli, progetti e tutto ciò che serve per l'identificazione.

Horizon invia una richiesta a Keystone che verifica se:

- l'utente è autorizzato
- l'utente è autenticato
- la richiesta è applicabile

3) Il comando viene inviato a NOVA che si occupa di gestire le API REST HTTP. Nova verifica se il token sia valido tramite le API Keystone

4) Keystone risponde alla verifica del token

5) Nova posza la richiesta in un oggetto python e salvato nel database di Nova

6) Nova con API RPC.CAST mette in coda un "messaggio" con le info della VM

7) Lo scheduler di Nova entra nella queue e legge il comando da eseguire

Lo scheduler può applicare filtri sugli host basati su proprietà statiche degli host stessi

8) Lo scheduler preso il comando ricava tutti i relativi dati dal NOVA DB e sceglie il compute node in base ai filtri

Al termine inserisce in queue un messaggio per lo supervisore della VM

9) Nova COMPUTE chiede informazioni sulla VM dal DB. Per aumentare la scalabilità e la riavetta questo scambio di informazioni passa attraverso il Nova conductor con RPC.CALL

10) Il NOVA COMPUTE chiede a NEUTRON di assegnare (creare) una rete alla VM.

Neutron tramite plugin e agent crea indirizzi MAC e IP e li associa alla VM

11) Nova COMPUTE chiede a CINDER di istanziare un volume per lo storage

12) Nova COMPUTE tramite GLANCE ricava una URI per l'immagine da usare nella VM. Con tale URI si prende in SWIFT l'immagine e lo si installa sulla VM

13) L'hypervisor di Nova istanzia la VM e notifica lo stato di RUN sulla dashboard

MESSAGE PASSING ASINCRONO

In http ci si aspetta sempre una risposta (risorsa), in tecnologie tipo RabbitMQ il passaggio di messaggi è asincrono e i componenti della struttura di rete (client, server ecc.) non aspettano la risposta per proseguire

RABBITMQ:

P (producer): sender

QUEUE: mailbox che "vive" in RabbitMQ. Dal punto di vista concettuale sono limiti

C (consumer): receiver

Si accosta al protocollo AMQP

La coda in RabbitMQ è scalabile poiché tutti i consumer si connettono ad essa e gestiscono i messaggi secondo il paradigma Round Robin (uno per uno alla volta)

Round Robin non garantisce però un equo corico di lavoro

Un altro problema è la perdita di task in caso di morte di un consumer: per tale problema RabbitMQ supporta gli ACK

ACK: cancella lo task dalla coda solo dopo che il consumer l'ha processata

NB. Nel caso "nuovo", la coda impostandola durable: true e i singoli messaggi a persistent: true abbiamo lo garantito di salvare la coda e i messaggi

Per distribuire il corico si usa il metodo prefetch = 1 che garantisce di inviare un messaggio solo dopo aver ricevuto l'ACK del precedente

Publish / Subscribe: un producer pubblica un messaggio e solo i "subscriber" consumer possono gestire tale messaggio

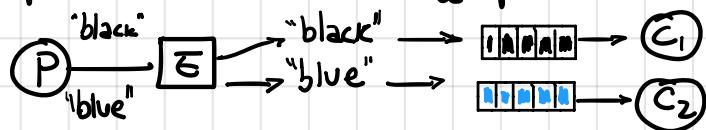
EXCHANGE

Il producer invia il messaggio ad un exchange che lo invia a sua volta alle varie code

Uno consumer ci si collega all'exchange prima della coda (binding)

Adesso avremo una coda per ogni consumer

È possibile inserire un "topic" durante il binding della coda all'exchange: così facendo è possibile dividere i messaggi per code



I topic possono andare a più volti (es. speed, color, species)

L'exchange come primo dividerà le code a seconda del binding utilizzando però più parametri



TECNOLOGIE WIRELESS

Rete ad infrastruttura: un access point gestisce più "ordine" a cui gli host si connettono

Comunicazione host-host passa sempre dal centro-stella

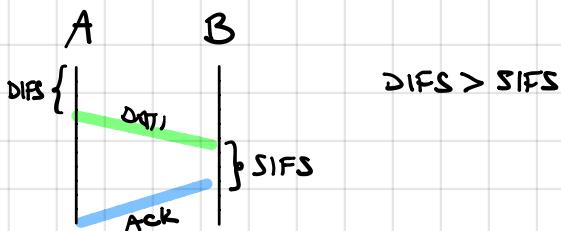
Rete Mesh: gli host stessi provvedono all'indirizzamento dei messaggi senza opporsi a nessuna infrastruttura.

