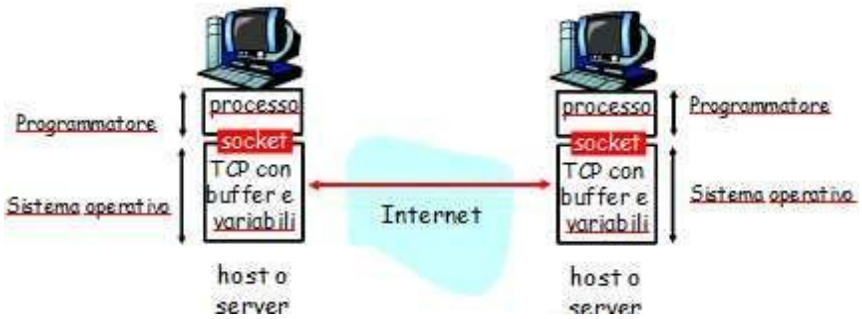
Raccolta di risposte alle domande d'esame del corso di reti

“Se io dò una mela a te e tu dai una mela a me, abb iamo entrambi una mela. Se io dò un'idea a te e tu dai un'idea a me, abbiamo entrambi due idee.”

George Bernard Shaw

Appunti: Livello applicativo

I paradigmi principali del livello applicativo sono il client-server e il p2p; i protocolli principali sono http, ftp, smtp, dns. Un processo è un programma in esecuzione su un host. Sullo stesso host i processi comunicano mediante meccanismi definiti dal SO. Processi in esecuzione su host diversi comunicano mediante meccanismi definiti dal protocollo dello strato di applicazione (application layer protocol).



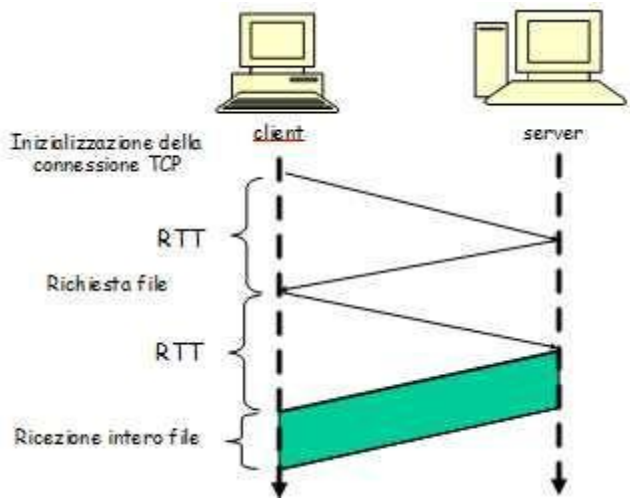
Due processi in esecuzione su due terminale collegati in una rete si individuano grazie a due informazioni: l'indirizzo ip dell'host e il numero della porta dedicata alla applicazione in questione. Il numero di porta (port number) permette l'individuazione da parte dell'host ricevente del processo ricevente locale.

API: application programming interface, definisce l'interfaccia tra applicazione e strato di trasporto. Un Socket è una particolare API detta API Internet, sostanzialmente due applicazioni in esecuzione (processi) comunicano tra loro inviando/leggendo dati nel/dal socket.

I servizi offerti dai protocolli di trasporto Internet:

SERVIZIO TCP:	SERVIZIO UDP:
Orientato alla connessione: richiesto “setup” tra client e server. Trasporto affidabile (reliable transfer) tra processi mittente e ricevente. Controllo di flusso) flow control: il mittente non sommerge il ricevente. Controllo della congestione (congestion control): si limita il mittente quando la rete è sovraccarica. Non offre: garanzie di banda e ritardo minimi.	Trasporto non affidabile tra processi mittente e ricevente Non offre: connessione, affidabilità, controllo di flusso, controllo di congestione, garanzie di ritardo e banda D: perché esiste UDP? Può essere conveniente per le applicazioni (si vedrà più avanti)

Le applicazioni dovranno quindi essere orientate (diciamo così) o alla affidabilità e alla connessione oppure alla velocità e alla larghezza di banda garantita. Nel progetto di una applicazione di rete la scelta dei protocolli giusti sarà una componente importante per il funzionamento dell'applicazione.



HTTP

Un protocollo (di livello applicazione) fondamentale del Web è senz'altro http, il quale permette di richiedere e scaricare file in formato ipertestuale utilizzando il servizio offerto dal protocollo TCP a livello di trasporto e utilizzando la porta 80. HTTP è stateless nella misura in cui non mantiene informazioni di stato (richieste precedenti). La prassi consiste in:

1) una richiesta TCP di connessione da parte del client, indirizzata al web server.
2) il processo sull'host server in ascolto sulla porta 80 riceve la richiesta del client e l'accetta.
3) avviene la richiesta da parte del client contenente l'URL.
4) il server web quando riceve la richiesta costruisce un messaggio di risposta e lo invia al socket.

WEB CACHING(PROXY SERVER)

Il Web caching rappresenta un approccio al problema, quello di accelerare il Web. In generale, le cache (indicate con il simbolo '$', perché si pronunciano come cash) sono memorie locali che contengono copie degli oggetti con un accesso più frequente, per esempio le pagine Web più viste.

Una Cache web, nota anche come server proxy, è un'entità di rete che soddisfa richieste http per conto di un server web d'origine. La cache web ha una propria memoria su disco in cui conserva copie di oggetti recentemente richiesti.

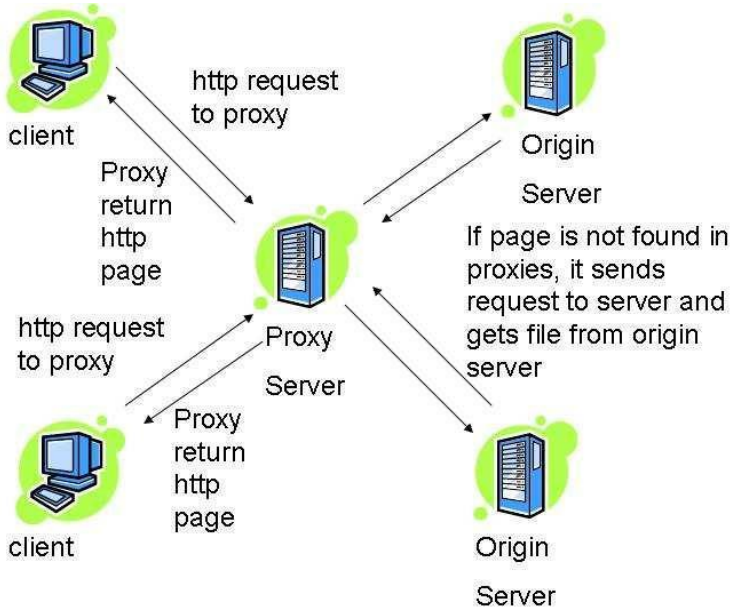
Il browser di un utente può essere configurato in modo che tutte le richieste http dell'utente vengano innanzitutto dirette alla cache web. Una volta configurato il browser ogni richiesta di oggetto da parte del browser viene inizialmente diretta alla cache. Per esempio, supponiamo che un browser stia richiedendo l'oggetto http://www.someschool.edu/campus.gif.
Ecco cosa succede:

1) Il browser stabilisce una connessione TCP con la cache web e invia alla cache una richiesta http per l'oggetto specificato.
2) La cache web controlla la presenza di una copia dell'oggetto memorizzata localmente. Se l'oggetto viene rilevato, la cache web lo inoltra all'interno di un messaggio di risposta http al browser client.
3) Se invece la cache web non dispone dell'oggetto, apre una connessione TCP verso il server di origine, ossia nel nostro esempio www.someschool.edu . Poi la cache web invia una richiesta http per l'oggetto nella connessione TCP. Una volta ricevuta questa richiesta, il server di origine invia alla cache web l'oggetto all'interno di una risposta http.
4) Quando la cache web riceve l'oggetto ne salva una copia nella propria memoria locale e ne inoltra un'altra copia, all'interno di un messaggio di risposta http, al browser client (sulla connessione TCP esistente tra il browser client e la cache web).

Discutiamo altri aspetti del web caching: un cluster di cache permette di alleviare il carico su una singola cache, occorre sapere quale singola cache contiene una specifica pagina, si ricorre quindi

all'instradamento attraverso tabelle hash (cash array routing protocol), l'elaborazione hash assegna ogni pagina ad una singola cache.

Inoltre è possibile implementare il caching cooperativo realizzato mediante un insieme di proxy cache in cooperazione distribuiti su Internet.



I proxy cache possono essere distribuiti in modo gerarchico, dunque avremo le cache di livello (tra di loro superiori o inferiori), le cache di livello superiore mantengono una copia della pagina mentre questa viene inviata ad una cache di livello inferiore. Permette a server con bassa connettività di servire grandi quantità di richieste.

