

Abstract

Questo è un documento in cui si collezionano tutti gli appunti del corso di *Reti di Calcolatori e Principi di Cybersecurity*. Il documento è suddiviso in tre macroaree principali: *Strato Applicativo*, *Strato Network/Internetwork* e *Principi di Cybersecurity*. Lo "*Strato applicativo*" è argomento del *primo* parziale, invece gli altri argomenti fanno parte del *secondo* parziale.

Si sconsiglia di usare questo materiale in alternativa alle lezioni o alle slide fornite dal docente A. Bartoli, in quanto i miei appunti potrebbero contenere refusi, errori di battitura, inoltre potrebbero essere difficili da comprendere senza aver avuto prima un'idea dei contenuti, consultando prima le slide o le lezioni del corso.

Detto ciò, non mi assumo nessuna responsabilità del vostro rendimento in questo corso se decidete di basarvi su questi appunti. Al massimo, si può assegnare al lettore il seguente esercizio:

Esercizio 1.

Individuare eventuali refusi o errori in questo testo

Dino Meng

29/11/2025



"Introduzione alle Reti di Calcolatori"

Processi Client e Server

X

*Fondamenti sulla comunicazione tra processi. Paradigma processo client e processo server.
Osservazione sulla terminologia "server".*

X

0. Voci correlate

- Concetto di Processo

1. Processi Client e Server

Richiamiamo dal corso di Sistemi Operativi che un *processo* è un programma in esecuzione. Come ben sappiamo, più processi possono essere *comunicanti tra di loro*: nel tema della sincronizzazione, si sofferma su *come* possiamo coordinare i processi che stanno eseguendo sullo stesso calcolatore.

Tuttavia, notiamo che i processi possono essere anche *distribuiti* su calcolatori diversi e comunque comunicarsi! In un certo senso, possiamo avere dei calcolatori "*localizzati potenzialmente ovunque*".

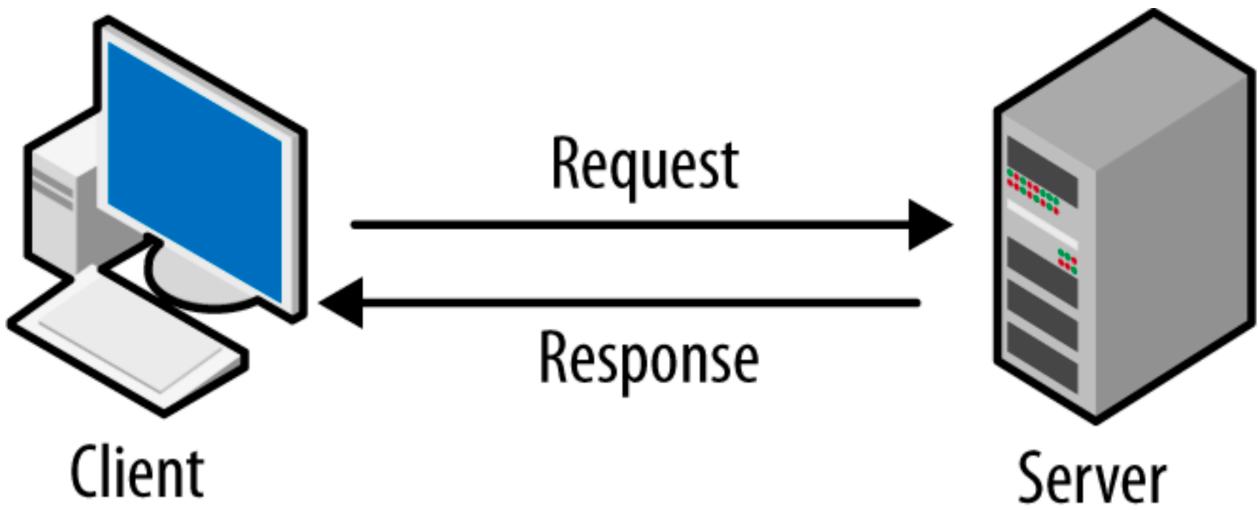
Il paradigma principale per la comunicazione tra processi è il modello *client-server*, in cui diamo le seguenti definizioni.