Oltre ai problemi di distanza e frequenza vi possono essere problemi come il terminale nascosto: $A \rightarrow B \leftarrow C$

A e C sono ignari dell'esistenza dell'altro e la loro connessione su B può creare interferenza facendo morire perde entrambe le trasmissioni.

→ CSMA / C (ollision) A (voidance)

CA:



- Se il canale è libero → invio
- Se il canale è occupato → ritardo casuale
- Se non ricevo Ack → aspetto un tempo doppio al ritardo precedente

In aggiunta quando (per esempio) A invia dei pacchetti (voli piccoli per avere minore probabilità di collisione) invia prima una RTS a B che a sua volta invia un CTS a C mettendolo a conoscenza di A. Così facendo chi invia l'RTS per primo "prenota" il canale.

IoT

Protocolli usati: CoAP e MQTT

MQTT: gestisce domande/risposte in modo asincrono (come RabbitMQ) e a basso consumo (pacchetti scombiati leggeri)

Non si occupa in alcun modo di cosa stiamo inviando (no MimeType al contrario per esempio di HTTP)

Funziona

- KeepAlive: capire chi è attivo e chi no (ping request e response)
- Will message: specifico il tipo di richiesta
- Retain message: i messaggi sono ricaricati anche da subscribers futuri (sono tenuti in "DB")
HTTP ha dei problemi quali la complessità dello sviluppo sia lato server sia lato client e la QoS non è garantita, in più HTTP è molto dispendioso in fatto di risorse

VS CoAP

A differenza di MQTT si basa su UDP.

Rispetto ad HTTP lo sviluppo client è semplificato tramite header binari, QoS con messaggi confermabili.

CoAP permette il "resource discovery" tramite broadcast

Tutti i nodi hanno anche il ruolo di server che può causare problemi di NAT (ogni nodo necessita di un'indirizzo)

VS AMQP (RabbitMQ)

RabbitMQ permette cose più sofisticate grazie ai canali con topic multivalore

MQTT è molto semplificato

CoAP:

Alternativa (o quasi) di HTTP, semplice e leggero e compatibile con metodi REST

A differenza di HTTP come il client si connette (vede) il server anche il server vede e può inviare richieste al client

RIEPILOGO BLOCK CHAIN (BITCOIN)

LEDGER: è l'olla base del blockchain e rappresenta la lista dei soldi dei conti degli utenti che partecipano al network (es. Bitcoin). Tramite messaggi broadcast un utente notifica di inviare tot all'utente x e tutti i nodi aggiornano il ledger.

SICUREZZA: ad ogni transazione viene associata una "firma" data dalla "combinazione" del messaggio e di una chiave privata. Chi riceverà il messaggio potrà verificare l'autenticità tramite un algoritmo che prende in input il messaggio, la firma e la chiave pubblica del mittente.

Ogni transazione garantisce che l'utente abbia abbastanza fondi linkando transazioni precedenti.

ANONIMITÀ: è garantita dal fatto che ogni utente è identificato solo da una chiave pubblica (e una privata) generata casualmente. Le chiavi pubbliche e private sono abbastanza complesse da renderne la duplicità impossibile.
NB. perdere la chiave segreta implica la perdita di bitcoin

BLOCKCHAIN: un utente malevolo potrebbe inviare due transazioni, una ad un destinatario qualunque per ricevere in risposta e lo secondo successiva a se stessa dello stesso importo. Nello stato di verifica il destinatario potrebbe eseguire il servizio per poi non ricevere i soldi poiché la transazione a lui destinata è arrivata per seconda, e, mondo Input (ammontare della transazione) già utilizzati, ritenuta invalida.

Il blockchain garantisce l'ordine delle transazioni. Ogni blocco contiene delle transazioni verificate e accadute nello stesso momento.

Ogni nodo raccolge transazioni non ancora verificate e ne crea un blocco che vorrà candidato come nuovo blocco della blockchain. Per essere "eletto" nuovo blocco della blockchain tale blocco dovrà includere una combinazione (hash) estremamente complessa. Le possibilità che due computer indovino la combinazione nello stesso tempo è molto bassa, ma in caso succedesse ogni computer continuerà sul primo nuovo blocco che riceverà generando più "rompi": al prossimo nuovo blocco (è quasi impossibile che copiti due volte di fila) si seguirà il rombo più lungo.