LA POSTA ELETTRONICA: SMTP, POP3, IMAP, HTTP

Le componenti basilari dell'applicazione posta elettronica sono le seguenti:

1) User Agent (Lettore di posta)
2) Server di posta (svolge funzionalità specifiche ma il funzionamento è simile a quello di un server web)
3) SMPT (protocollo di livello applicazione)

Ciascun utente ha una mailbox collocata esattamente nel server di posta, nella quale ci sono i messaggi gestiti dall'utente: inviati e ricevuti.

Il server di posta mittente comunica con il server di posta del destinatario mediante SMTP e senza server intermediari, utilizzando la porta 25 e i servizi offerti da TCP, quindi la comunicazione tra i due server di posta sarà orientata alla connessione e alla affidabilità.

Utente <---------- > server di posta ----------SMTP--------> server di posta <----------> Utente

Vediamo un esempio del funzionamento del protocollo:

1) Il mittente invoca uno user agent per la posta, compone e invia una mail al destinatario.
2) Il mittente invia la mail che si accoda nel server di posta del mittente.
3) Si stabilisce una comunicazione TCP tra i server di posta SMTP
4) Dopo una fase di handshaking SMTP, il server di posta mittente manda effettivamente la mail al server di posta destinatario.
5) Nel server di posta destinatario la mail verrà posizionata nella mailbox dell'utente destinatario.
6) Il destinatario quando lo ritiene opportuno invoca il suo user agent per la posta per leggere il messaggio.

[NB]SMTP e HTTP sono protocolli differenti nonostante entrambi servono a trasferire oggetti tra host della rete: http trasferisce oggetti da server web a client, SMTP trasferisce oggetti da server di posta a server di posta.

PROTOCOLLI DI ACCESSO ALLA POSTA

Come fa un destinatario che esegue un agente utente sul proprio PC locale, a ottenere i messaggi che si trovano nel server di posta del suo provider? Osserviamo che l'agente utente del destinatario non può usare SMTP per scaricare i messaggi dal server perché si tratta di una operazione di pull mentre SMTP è un protocollo di push. Il puzzle viene completato introducendo uno speciale protocollo di accesso alla posta, che trasferisce messaggi dal server di posta al suo PC locale (operazione di tipo pull).

PROTOCOLLO POP3

POP3 usa i servizi TCP e la porta 110, quando la connessione è stabilita avviene l'autenticazione, la transazione e l'aggiornamento. Il funzionamento di questo protocollo si bassa su un paradigma di tipo "autenticazione e scaricamento".

Scarica ed elimina:

1) User Agent elimina la posta nella mailbox dopo averla scaricata.
2) Un utente disperde la posta sui diversi host da cui accede alla mailbox.
3) User Agent permette di creare cartelle, spostare messaggi, effettuare ricerche nei messaggi.

Scarica e conserva:

1) User Agent conserva la posta nella mailbox dopo averla scaricata.
2) Utente può leggere gli stessi messaggi da macchine diverse.
3) POP3 stateless, non permette di strutturare i messaggi indirectory.

PROTOCOLLO IMAP

IMAP permette di gestire cartelle di posta remote come se fossero locali, IMAP deve mantenere una gerarchia di cartelle per ogni utente, permette di scarica solo parti del messaggio: intestazione, intestazione file MIME multipart, messaggi di dimensione piccola per utenti a banda limitata.

USER AGENT -----SMTP---- > server -----SMTP ----- > server <--------pop3, imap------- USER AGENT

POSTA BASATA SUL WEB

E' oggi in costante crescita il numero di utenti che inviano posta e accedono alle proprie e-mail tramite browser web. Tramite questa tecnologia è possibile inviare e ricevere mail sulle caselle di posta di Hotmail, Google, Yahoo e altri.

BROWSER WEB -----http---- > server -----SMTP ----- > server <--------http------- BROWSER WEB

DNS: DOMAIN NAME SYSTEM

Dobbiamo porci il problema di come identificare gli host nella rete, si intuisce facilmente che i calcolatori elettronici non si chiamano per nome come fanno gli uomini ("Marco", "Alberto", "Alice", etc.).

I calcolatori si identificano mediante l'indirizzo IP, quindi, da qualche parte nel mondo c'è qualcuno che ha il compito di tenere aggiornata un database dove sono memorizzate le coppie nome-indirizzo IP, questo qualcuno sono i server DNS.

DNS è un protocollo di livello applicativo che utilizza i servizi offerti da UDP, il suo scopo è quello di dirigere le richieste dei client al server più vicino.

Questi server rispettano una gerarchia, oltre ad essere semplicemente distribuiti. Il motivo per cui sono distribuiti (ce ne sono tanti), è importante:

- Minore tolleranza ai guasti.
- Traffico eccessivo.
- Database centrale troppo distante in molti casi.
- Scarsa scalabilità.
- Autorizzazione ed accesso per registrare nuovo host.

Caching e aggiornamento nei server DNS è pratica diffusa, quando un qualsiasi name sever apprende una traduzione la memorizza localmente (caching). Le traduzioni memorizzate localmente (cache entries) scadono (time-out) dopo un certo tempo (di solito un paio di giorni).

DOMANDE D'ESAME sul livello applicativo

Quali sono le differenze tra il paradigma "Client-Server" e il "peer to peer"?

Ruoli dei dispositivi
Nel client-server è noto a priori chi offre e chi richiede un servizio.
Nel p2p tutti gli host (peer) sono client e server nello stesso momento e quindi sono sullo stello livello gerarchico.
Localizzazione risorse
Nel C/S le risorse necessarie per fornire il servizio sono concentrate nel server ovvero abbiamo una localizzazione centralizzata. Questo implica che : 1) all'aumentare del numero di richieste, le prestazioni peggiorano 2) in caso di guasto del server il servizio non è più disponibile.
Nel p2p le risorse sono sparse nei vari peer ovvero abbiamo una localizzazione completamente distribuita. Questo implica che : 1) se un peer si disconnette il servizio continua ad essere fornito 2) la qualità dipende dalle risorse messe a disposizione da ogni peer.
Reperimento risorse
Nel C/S la risorsa è trovata facilmente.
Nel P2P gli utenti devono effettuare una fase di ricerca dei contenuti . Questo implica che ogni peer deve : 1) inserirsi nella rete conoscendo almeno un peer connesso (bootstrap) 2) conoscere altri peer connessi attraverso operazioni di Ping/Pong e stabilire connessioni con alcuni di essi.

Quali sono i vantaggi dell'architettura di Gnutella rispetto a quella di Napster

L'architettura NAPSTER è a DIRETORY CENTRALIZZATA cioè ha una localizzazione centralizzata e un trasferimento tra peer distribuito e presenta diverse limitazioni tra cui: un Unico punto di guasto, un Collo di bottiglia prestazionale e i Diritti d'autore.
L'architettura GNUTELLA *risolve le limitazioni di Napster* perchè è QUERY FLOODING cioè rappresenta un approccio completamente distribuito per la localizzazione e trasferimento dei contenuti. I peer sono connessi mediante una rete logica (overlay network) in cui ogni collegamento è una connessione tcp.

Il Query Flooding consiste nella propagazione di una richiesta lungo la rete fino a quando non si arriva ad un peer che possiede il file ; a quel punto viene generata una query hit che torna indietro fino al peer d'origine.

A cosa serve il TTL (time to live) nella fase di ricerca in una rete p2p (peer to peer)? Per quale motivo è importante?

Nelle reti p2p decentralizzate (es. GNutella) si presenta un problema di scalabilità dovuto all'utilizzo del query flooding che genera una significativa quantità di traffico nella rete Internet.
Per risolvere il problema si ricorre al QUERY FLOODING A RAGGIO LIMITATO ovvero si limita l'inondazione di richieste ad un raggio stabilito grazie ad un campo aggiuntivo detto TTL.
Ogni volta che il messaggio di richiesta viene inoltrato ad un altro nodo, il valore del TTL (time-to-live) viene decrementato di uno. Se il TTL > 0 allora il messaggio viene inoltrato altrimenti non viene preso in considerazione.
Va detto che questo tipo di soluzione riduce l'intensità di traffico ma ha lo svantaggio di ridurre la probabilità di trovare un file.

Quali sono le principali differenze tra il protocollo HTTP e il protocollo SMTP?

Entrambi utilizzano connessioni persistenti TCP, su porte diverse naturalmente.

Tipo di informazioni

HTTP trasferisce oggetti in generale.
SMTP trasferisce solo oggetti di tipo posta elettronica.