- *Processo Server*: Offre un "*servizio*" ad altri processi
- *Processo Client*: Usa un "*servizio*" offerto da altri processi

Vedremo cosa si intende per "*servizio*" nelle parti successive del corso.

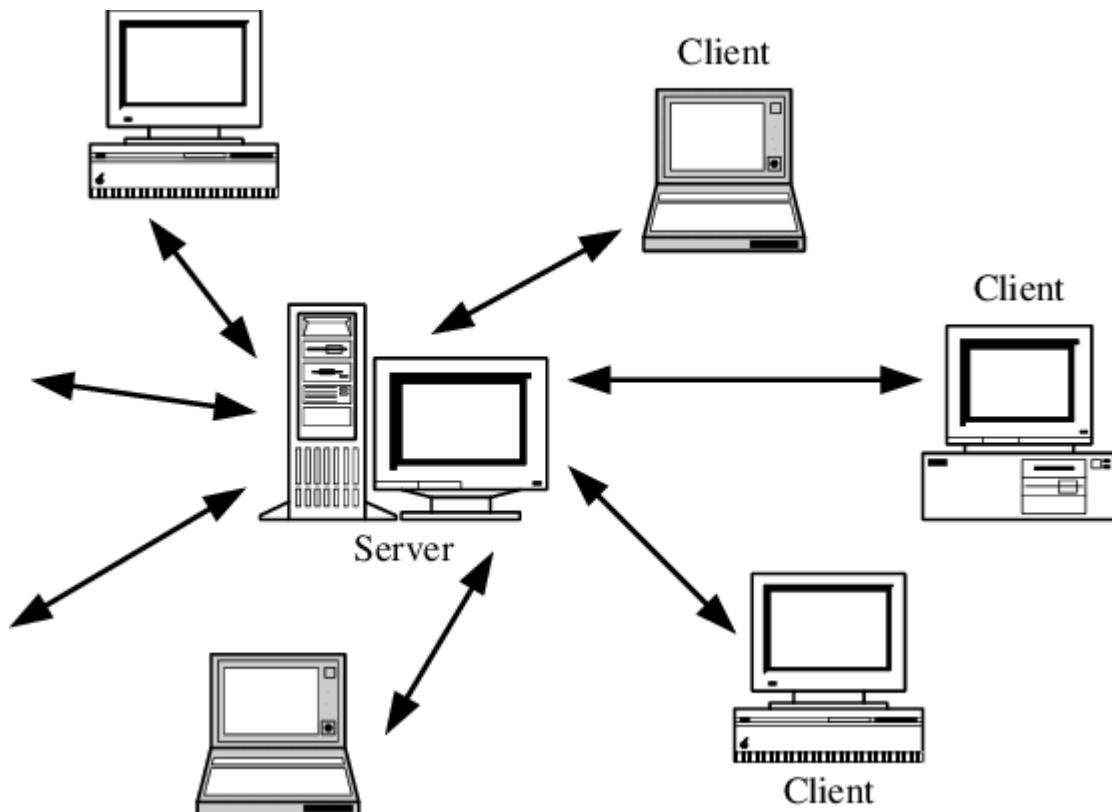
Nel modello, si esegue il seguente schema (ultra semplificato!)

1. *Client* invia una *richiesta* al *Server*
2. Il *Server* riceve la richiesta, la interpreta ed esegue il servizio
3. Il *Server* trasmette una risposta al *Client*



Stiamo effettuando una semplificazione del vero modello in quanto possiamo avere a che fare con casistiche più flessibili! In stesso intervallo temporale possiamo verificare:

- **Concorrenza:** Un *client* può usare servizi di più *server* diversi o viceversa (un *server* offre più *servizi* a *client diversi*)
- **Simultaneità:** Un processo può essere sia che *server* che *client*



Esempio. Per fare un esempio, prendiamo il *web* (semplificando la realtà):

- Un *Processo Client* sarebbe un browser in esecuzione
- Un *Processo Server* sarebbe il web server. Può prestare molti servizi, come prelevare documenti, eseguire programmi o effettuare delle query sul database, tutto fatto nel server.