Tornando al problema della doppia transazione: il mittente malevolo darebbe in questo caso generare un rombo più lungo di tutti quelli in circolazione per "informare" la rete cosa che richiederebbe troppa capacità di calcolo = soldi e non ne garantirebbe comunque la riuscita (1 vs il mondo).

RIEPILOGO MESSAGE QUEUING

RabbitMQ è un BROKER che accetta e inoltra messaggi basato su AMQP
AMQP fornisce flusso controllato, comunicazione orientata ai messaggi con garanzia di ricezione del messaggio (at most once, at least once, exactly once), autenticazione.

INTRODUZIONE: un (o più) produttore si connette alla coda (dentro il broker) e pubblica il suo messaggio. Il consumer allo stesso modo si connette alla coda e "consuma" i messaggi (onde i consumer possono essere più di uno).

CODE DI LAVORO

È possibile inviare "tasks" a più consumer che di default lo eseguiranno secondo il round-robin. In caso di morte di un consumer durante lo svolgimento del task tale task verrà perso a meno che non sia implementato un sistema ad ACID, in tal caso il messaggio viene rimesso in coda o inviato a consumer online.

La perdita dei messaggi può avvenire anche nel caso in cui il server RabbitMQ crashi, ma onde in questo caso è possibile salvare un "backup".

PUBLISH / SUBSCRIBE

Tra il produttore e la coda vi è in realtà un altro elemento chiamato EXCHANGE che (a seconda di come è configurato) decide cosa fa del messaggio: quale coda? Tutte? Da bustone? ecc.

ROUTING

Durante lo step di Binding ovvero dell'assegnazione dell'exchange ad una coda è possibile specificare lo "routing-key": ogni messaggio verrà indirizzato nella coda appropriata.

TOPIC

Estrattamente come per il routing con la differenza che lo "routing-key" in questo caso è composta da più parole es. lazy.orange.elephant (categorie) ogni parola può essere sostituita da * o # per indicare rispettivamente una parola qualsiasi e una o più parole qualsiasi.

Es. *.*.rabbit: una coda controlla topic accetterà tutti i messaggi che hanno come terzo criterio la parola "rabbit".

Il principale uso del message queuing è nei sistemi distribuiti per dividere il carico di lavoro su più nodi con semplice messaggio ASINCRONO.

RIEPILOGO P2P

Il peer-to-peer è un paradigma di progettazione per le applicazioni distribuite in cui le entità partecipanti condividono le loro risorse per fornire il servizio

DIFFERENZE C/S

Un peer agisce sia da client che da server

I servizi sono forniti in modo distribuito e decentralizzato, la loro capacità è costituita dall'offerta delle risorse e la loro qualità dipende da esse (messa a disposizione AUTONOMICAMENTE)

I peer si connettono tra loro tramite un overlay network ovvero una rete logica sopra lo strato fisico (tramite TCP)

- SCALABILITÀ: limitata solo dalle operazioni di coordinamento e sincronizzazione
- CONDIVISIONE / RIDUZIONE COSTI
- DISPONIBILITÀ DEL SERVIZIO
- AUTONOMIA: ogni peer è autonomo in ogni scelta
- ANONIMATO / PRIVACY

Oltre al file-sharing il sistema P2P è usato anche per condividere capacità di calcolo (cicli di CPU) nel distributed computing dove problemi complessi sono eseguiti dividendoli ed eseguendoli parallelamente su più computer.

LOCALIZZAZIONE RISORSE

MODELLO CENTRAZZATO

I peer pubblicano su un server le risorse che condividono. Il server risponde alle query con le identità dei peer che offrono la risorsa richiesta.

- Più scalabile dell'architettura C/S: il server contiene solo metadati
- Non anonimo
- No FAULT-Tolerant: se il server cede, il servizio viene interrotto
- Servizio Deterministico: il server ha visione globale del sistema e se una risorsa esiste è localizzabile

MODELLO DISTRIBUITO NON STRUTTURATO

Sono i sistemi P2P puri senza entità gerarchicamente superiori

NON STRUTTURATO = la formazione dell'overlay non è controllata
è necessario un algoritmo di routing per i messaggi del protocollo (TCP)

Una volta trovato il peer con la risorsa che si sta cercando viene instaurata una connessione esterna all'overlay (out-of-network) http

- Servizio Non Deterministico: non si ha una visione globale del sistema
- Lo load di mantenimento e di ricerca del file consuma molta banda

SISTEMI IBRIDI

È un compromesso tra il determinismo del modello centralizzato e la scalabilità del sistema puro.