Funzione

HTTP è un "pull protocol" e viene utilizzato per scaricare oggetti da server web.
SMTP è un "push protocol" e serve a caricare oggetti su un server di posta.
(user agent --- smpt ---> server di posta di invio --- smpt ---> server di posta di ricezione)

Divisione informazioni

HTTP incapsula ogni oggetto in un messaggio differente.
SMTP mette tutti gli oggetti in un unico messaggio.

Conversioni

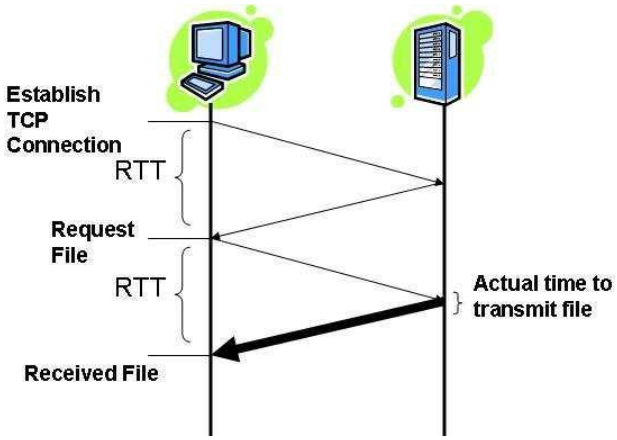
HTTP incapsula il tutto in pacchetti TCP senza effettuare nessuna conversione.
SMTP converte tutto in ASCI a 7 bit e sfrutta il protocollo MIME (multipupose internet mail extension) per il file multimediale e poi incapsula in pacchetti TCP.

Quali sono le differenze tra HTTP e SMTP nella gestione di un documento che contiene sia testo che immagini?

SMTP converte tutto in ASCI a 7 bit e sfrutta il protocollo MIME per il file multimediale e poi incapsula in pacchetti TCP, mentre HTTP incapsula il tutto in pacchetti TCP senza effettuare nessuna conversione.

Per quale motivo una connessione http permanente sfrutta meglio il protocollo TCP.

Il tcp è caratterizzato da un handshaking iniziale (che richiede del tempo : 1RTT) per instaurare la connessione tra le entità coinvolte.



Http persistente sfrutta meglio il protollo tcp perché, al contrario di quella non persistente, la connessione viene mantenuta attiva al termine del trasferimento di un oggetto.
Questo permette di trasferire più oggetti in un'unica connessione e quindi di risparmiare RTT ed sfruttare meglio lo slow start.

Numericamente (tralasciando lo slow start) :
Persistente = 1RTT connessione + (1RTT richiesta + T di trasf. dati)* ogni oggetto
Persistente con pipe = 1RTT connessione + 1RTT richiesta tutti gli oggetti + (T di trasf. dati)* ogni oggetto
Non Persistente = (1RTT connessione + 1RTT richiesta + T di trasf. dati)*ogni oggetto

Per quale motivo in http sono necessari i cookie? Descrivere cosa sono e come si utilizzano attraverso un semplice esempio.

Alcuni web server hanno la necessità di mantenere le informazioni di stato di una connessione.
Esempio : autenticazione , preferenze utente

Dato che htpp è stateless, ovvero non offre questo tipo di servizio, vengono utilizzati i cookie.

Un cookie è un informazione aggiuntiva (es.codice) che viene fornita dal server e memorizzata nel client.

Per il funzionamento vedi lo schema a lato.

Un esempio dell'utilizzo dei cookie è il carrello della spesa virtuale in siti commerciali oppure la preferenza sul numero di risultati da visualizzare con una ricerca su Google.

Per quale motivo i protocolli di tipo stop and wait non sono indicati in link ad alta latenza ? Si chiarisca il concetto con un esempio caratterizzato da un link da 1Gbps e un RTT di 0.25s e dimensioni dei pacchetti da 3kb. Quale frazione della banda viene effettivamente usata?

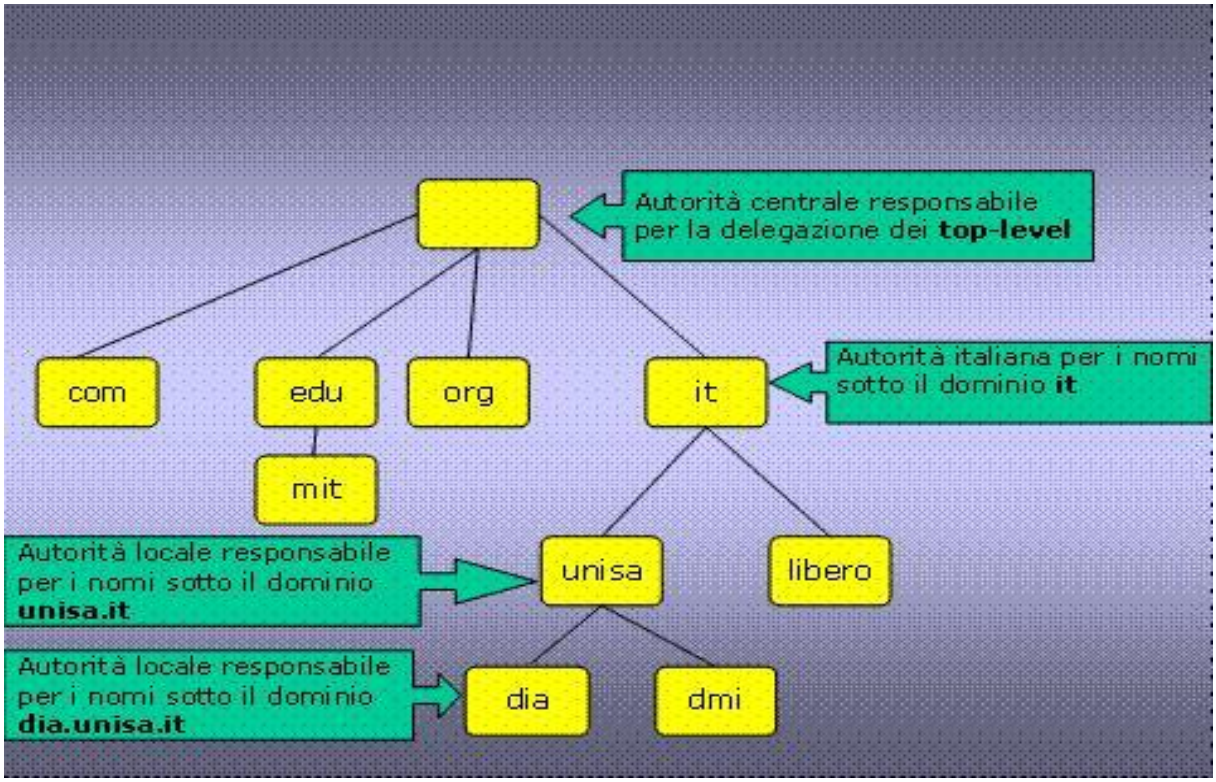
Un protocollo stop and wait è molto semplice da implementare ma è inefficiente perché bisogna attendere l'ack di un pacchetto prima di inviare il successivo.
E' particolarmente inefficiente in linee ad alta latenza con RTT >> d, *dove d è il tempo richiesto a inviare o ricevere un pacchetto*.

Abbiamo:
RTT = 0.25 s
Link con rate R = 10^9 b/s
Dimensione pacchetto L = 3*10^3 bit
d = L/R = 3x10^3 / 10^9 = 3x10^-6 s.

Per inviare ogni pacchetto si attende quindi Ttot = 0.25s + 3x10^-6 s = circa 0,25s
Quindi in un secondo inviamo quattro pacchetti ovvero abbiamo un rate effettivo = 12 kbps.

% banda effettivamente utilizzata = [12x10^3 / 10^9] * 100 = 1,2 * 10^-5 = 0.0012% che è pochissimo.

Descrivere la struttura gerarchica dei DNS e fare un esempio di schema ricorsivo.



Dalla figura precedente è evidente che :
C'è il server radice al quale fanno riferimento vari domini. A loro volta ci sono sottodomini che fanno riferimento alle autorità garanti del dominio del livello superiore.

Esempio schema ricorsivo
1 radice
1.1 com
1.1.1 google.com
1.1.2 msn.com
1.2 org
1.2.1 altervista.org
1.3 net
1.3.1domino.net

Google desidera indirizzo ip di domnio.net
risale a 1.1 quindi com
risale a 1 radice
che va in1.3 net
E trova 1.3.1 domino .net
viene restituito a .net, che lo passa a radice, e lo comunica .com che lo porta google.com

Si descrivano i passi e gli strumenti necessari per inviare una mail falsa a nome di un'altra persona semplicemente utilizzando il protocollo SMTP e si accenni alle possibili contromisure.