2. Terminologia Server (!)

Prestiamo attenzione che *server* può riferirsi a due significati:

- Il *processo*, come faremo in questo corso
- Il *calcolatore* la cui funzione principale è *eseguire processi server sempre attivi*, come si intende di solito nel linguaggio comune

Comunicazione tra Processi

X

Comunicazione tra processi. Livello fisico: definizione di internet. Livello logico: definizione di connessione. Struttura di un client e di un server.

X

0. Voci correlate

- Concetto di Processo
- Sistema Client-Server

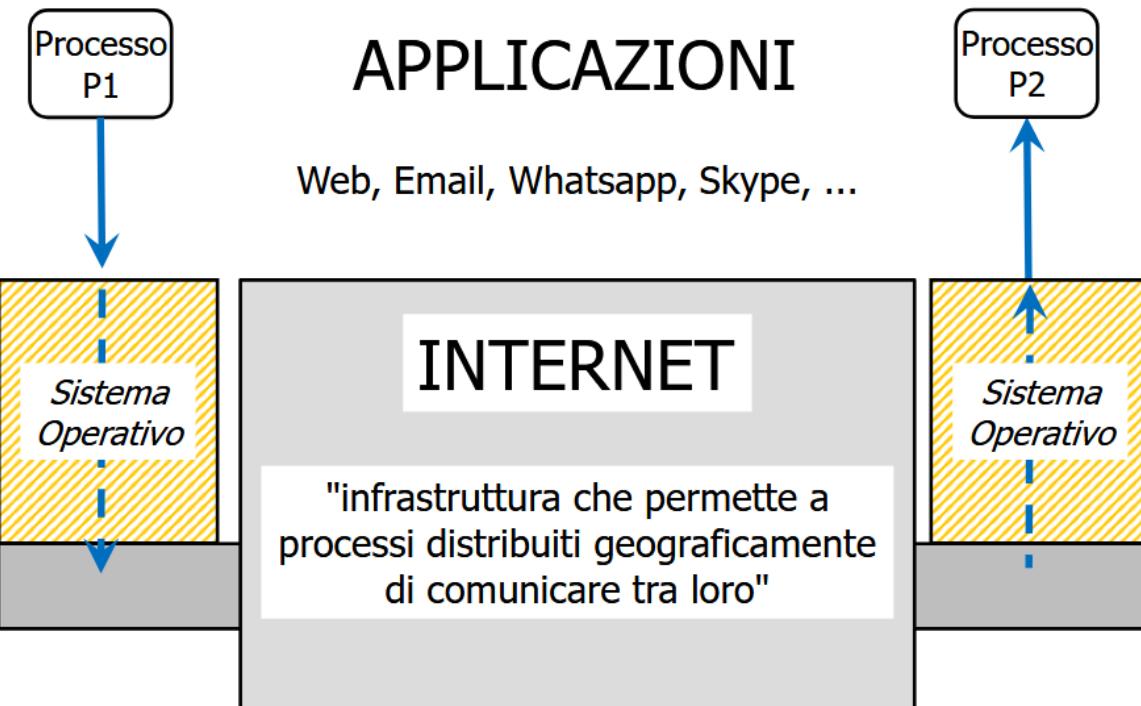
1. Comunicazione tra Processi

1.1. Livello Fisico

Supponiamo di avere due processi, P_1 e P_2 , ove P_1 desidera mandare un messaggio m al processo P_2 . Per farlo, P_1 effettuerà delle *System Call* (Libreria Standard e System Call) per istruire al sistema operativo di mandare il messaggio mediante mezzo fisico.

Dopodiché, si localizza P_2 e mediante l'*internet* il messaggio viene comunicato al calcolatore in P_2 e poi tramite delle system call riceve il messaggio.