L'overlay è diviso in cluster e la ricerca delle risorse è ristretta al cluster di appartenenza; eventualmente è diffusa all'esterno.

TORRENT

Nel caso dei torrent la ricerca è effettuata out-of-band (es. via web) e per ogni file viene instaurato un overlay: il file è diviso logicamente in chunk che possono essere scaricati parallelamente da più peer.

L'entrata nell'overlay (bootstrap) è gestita da un server (tracker) che contiene una piccola lista di nodi già presenti nell'overlay richiesta.

RIEPILOGO CRITTOGRAFIA

Necessità di garantire: disponibilità, riservatezza, integrità, autenticazione e non ripudiazione

SNIFFING: utente malvagio legge i pacchetti in transito tra A e B

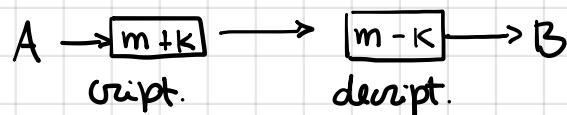
SPOOFING: utente malvagio si finge A e invia pacchetti a B

DoS: utente (o più = DDoS) sommerge A di pacchetti maligni bloccandone il servizio

CHIAVE SIMMETRICA

Una chiave segreta per criptore e decriptore i messaggi

La sicurezza è volutata in base alle risorse di tempo e calcolo necessarie per risolvere la decriptazione senza la chiave segreta e in base al numero possibile di chiavi

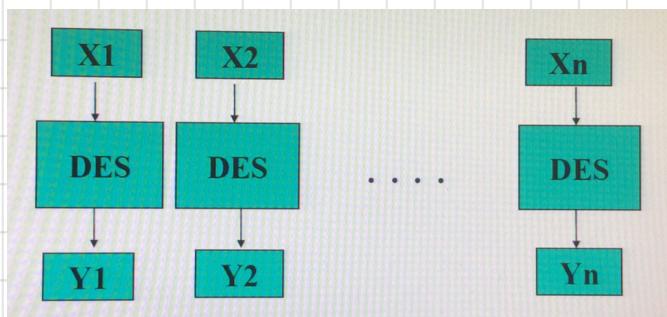


A e B necessitano della stessa chiave segreta con relativi problemi sul come accordarsi
Numero elevato di chiavi: $O(n^2)$

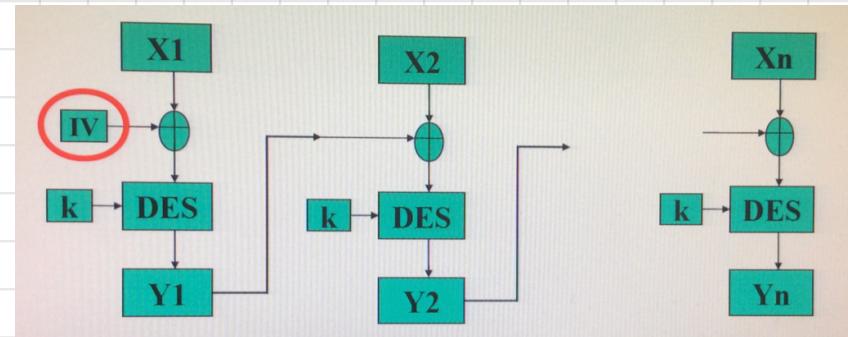
Esistono processi di codifica / decodifica implementati nell'hardware che rendono il processo molto veloce come DES in cui ogni 64 bit del messaggio passano 16 rounds di permutazione ognuno in funzione di bit differenti della chiave (processo inverso per decodificare)

DES a sua volta è usato per altre strutture: Electronic Codebook Chaining (ECB) o cipher Block Chaining (CBC). La prima divide semplicemente il messaggio in blocchi da 64 bit e applica DES a ognuno, la seconda invece inserisce nell'algoritmo di un blocco il blocco precedente