Con SMTP il client, una volta collegato al server, comunica tramite il comando MAILFROM< address@mail.it> il mittente.

Con il comando telnet ServerName 25 è possibile comunicare, da linea di comando, direttamente col server di posta senza un agent intermediario: quindi per mandare una mail falsa basta utilizzare il comando MAILFROM mettendo un indirizzo email falso..

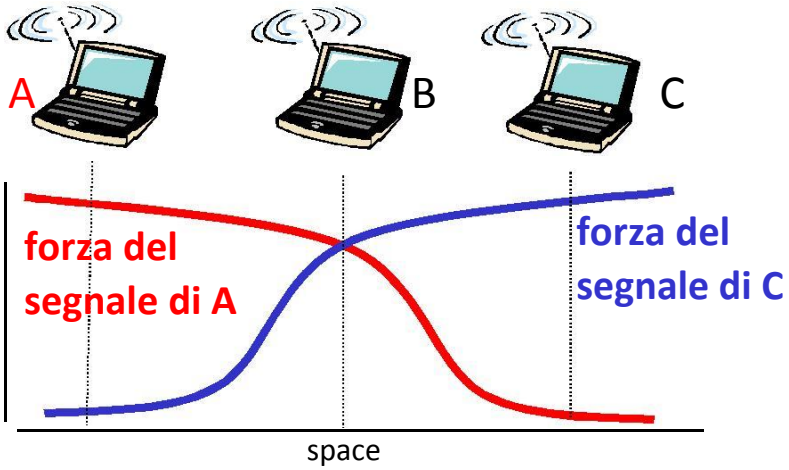
Per ovviare a questo problema, si fa uso del protocollo PGP che garantisce l'autenticità tramite inserimento di un diggest nel messaggio.

Wireless e reti mobili (datalink)

Per quale motivo i pacchetti RTS hanno una minor probabilità di collisione rispetto ai pacchetti standard?

I pacchetti RTS (Request-To-Send) hanno una minor probabilità di collisione rispetto ai pacchetti standard perchè sono di dimensioni ridotte.

Per quale motivo nelle reti cablate non c'è il problema del terminale nascosto ? Come si risolve tale problema nelle reti wireless ?



Nelle reti cablate non c'è il problema del terminale nascosto perché grazie alle caratteristiche fisiche del filo che collega i vari dispositivi, è possibile implementare un protocollo che fa collision detection (esempio csma/cd). Ogni terminale sa esattamente se qualcun altro sta trasmettendo o meno.

La stessa cosa non può essere implementata nelle reti wireless proprio a causa del problema del terminale nascosto.
Supponiamo di trovarci nella condizione in cui i terminali A e C sono abbastanza lontani da non poter percepire la presenza l'uno dell'altro. Se per esempio A sta trasmettendo verso B , C non ha modo di rendersi conto di questo ed inizia anche lui a trasmettere verso B. A questo punto è facile capire che in B si verifica una collisione che compromette entrambe le comunicazioni.

Una soluzione a tale problema consiste nell'introduzione di

- un sistema di prenotazione implementato con l'introduzione di due nuovi pacchetti: Request-To-Send (RTS) e Clear-to-Send (CTS)
- meccanismo di backoff

Quando il trasmittente vuole inviare un frame dati, aspetta DIFS secondi ed invia un frame RTS all'Access Point indicando il tempo totale richiesto per la trasmissione (compreso il tempo di ricezione dell'ACK). Se si verifica una collisione oppure non viene ricevuto l'ack si applica un backoff casuale e si fa un nuovo tentativo.

Quando l’Access Point riceve un frame RTS, aspetta un tempo pari a SIFS secondi e risponde in broadcast con un frame CTS contenente a sua volta la durata complessiva della prenotazione. In questo modo tutti i terminali sono consapevoli che è in atto una trasmissione di cui essi possono anche non essere in ascolto.

Che tipo di meccanismo di instradamento mobile IP usereste nell'ambito della LAN del secondo piano del dipartimento?

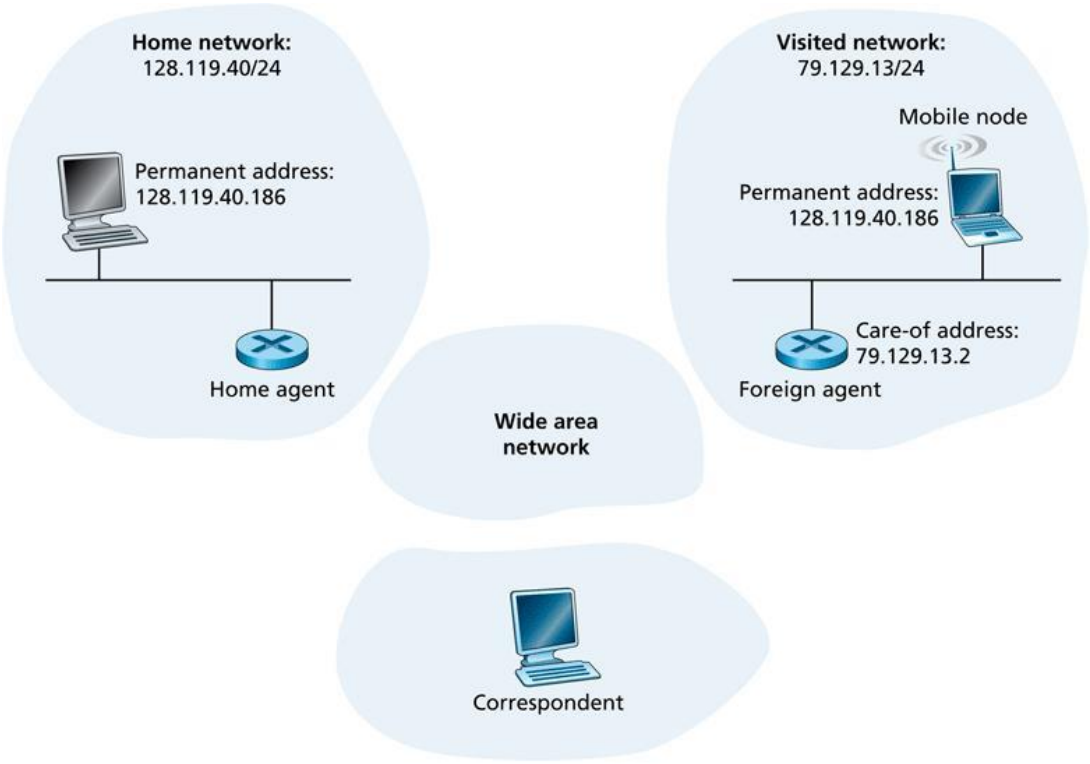
Instradamento diretto, poiché con instradamento indiretto si incontrerebbe il problema del routing triangolare.

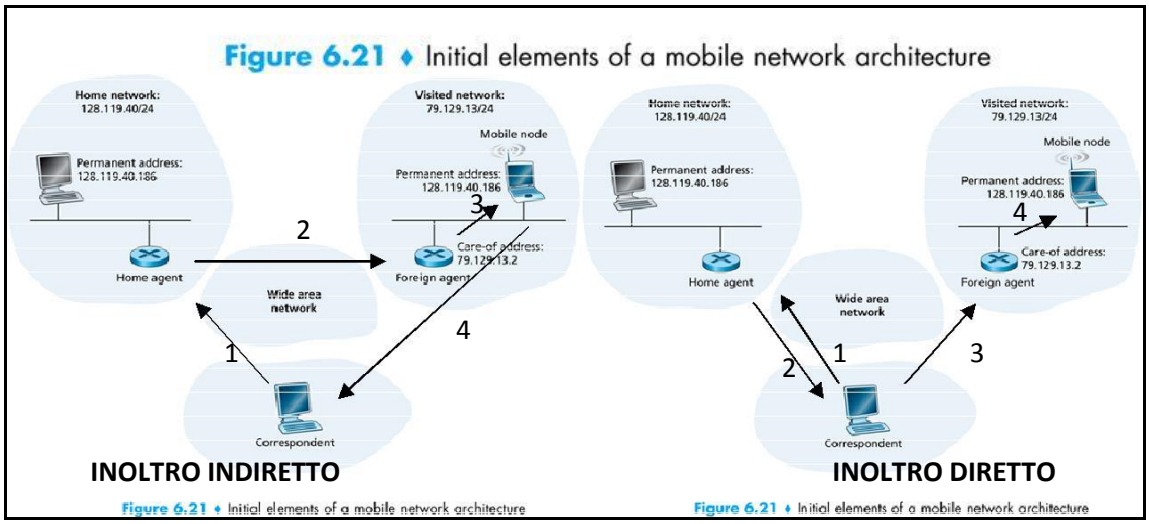
In cosa consiste il problema del “triangolo d’instradamento” nelle reti mobili?