Il passaggio cruciale (in questo caso) è proprio l'*internet*, che è una *infrastruttura che permette tale processo di comunicazione*.



1.2. Livello Logico

Ci focalizziamo sul *"come si usa"* questo sistema, senza soffermarci sui tecnicismi fisici.

Osserviamo che mediante i sistemi operativi, si va a creare una *connessione logica* in cui i processi sono in grado di comunicarsi tra di loro, inviando dei semplici comandi al proprio sistema operativo. Essa funziona come un *"tubo"*, in cui il client è in grado di inviare richieste e ricevere risposte. In un certo senso, è un'illusione prodotto dal sistema operativo.



Vediamo come i processi comunicanti useranno questo "*tubo*" (connessione logica)

Client:

1. Determina l'identificatore del server (vedremo nella parte successiva del corso)
2. Apre la connessione verso l'identificatore del server
3. Finché non si decide di chiudere la connessione,
 1. Trasmettere una *Request*
 2. Ricevere una *Response*

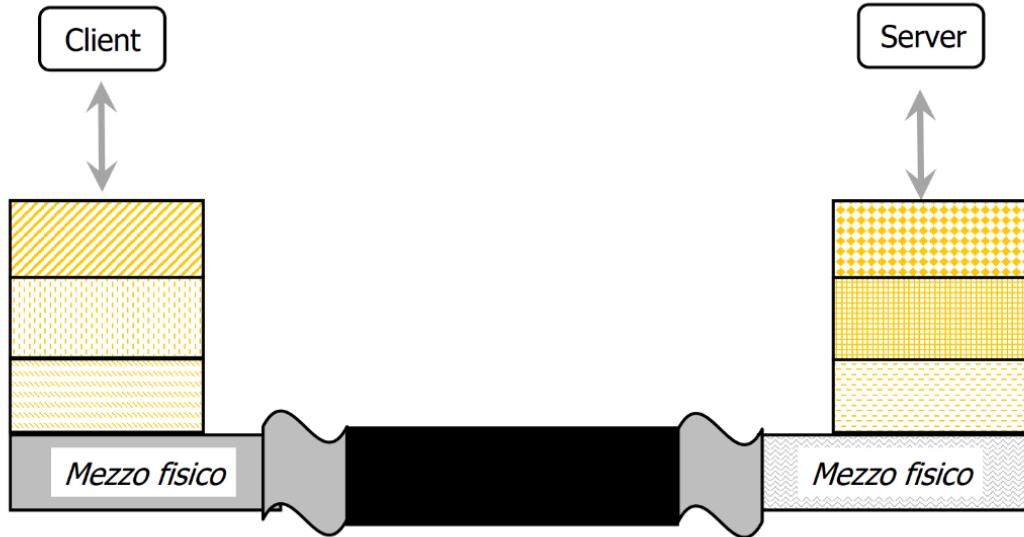
Server:

1. Sceglie il proprio identificatore
2. Attendere richieste di apertura verso il proprio id, quindi è in "*standby*"
3. Ogni volta che mantiene la connessione, finché non si chiude:
 1. Ricevere una *Request*
 2. Elaborare la *Request* internamente
 3. Trasmettere una *Response*

X

3. Rete Software

Notiamo che i *sistemi operativi* e i *mezzi fisici* associati ai processi possono essere *diversi*. In particolare, il *software di rete* è internamente suddiviso in *3 layer* (vedremo dopo perché).