ECB



CBC



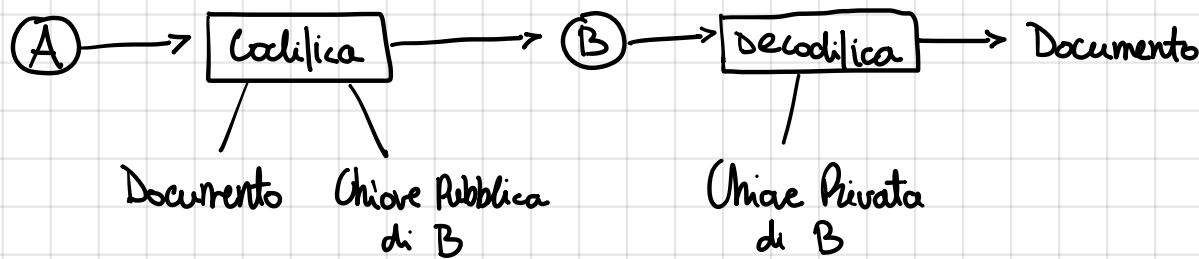
CBC più lento

ECB non ha dipendenza tra blocchi ed è quindi più vulnerabile ad attacchi di sostituzione

CBC ha propagazione di errori che non ha in ECB

CHIAVE ASIMMETRICA

Vengono usate due chiavi: una pubblica per codificare e una privata per decodificare



Le chiavi sono scelte tramite RSA:

- Si sceglono due numeri primi grandi p e q (es 1024 bit)
- Si calcola $n = pq$ e $\varphi = (p-1)(q-1)$
- Si sceglie e ($e < n$) con e e φ primi tra loro
- Si sceglie d tale che $ed - 1$ sia divisibile per φ
- Lo chiave pubblica sarà (n, e) e quella privata (n, d)

Cifratura: $c = m^e \text{ mod } n$

Decifratura: $m = c^d \text{ mod } n$

($m = \text{messaggio } < n$)

La sicurezza è data dal fatto che è veloce codificare, ma impossibile decodificare senza la chiave

Di contro si ha che codificare un messaggio lungo può impiegare tanto tempo: per risolvere questo problema il mittente genera una chiave con cui codifica il messaggio con DES e poi codifica solo la chiave con RSA. Il ricevente userà la sua chiave privata per decodificare la chiave DES che userà per decodificare il messaggio.

FIRMA DIGITALE

Chiunque deve poterne verificare l'autenticità, deve essere non falsificabile e non ripudiabile

OZIONE 1:

Si codifica il documento con la chiave privata del mittente e si invia il documento in chiaro. Il ricevente lo decodifica con la chiave pubblica del mittente e vede se coincide con l'originale.

Il metodo è lento sia per la codifica che per la decodifica

OZIONE 2

Firmare solo una porzione di documento → IMPRONTA DIGITALE

L'impronta è generata dal documento tramite algoritmi come SHA

Si invia l'impronta codificata con RSA e il documento: il ricevente decodifica l'impronta e la compara con l'impronta che il documento genera con SHA

Ogni documento (anche della stessa persona) ha impronte diverse: trovare due documenti con lo stesso impronta è possibile solo per tentativi → IMPOSSIBILE (l'impronta deve essere comunque lunga)

OAUTH

Protocollo per accedere a servizi attraverso terzi senza dover esplicitamente le credenziali ad essi.

Il client richiede l'autorizzazione tramite login (es. Accesso tramite Facebook) e gli viene restituito un codice di autorizzazione.

Con questo codice è possibile ricevere un token dal server di autorizzazione che permette l'accesso alle risorse.

Inviamo il token al server del servizio / delle risorse abbiamo accesso a quello per cui siamo autorizzati.

RIEPILOGO QoS

Esistono varie classi di applicazioni multimediali ed ognuna ha delle esigenze per la sua corretta esecuzione

TCP/UDP/IP forniscono un servizio Best Effort che non garantisce sempre le necessità di un'applicazione

SOLUZIONI IP

Sovraccarico: più banda e possibilità di caching

Modifiche al protocollo: riservare risorse, combinare le politiche di scheduling o stabilire accordi preliminari con le app. sul livello di servizio fornito

Modifica delle politiche di routing: non più FIFO

Lo jitter di due pacchetti è la differenza tra l'intervallo di tempo che intercorre tra la trasmissione e la ricezione di essi. Per risolvere tale problema può essere implementato un ritardo tramite bufferizzazione: questo ritardo non deve comunque essere né troppo grande (telefonia IP) né troppo piccolo (non risolverebbe il problema)

Il ritardo può essere adattivo in base all'uso che si sta facendo dell'app.