E’ un’efficienza tipica dell’instradamento indiretto. Ogni datagramma indirizzato al nodo mobile deve essere instradato all’home agent che poi lo inoltrerà alla foreign agent, anche in presenza di rotte più efficienti tra il corrispondent e il nodo mobile.

Esempio : immaginiamo un professore in visita presso un suo collega in un’altra università. I due sono seduti sulla stessa scrivania e si scambiano dati attraverso la rete. In tal caso i datagrammi dell’uno sono instradati all’home agent dell’altro e poi rimbalzati nella prima rete.

Si consideri la figura 6.21 e la si completi illustrando i principali flussi di informazione relativi all'instradamento diretto ed indiretto di un utente mobile.





Reti multimediali e applicazioni di rete real-time

Si assuma che in coda ad un router siano presenti i pacchetti p1,p2,p3,p4,p5. I pacchetti p1 e p4 sono di una classe C1 di priorità inferiore rispetto alla classe C2 di p2,p3 e p5. Qual è la sequenza di output dal router per gli scheduling: FIFO, PRIORITA' e ROUND ROBIN? Cosa succede nel caso in cui lo scheduling sia di tipo WFQ se il peso della classe C1 è 1 e il peso di C2 è 2?

FIFO: p1 -> p2 -> p3 -> p4 -> p5
PRIORITA: p2 -> p3 -> p5 -> p1 -> p4
ROUND ROBIN: p2 -> p1 -> p3 -> p4 -> p5

Per quanto riguarda la seconda domanda (WFQ), i pacchetti vengono trasmessi contemporaneamente ma ad un rate diverso (la classe con peso maggiore avrà un rate di uscita più alto rispetto alla classe con peso minore, la banda totale viene suddivisa)

Sapendo che w1 e w2 sono i pesi delle rispettive classi C1 e C2

$Rate_{C1} = Rate_{TOT} * (w1/w1+w2) = (1/3)Rate_{TOT}$
$Rate_{C2} = Rate_{TOT} * (w2/w1+w2) = (2/3)Rate_{TOT}$

Si consideri un percorso fatto di 3 router che accettano pacchetti di dimensione 1Kb appartenenti a due classi di priorità P1 e P2. Tutti i router implementano WFQ e hanno la medesima banda in uscita = 10Mbps. La dimensione delle code assegnate a P1 e P2 è di 10 pacchetti

- **Assegnare il peso W1 alla classe P1 affinché il ritardo massimo di un suo pacchetto nell'attraversare i 3 router sia di 150ms**
- **Qual è in questo caso il ritardo massimo di un pacchetto che appartiene a P2?**

R = 10 Mbps = 10^7 b/s
Pck = 10^3 bit
Coda = 10pck = 10^4 bit

$$\frac{\text{Buffer tot}}{\text{Rate P1}} = \frac{3 * 10^4}{10^7 * ((w1/(w1+w2))} = 0{,}150 = \text{Ritardo massimo totale}$$

$10^{-3} * ((w1+w2)/w1) = 0{,}05$

$10^{-3} * (w1+w2) = w1 * 0{,}05$

$w1 + w2 = w1 * 50$

$w2 = w1 (50-1) = 49w1$

se scelgo w1= 1 allora w2=49

Per la seconda domanda applico la stessa formula di prima ma con al denominatore RateP2 e come incognita il ritardo massimo ovvero:

Peso P2 = w2/(w1+w2) = 49/50 = 0,98
Ritardo Massimo = 3* 10^-3 / 0,98 = 3ms circa

Si consideri un leaky bucket di dimensione del secchio =10 pacchetti e token rate = 20 pacchetti al secondo. Nell'ipotesi che il traffico in entrata sia caratterizzato da 1 pacchetto ogni 2mS, quanti pacchetti escono al più dal leaky bucket?

b = dimensione del bucket = 10 ; r = token rate = 20 pck/sec ;
p = pacchetti in entrata = 500 pck/sec

Numero di pacchetti in uscita = rt + b

- Se partiamo con il secchio pieno abbiamo in uscita 30 pck nel primo secondo e poi sempre 20pck/sec
- Se partiamo con il secchio vuoto abbiamo in uscita sempre 20 pck/sec

[NB]Nel leaky bucket il numero dei pacchetti uscenti è indipendente dal tasso dei pacchetti in entrata.

Si consideri un percorso fatto di 2 router. Il primo implementa la politica weighted fair queuing (WFQ) con due classi di priorità P1 di peso 2 e P2 di peso 4, entrambe le classi di priorità hanno un buffer di 100 pacchetti. Il secondo implementa la semplice politica FIFO, ha un packet rate in uscita di 1000 pacchetti per secondo ed un buffer di 200 pacchetti. Qual è il packet rate che deve avere il primo router affinché il massimo ritardo di un pacchetto con priorità P1 che attraversa entrambi i router sia 500ms?

Rate P1 = RatePrimoRouter * (w1/(w1+w2)) = RatePrimoRouter * 1/3

$$\frac{\text{Buffer P1}}{\text{Rate P1}} + \frac{\text{Buffer R2}}{\text{RateR2}} = \text{Ritardo massimo totale}$$

$$\frac{100}{\text{RatePrimoRouter} * 1/3} + \frac{200}{1000} = 0{,}5$$

300 / RatePrimoRouter = 0,3

RatePrimoRouter = 1000 pck/sec

Si consideri un percorso fatto di 2 router con rate di 600Kbps ciascuno che accetta pacchetti di dimensione 1 Kb in ingresso a due distinte code. Il traffico in ingresso alle code è regolato da un sistema di leaky bucket con r0=10 token per secondo e B0=10 pacchetti e r1=30 token per secondo e B1=30 pacchetti. Infine la politica di scheduling è WFQ con peso w0=2 per la coda 0 e w1=4 per la coda 1.
1)Qual è la dimensione minima delle code del router affinché nessun pacchetto vada perduto ??
2)Qual è il massimo ritardo end-to-end?

??

Un servizio real-time è caratterizzato da un R_spec con banda pari a 100Kbps e tempo di slack pari a 0.3s. Si considerino pacchetti di dimensione 1Kb e tre possibili percorsi alternativi costituiti da due router identici, che assegnano al servizio rispettivamente:
1.Banda in uscita 70Kbps, buffer 5 pacchetti
2.Banda in uscita 200Kbps, buffer 10 pacchetti
3.Banda in uscita 200Kbps, buffer 100 pacchetti
Quale dei percorsi garantisce il rispetto del R_spec? Giustificare la risposta.

1)Non può essere perché se la R_spec comunica una velocità di invio di pacchetti a 100Kbps, il router deve essere in grado di smaltire questi pacchetti. Se il rate di uscita del router è inferiore a quello specificato nella R_spec, il buffer si riempirebbe subito, e c'è possibilità di perdita di pacchetti.
2) abbiamo un ritardo rit2=2*10kb/(200kbps)=0,1sec
3) abbiamo un ritardo rit3=2*100kb/(200kbps)=1 sec
Considerando che il tempo di slack è pari a 0,3 la risposta è la 2 essendo rit2<slack

Si consideri il seguente traffico nel tempo (t):

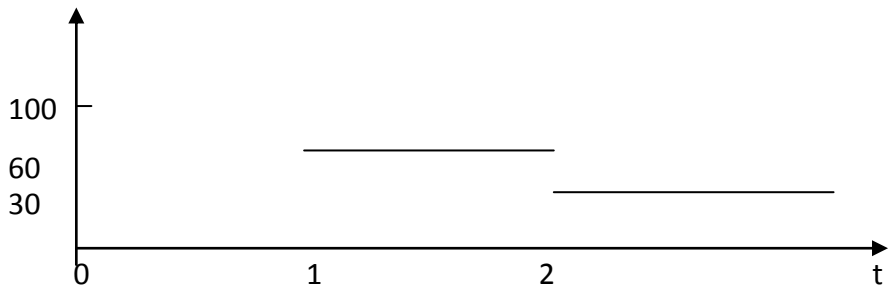
- **0<=t<2 secondi 100 pacchetti al secondo**
- **t=2 secondi 1000 pacchetti istantanei**
- **t>2 secondi 35 pacchetti al secondo**

Se questo traffico arriva in ingresso ad un leaky bucket di parametri b=100 pacchetti e r = 30 gettoni per secondo, come viene modificato in uscita? Disegnare l'andamento nel tempo.