Nel caso di perdita di pacchetti:

- 1) ARQ: non accettabile per applicazioni interattive
- 2) FEC: invio copie冗余的 dei pacchetti

Per le perdite di pacchetti a burst si applica l'interleaving: il file è diviso in blocchi e inviato in modo non ordinato e riordinato a destinazione. Così facendo le perdite a burst sarà suddivisa in blocchi non consecutivi: correggere perdite piccole è più semplice di una di grandi dimensioni. Lo svantaggio è lo richiesta di buffering e ritardo di playback

RTP

È un protocollo pubblico di segmentazione e incapsulamento usato soprattutto nelle applicazioni streaming

Streaming Server: permette di evitare HTTP e di scegliere UDP piuttosto che TCP e un protocollo appositamente progettato per lo streaming Real Time Streaming Protocol (RTSP)

RTP definisce un formato standard per i pacchetti multimediali, ma non specifica QoS.

RTCP è usato per inviare statistiche del sistema per ottimizzarne le performance (pacchetti persi, jitter, ultimo numero di sequenza ricevuto ecc.)

APPROCCIO A LIVELLO DI RETE

- Marking: marcare i pacchetti per distinguerli in base alla loro applicazione
- Evitare che un'applicazione non interferisca con un'altra utilizzando più banda di quella prenotata (policing)
- Utilizzare le risorse efficientemente
- Call Admission: accettare un'applicazione solo se si può garantire il servizio

SCHEDULING:

- FIFO = se un pacchetto arriva e trova il buffer pieno viene scartato o sostituito

- Coda a Priorità: ogni pacchetto ha una priorità ed è trasmesso in base ad essa (una coda per diverse classi di priorità)
- Round Robin: una coda alla volta eiquamente
- Weighted Fair Queuing: round robin con servizi differenziati in base al peso della coda per un periodo di tempo limitato.

POLICING: si utilizzano 3 criteri per volutare le risorse da consegnare, flusso medio, flusso massimo, Dimensione Burst

I parametri di un flusso (o di una serie di flussi) sono riassunti nel TSpec mentre i parametri richiesti dall'app. nel RSpec

Token Bucket: in base al TSpec viene "generato" un contenitore di token con 3 parametri, token rate (r), maximum service rate (p ; $p \geq r$) e bucket size (b)
 Il bucket si riempie di r token/s fino a un massimo di b , ogni pacchetto che viene trasmesso consuma un token. Tale procedura può essere combinata con Weighted Fair Queuing: un bucket per coda

RIEPILOGO PROTOCOLLI APPlicativi

I protocolli applicativi definiscono il formato dei messaggi scambiati e il loro significato utilizzando i servizi degli strati inferiori

USER-AGENT: interfaccia tra l'utente e l'applicazione di rete

API: definisce l'interfaccia tra applicazione e strato di trasporto

TCP: orientato alla connessione, trasporto affidabile, controllo di flusso, controllo congestione
Non offre garanzie di banda e ritardo minimi

UDP: non offre gli stessi servizi di TCP ma può essere utile per alcune applicazioni che trasmettono pacchetti singoli o comunque brevi (es. DNS)

Non essendo orientato alla connessione UDP non necessita di stabilirne una (il mittente invia esplicitamente i datagrammi al server). TCP invece necessita handshaking:

sia client che server creano una socket per connettersi l'uno all'altro

HTTP: protocollo per il web di livello applicativo, modello client/server, usa TCP, stateless
la connessione può essere Non-Persistente in cui il server chiude la connessione dopo ogni richiesta (2 RTT: connessione + richiesta) e Persistente in cui la connessione è chiusa dopo un certo tempo di inattività (meno RTT)

Formato Richiesta: Richiesta (metodo, path, versione protocollo) Header (host, tipo connessione ecc) Body

Formato Risposta: Status (status code) Header (content-type ecc) Body (risposta)

WEB CACHING: se ripetiamo una richiesta già fatta, la risposta sarà presa dalla cache perché:

- 1) Più veloce perché più vicina
- 2) Diminuisce il traffico verso server lontani

La cache può essere distribuita sulla rete in cache server o cluster di cache

FTP: modello c/s per il trasferimento di file. Scambio di messaggi di controllo (farsi banda) e invio dati su due connessioni TCP parallele entrambe aperte dal client.
Ogni file ha una sua connessione. FTP utilizza comandi simili alle chiamate REST.

API REST: l'idea è accelerare a servizi web tramite semplici interfacce (GET, POST, DELETE, PUT, HEAD)

Sfiorie a questo si è espansa l'idea dell'XaaS ovvero che qualunque cosa possa essere considerata un servizio web sfiorie al cloud computing.