Numero pacchetti in uscita = b + r*t

Considerando il bucket inizialmente pieno con 100 token allora abbiamo:
0 → 1 = 100 pacchetti (tutti usciti istantaneamente grazie al bucket pieno)
1 → 2 = 60 pacchetti (grazie ai 30 token accumulati nel sec precedente + quelli in questo sec)
in t = 2 i 1000 pacchetti istantanei vengono persi tutti dato che ho il bucket vuoto
2 → infinito = rate costante di uscita di 30 pck/sec

L'andamento nel tempo è il seguente :



Per quale motivo l'header dei pacchetti RTP contiene un numero di sequenza ed una marcatura temporale? Discutere in che modo si potrebbe realizzare un protocollo affidabile usando solo datagrammi UDP

I *Numeri di sequenza* sono utilizzati dal ricevente per rilevare perdite e per ricostruire la sequenza d'invio dato che UDP non assicura che i pacchetti arrivino con lo stesso ordine con cui sono stati inviati.
Il T*imestamp* è utilizzato per scartare i pacchetti che hanno subito un ritardo che non può essere tollerato. Per esempio nella telefonia scartiamo pacchetti con più di 400ms di ritardo.

Udp affidabile realizzato con :

- Buffer di dati nel client
- Timestamp, numeri di sequenza e ritardo di playout per combattere lo jitter
- Ridondanza per ridurre perdita pacchetti

NEW - Per quali motivi il protocollo UDP è di norma più indicato per le applicazioni real-time ?

Le applicazioni real-time devono essere in grado di rispettare tempi di risposta prestabiliti. Anche se non garantisce la consegna in ordine e senza perdite è più indicato il protocollo udp perché :

- è senza connessione quindi non ha bisogno dell' handshaking iniziale
- i dati elaborati dal processo vengono spediti immediatamente indipendentemente dalla congestione della rete

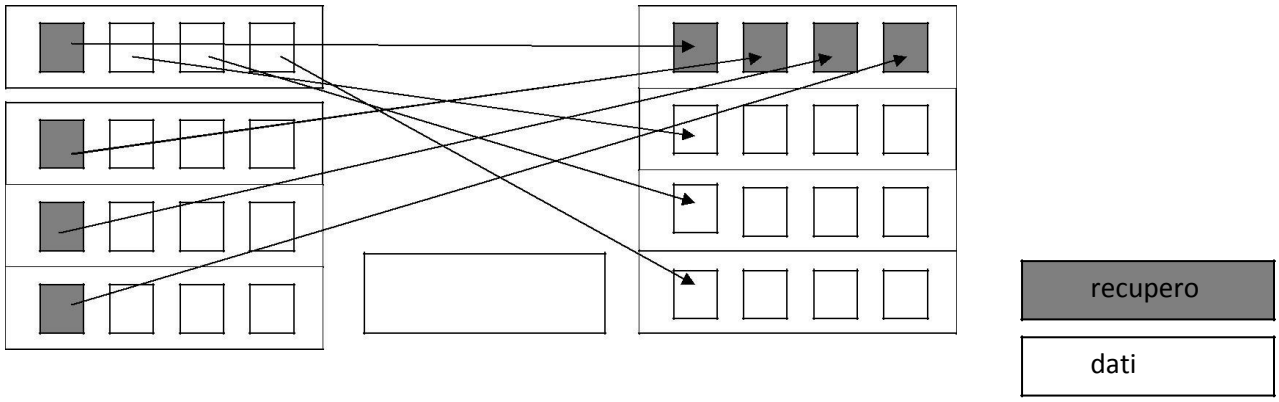
ESERCIZI SU RECUPERO ERRORI (FEC, interleaving, Reed Solomon)

Si consideri un codice FEC che agisce su blocchi fatti di 3 pacchetti +1 di recupero. Progettare e disegnare uno schema di interleaving affinché l'uso congiunto di codice FEC ed interleaving sia in grado di tollerare la perdita di un intero blocco di interleaving

E' sufficiente mettere i pacchetti di recupero tutti nello stesso blocco, così posso perdere un blocco intero a burst.

Esempio
Sequenza invio : [R1,R2,R3,R4] [1,4,7,10] [2,5,8,11] [3,6,9,12]
supponiamo che il secondo blocco vada perso
Allora posso recuperarlo :
1 = FEC (R1,2,3) , 4 = FEC (R2,5,6) , 7 = FEC (R3,8,9) , 10 = FEC (R4,11,12)

Schematicamente :



Si consideri un codice a correzione di errore che introduce un pacchetto di recupero ogni due pacchetti di dati. Si consideri inoltre una codifica interleaving che opera su blocchi formati ognuno da tre pacchetti, due di dati ed il corrispondente pacchetto di recupero. Si determini la massima lunghezza di un singolo burst di perdita di pacchetti a cui la codifica è tollerante, il minimo ritardo di riproduzione introdotto dalla codifica e lo spreco di banda introdotto.

(schema simile all'esercizio nella pagina precedente)

Abbiamo :
Sequenza invio : [R1,R2,R3] [1,4,7] [2,5,8] [3,6,9]
supponiamo che il secondo blocco vada perso
Allora posso recuperarlo :
1 = FEC (R1,2,3) , 4 = FEC (R2,5,6) , 7 = FEC (R3,8,9)
E' evidente che con questa tecnica possiamo recuperare al più un blocco intero ovvero è tollerante alla perdita di un burst di 3 pacchetti .
Lo spreco di banda è di 1/4 perché introduciamo un nuovo blocco composto da 3 pacchetti di recupero.
Il minimo ritardo di riproduzione = tempo per l'interleaving + tempo trasmissione di tutti i blocchi + tempo per riordino + tempo eventuali correzioni dovute a perdite

Crittografia e sicurezza nella rete

Essenzialmente abbiamo due tipi differenti di crittografia:

SIMMENTRICA (chiave privata)
La chiave di codifica è uguale alla chiave di decodifica ed entrambe sono segrete.

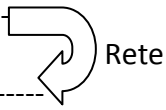
BOB [testo] ----- cod. chiave privata -----> [CIFRATO] ----------

ALICE [testo] <----- dec. chiave privata ----- [CIFRATO] <----------

ASIMMETRICA (chiave pubblica)
Ogni utente ha una chiave privata segreta e non deducibile dalla chiave pubblica nota a tutti.
La chiave privata viene utilizzata per decodificare messaggi cifrati da altri utenti con la propria chiava pubblica.

BOB [testo] ----- cod. chiave pubblica alice -----> [CIFRATO]----------

ALICE [testo] <--- dec. chiave privata alice -------- [CIFRATO] <----------



DOMANDE D'ESAME sulla sicurezza e sulla crittografia

In cosa consiste il paradosso del compleanno e per quale motivo è importante per la sicurezza delle funzioni hash? Spiegarlo attraverso un esempio.

Il paradosso del compleanno afferma che la probabilità che in un insieme di persone ce ne siano almeno due che festeggiano il compleanno nello stesso giorno è molto più alta di quella che potrebbe sembrare intuitivamente, tanto da sembrare paradossale.
Esempio : in un gruppo di 23 persone la probabilità è superiore al 50%.

Il paradosso è importante nel calcolo dell'impronta di un documento tramite funzione hash perché è usato per quantificare la sua validità.
Esempio : DES a 64 bit
Dato un messaggio M, per alterarlo in M' ed ottenere la stessa firma ho bisogno di generare 2^64 messaggi M' (abbastanza laborioso).
Il discordo cambia alterando sia M che M' con piccoli cambiamenti . In questo caso ho bisogno di generare un totale di 2^33 messaggi per avere una probabilità di successo maggiore del 50% (troppo facile).

Su cosa si basa la sicurezza di un algoritmo a chiave pubblica? Descrivere brevemente gli aspetti principali di RSA.

L'algoritmo di basa sul fatto che :

- testo cifrato con la chiave pubblica del destinatario può essere decifrato solo con la chiave segreta che possiede solo quest'ultimo.
- chiave privata non è deducibile da quella pubblica

[NB] E' la CA a garantire la sicurezza della corrispondenza tra utente e sua chiave pubblica

RSA :

- p e q numeri primi grandi
- n = p*q , z = (p-1)(q-1)
- scelgo e<n tale che MCD(e,z)=1
- scelgo d tale che ed-1 sia divisibile per z

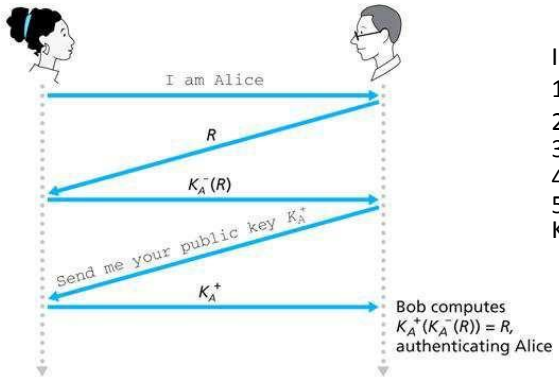
Chiave pubblica = (n,e) , chiave privata = (n,d) . RSA funziona perché non esiste (per ora) nessun algoritmo polinomiale che fattorizzando n ed e (della chiave pubblica) mi permetta di risalire alla chiave privata.

Assumete che un KDC server o un CA server si guasti. Chi può comunicare in modo sicuro e chi no nei due casi?

Nel KDC le chiavi segrete sono di sessione quindi vengono usate in quella sessione e poi buttate. Questo implica che se KDC si guasta non c'è modo di scambiare in modo sicuro le chiavi simmetriche e quindi di comunicare in modo sicuro.

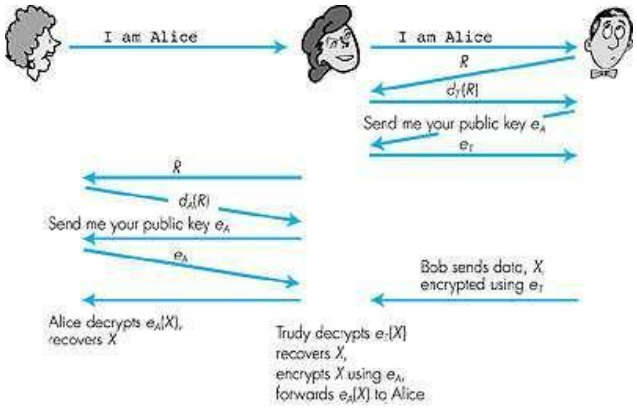
Nel caso di guasto della CA abbiamo la possibilità di comunicare ugualmente in modo sicuro dato che la chiave pubblica l'abbiamo già ottenuta e l'associazione tra utente e chiave pubblica non cambia così spesso.

Per quale motivo il protocollo rappresentato in figura 8.19 è da ritenersi insicuro? A che tipo di attacchi è soggetto? Come lo si può rendere sicuro? In cosa consiste l'attacco man in the middle, e come può essere evitato?



In figura abbiamo il protocollo ap5.0
1) Alice contatta Roberto
2) Roberto gli manda un nonce R
3) Alice gli manda $KA_{pri}(R)$
4) Roberto si procura la chiave pubblica di Alice KApub
5) Roberto calcola $KA_{pub}(KA_{pri}(R))$ e verifica che $KA_{pub}(KA_{pri}(R))=R$

Figure 8.19 ♦ Protocol *ap5.0* working correctly



Il protocollo ap5.0 è da ritenersi insicuro perché è soggetto all'attacco "man in the middle " in cui un intruso può interporsi nella comunicazione tra due soggetti in modo del tutto trasparente.

Infatti Trudy può fingere di essere Alice con Bob e di essere Bob con Alice procedendo in questo modo:

1) Trudy contatta Bob
2) Bob manda un nonce R ad Alice e Trudy lo intercetta.
3) Trudy manda $KT_{pri}(R)$ a Bob
4) Bob richiede la chiave pubblica di Alice KA_{pub}, ma questa richiesta viene intercettata da Trudy che gli manda la sua: KT_{pub}
5) Bob senza saperlo calcola $KT_{pub}(KT_{pri}(R))$ e verifica che $KT_{pub}(KT_{pri}(R))=R$

Lo stesso procedimento può essere ripetuto da Trudy con Alice. Per Alice e Roberto l'autenticazione è andata a buon fine ma in realtà non è così.
Possiamo rendere sicuro il protocollo di autenticazione ap5.0 tramite la certificazione della chiave pubblica (per esempio con CA)

Per quale motivo per firmare un documento è necessario prima ottenerne l'impronta attraverso una funzione di hash? Quali caratteristiche deve avere una funzione di hash?

Firmare digitalmente un documento con RSA può essere un compito molto oneroso se il file è di grandi dimensioni. Per risolvere questo problema è stata introdotta l'impronta digitale anche detta hash. L'idea alla base di questa soluzione è la seguente:
Dato un documento, attraverso una funzione hash si ottiene l' hash (un codice in teoria univoco) che lo identifica e si applica la chiave segreta solamente ad esso e non a tutto il documento.

La funzione crittografica di HASH deve avere 3 caratteristiche fondamentali:

1. Deve essere estremamente semplice calcolare un HASH da qualunque tipo di messaggio.
2. Deve essere estremamente difficile risalire al testo da un dato HASH.
3. Deve essere estremamente improbabile che due messaggi differenti, anche se simili, abbiano lo stesso HASH.

Un compratore si rivolge ad un sito di commercio elettronico. Quali strumenti garantiscono il compratore dell'identicità del sito di commercio elettronico e viceversa? Quale protocollo permette di mantenere il numero di carta di credito del compratore ignoto al sito di commercio elettronico?

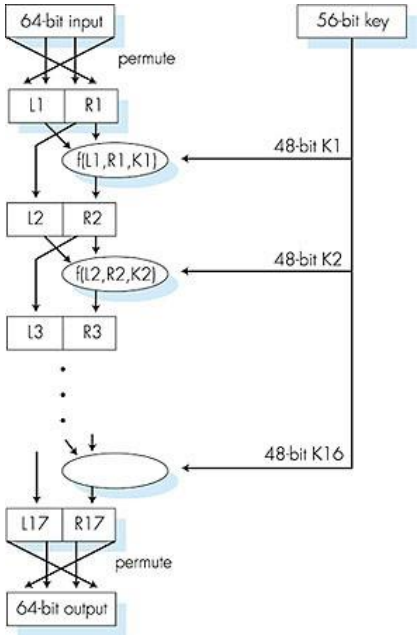
Si potrebbe utilizzare la certificazione delle chiavi e i nonce, il compratore manda un nonce r1 con la chiave pubblica certificata del sito di commercio elettronico. Il sito cifra r1 e r2 con la chiave pubblica certificata del compratore. Quando il compratore riceve K(r1,r2) risalendo a r1 autentica il sito. Il sito, con le stesse modalità risale a r2 e autentica il compratore.

Il protocollo per mantenere ignoto il numero della carta di credito al sito di commercio elettronico è SET, si basa su SSL.

Dovete realizzare un sistema per l'autenticazione in una rete aziendale non connessa ad Internet a cui accedono soli 5 utenti. Che soluzione proporreste? Motivare la risposta.

Poiché ci sono solo 5 utenti, io lascerei stare l'autenticazione intesa in senso classico ed utilizzerei un sistema con chiavi private. Infatti il solo fatto che due utenti conoscano la chiave privata da loro condivisa, ci assicura l'autenticazione. L'unico problema delle chiavi private è lo scambio delle chiavi e il numero delle chiavi: problema aggirabile perché, appunto, ci sono solo 5 utenti!

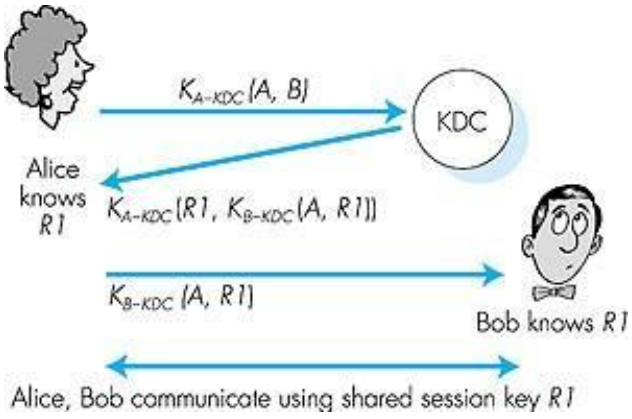
Descrivere graficamente un metodo per il calcolo dell'impronta di un documento che utilizza DES (e calcola impronte di 64 bit). Infine spiegare in dettaglio perché impronte di 64 bit non sono considerate sicure (indipendentemente dal metodo utilizzato per il calcolo).



Il DES è a chiave simmetrica e questa è già una limitazione per le applicazioni di rete, in più, considerando che quello che lavora a blocchi di 64 bit ha una chiave di 56 bit, si capisce facilmente che è soggetto ad attacchi a forza bruta. Essendo a 56 bit le possibili chiavi sono solo 2^56 = 72'057'594'037'927'936 .

Spiegato nell'esempio della risposta sul paradosso del compleanno

Descrivere l'architettura di un KDC (Key distribution center). A cosa serve? Come si differenzia da una Certification Authority ?



KDC è un centro di distrubuzione chiavi il cui compito è quello di garantire la sicurezza nello scambio della chiave di sessione R tra due utenti.

dove Ka = chiave segreta tra KDC e Alice
Kb = chiave segreta tra KDC e Bob
R = chiave di sessione

[NB] Le chiavi segrete tra utenti e KDC sono scambiate al momento della registrazione.

KDC garantisce quindi sicurezza per la crittografia simmetrica.

CA garantisce la corrispondenza tra chiave pubblica e utente quindi sicurezza per la crittografia asimmetrica. Quando A vuole conoscere la chiave pubblica di B, piuttosto che chiederla a B stesso si rivolge alla CA per ottenere il certificato di B. Il certificato viene poi sottoposto a verifica della firma digitale sulla base della chiave pubblica della CA stessa, per assicurarsi che non sia stato manipolato.

NEW - Dovete progettare un sistema sicuro per lo scambio di documenti di grandi dimensioni. Il sistema deve gestire un numero molto elevato di utenti, deve essere in grado di verificare l'identità di chi effettua il download e di garantire che il documento scaricato sia incomprensibile a terze parti. Che soluzione adottereste?

Utilizzerei :

Crittografia a chiave asimmetrica perché :

- n soggetti → 2n chiavi (<< n^2 della simmetrica)
- no accordo e scambio chiavi

Firma con impronta digitale perché :

- documenti di grandi dimensioni

Chiavi certificate con CA perché :

- abbiamo una crittografia a chiave asimmetrica

NEW - Se dobbiamo firmare documenti grandi come un impronta digitale (hash) come si può modificare il processo di firma in modo da renderlo più veloce ?

L'algoritmo di hashing ha lo scopo di calcolare un'impronta sulla quale applicare la firma nel caso il documento originale sia troppo grande.
In questo caso il doc è piccolo quindi possiamo evitare l'algoritmo di hash e applicare direttamente la firma sul doc originale.

DOMANDE D'ESAME sui motori di ricerca web

Cos'è il pagerank intuitivamente e come può essere calcolato? Quali vantaggi offre?

Attraverso il pagerank è possibile ordinare le pagine web in base alla loro importanza (quindi indipendentemente dalle query) ovvero in funzione del numero di pagine puntanti e dall'importanza di quest'ultime.
Per calcolare il pagerank si utilizza il random walk con teleporting : si parte da un nodo a caso e ad ogni passo si sceglie un link uscente in maniera equiprobabile (random walk) ; per non arrivare a nodi senza link uscenti, ad intervalli regolari viene scelto un nodo casuale interno al grafo (teleporting).
Random Walk con teleporting può essere modellizzato tramite una catena di markov ergodica (ovvero in cui presi due stati c'è sempre un cammino che li congiunge) e sotto queste condizioni esiste ed è unica la distribuzione delle frequenze che è proprio il pagerank.

Quali sono i motivi fondamentali per cui un motore di ricerca che utilizza informazioni topologiche ha una maggiore efficacia di un motore basato sulle solo informazioni testuali?

Basandosi su informazioni topologiche, oltre a inquadrare una pagina in un particolare contesto informativo (es. sport, news, computer, etc...), è possibile classificare le pagine tenendo conto della popolarità (pagerank o (un)directed popularity).

Per quale motivo il pagerank è "query indipendent"?

Perché dipende solo dalla struttura dei link: infatti è possibile calcolare il vettore di probabilità e quindi assegnare il pagerank (preprocessing) prima dell'effettiva richiesta da parte di un client (query processing).

Perché il pagerank è resistente allo spamming, inteso come la capacità di una entità di alzare indebitamente la rilevanza di una pagina, mentre il solo in-degree (numero di pagine che puntano alla pagina) non lo è?

(PLUS) Inizialmente si è pensato di attribuire a ciascuna pagina un peso che fosse funzione del numero di link entranti (in-degree). A questo punto, l'ordine con cui le pagine comparivano nei risultati di ricerca non era più strettamente dipendente dalla query ma era funzione della popolarità di ogni pagina. Questo algoritmo, seppur interessante, era ancora sensibile allo spamming: un gruppo di autori poteva decidere di linkare fra loro le proprie pagine in modo tale da accrescere la popolarità di ciascuna di esse.

Con il pagerank le pagine sono resistenti allo spamming poiché non conta solamente il numero di pagine che hanno linkato quella data pagina, ma anche la frequenza con cui queste pagine sono state visitate. Una pagina ha un pagerank alto se è stata linkata da molte pagine aventi a loro volta pagerank alto quindi non è sufficiente chiedere ad un gruppo di amici, per quanto numeroso possa essere, di linkare la mia pagina nella loro.

ESERCIZI CLIENT/SERVER IN JAVA

Illustrare schematicamente la realizzazione di un'applicazione client-server scritta in JAVA mettendo in evidenza come fluisce l'informazione tra il client e il server.

```
public class Server {
    public static void main (String []args){
        ServerSocket serversocket = new ServerSocket(4444);
        Socket clientSocket = null ;
        while (true){
            clientSocket = serverSocket.accept() ;
            PrintWriter out = new PrintWriter ( clientSocket.getOutputStream(),true);
            BufferedReader in = new BufferedReader (new InputStreamReader
                                        (clientSocket.getInputStream() ));
            while ( in.readLine() != null)
                // leggo messaggi del client da IN con "in.readLine()"
                // eventualmente rispondo al client su OUT con  "out.println( stringa )"
            out.close();
            in.close();
            clientSocket.close();
        }
    }
}
```

```
public class Client {
    public static void main (String[] args){
        Socket client = new Socket("IP del Server" , 4444);
        PrintWriter out = new PrintWriter ( client.getOutputStream(),true);
        BufferedReader in = new BufferedReader (new InputStreamReader
                                        (client.getInputStream() ));
        // prendo eventuali richieste dal terminale
        while ( richiesta dal terminale != null)
            // mando la richiesta al server su OUT con  "out.println( richiesta )"
            // leggo la risposta del server su IN con "in.readLine()"
        out.close();
        in.close();
        client.close();

    }
}
```

Descrivere un'applicazione client server multi thread in JAVA limitatamente ai soli aspetti di networking.

```
public class ServerMultiThread  {
    public static void main (String []args){
        ServerSocket serversocket = new ServerSocket(4444);
        Socket clientSocket = null ;
        while (true){
                clientSocket = serverSocket.accept() ;
                new ClientThread(clientSocket).start();
        }
    }
}
```

```
public class ClientThread extends Thread  {
    public static void main (String []args){
        Socket client = null ;
        public ClientThread (Socket in)
                client = in ;
        public void run () {
                PrintWriter out = new PrintWriter ( client.getOutputStream(),true);
                BufferedReader in = new BufferedReader (new InputStreamReader
                                                (client.getInputStream() ));
                while ( in.readLine() != null)
                        // leggo messaggi del client da IN con “in.readLine()”
                        // eventualmente rispondo al client su OUT con  “out.println( stringa )”
                out.close();
                in.close();
                client.close();

        }
    }
}
```

```
public class Client {
        Come nell’esercizio precedente
}
```

Schema in java di un'applicazione client server che utilizza UDP.

```
public class UDPServer {
   public static void main(String[] args) {
        DatagramSocket socket = new DatagramSocket(4445);

        while (true) {
            byte[] buf = new byte[256];

            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);

            /* eventualmente lavoro sul contenuto del pacchetto ricevuto */
            /* memorizzo la risposta per il client in buf (es una stringa)      */

            InetAddress address = packet.getAddress(); // prendo l'indirizzo del client dal pacchetto ricevuto
            int port = packet.getPort();               // prendo la porta del client dal pacchetto ricevuto
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
        }
        socket.close();
   }
}
```

```
public class UDPClient {
        public static void main(String[] args) {
            DatagramSocket socket = new DatagramSocket(); // socket con  porta casuale tanto server sa trovarla
            InetAddress address = InetAddress.getByName(args[0]);     //indirizzo ip passato da terminale

            byte[] buf = new byte[256];

            DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
            socket.send(packet);

            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);

            /* corpo del client/socket */

            socket.close();
        }
}
```

Schema in java di un'applicazione client server che utilizza UDPmulticast.

```
public class MulticastServer {
   public static void main(String[] args) {
        DatagramSocket socket= new DatagramSocket() ;    //casuale perché mi serve solo per spedire pacchetti
        InetAddress group = InetAddress.getByName("230.0.0.1");

        byte[] buf = new byte[256];

        DatagramPacket packet = new DatagramPacket(buf, buf.length, group, 4446);

        while (true) {
             socket.send(packet);

             /* sleep for 5 seconds  */
        }
        socket.close();
   }
}
```

```
public class MulticastClient {
    public static void main(String[] args) {
        MulticastSocket socket = new MulticastSocket(4446);        //prestabilita e conosciuta da tutti i client
        InetAddress address = InetAddress.getByName("230.0.0.1");
        socket.joinGroup(address);

        byte[] buf = new byte[256];

        DatagramPacket  packet = new DatagramPacket(buf, buf.length);
         socket.receive(packet);

        /* corpo del client/socket */

        socket.leaveGroup(address);
        socket.close();
    }
}
```