Ogni layer dovrà implementare lo *stesso layer del protocollo corrispondente*, per "capirsi tra di loro" (anche se non è vero ma supponiamo vera questa cosa, con uno sforzo di fantasia)

I layer sono le seguenti:

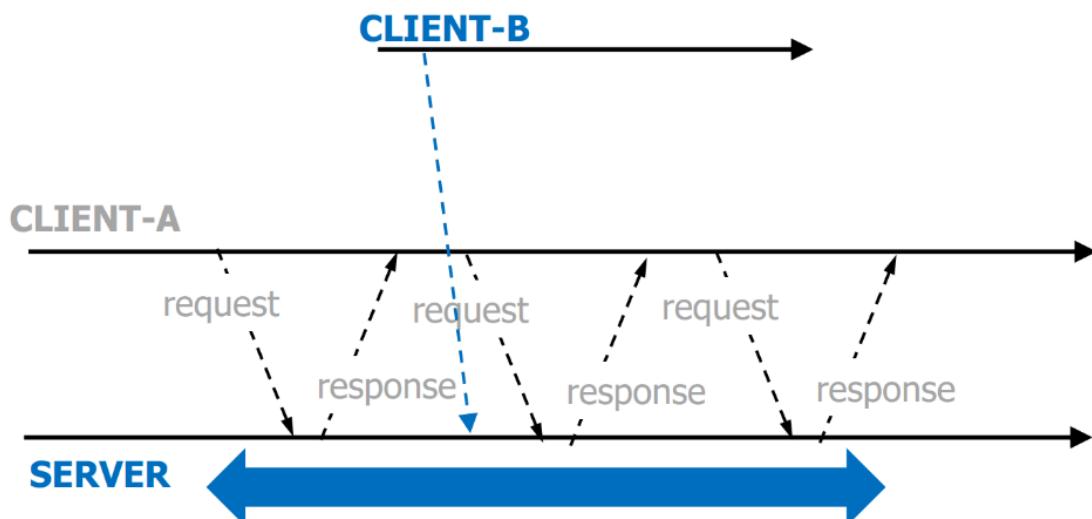
- *Protocollo IP*: Livello Intermedio
- *Protocollo TCP o UDP*: Livello più alto, e verranno "usate" dai processi.

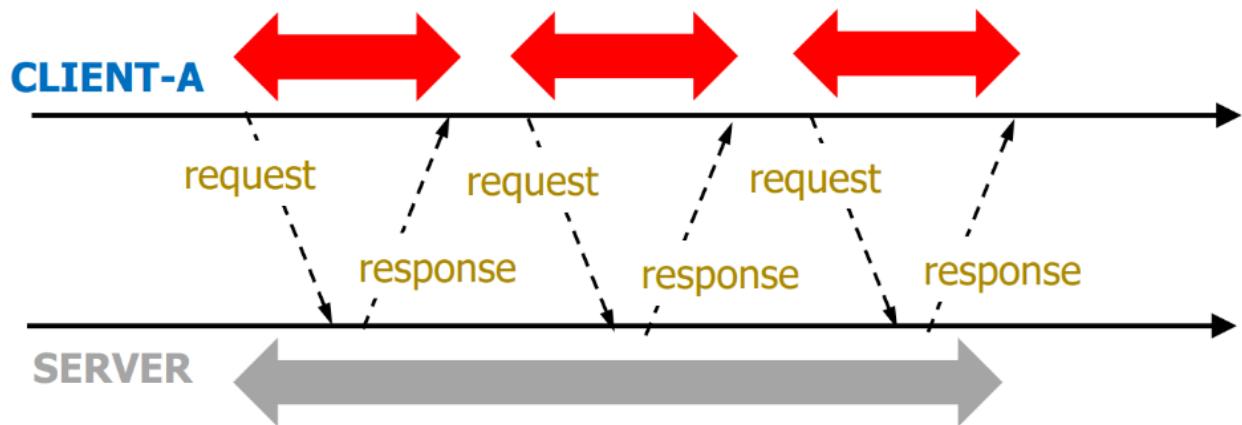
Nel corso vedremo i protocolli che usano *TCP*. L'unica l'eccezione sarà il protocollo *DNS*, che può funzionare sia su *UDP* che *TCP*.

 X

4. Concorrenza

Q. Nel modello *client-server*, è possibile che più *client* comunichino con lo stesso server (e viceversa). Si vuole gestire la *concorrenza*? Ovvero, ha senso che il *server* mantenga un client in attesa fino a quando l'altro client ha chiuso la connessione? Oppure, se è ragionevole che un client rimanga inattivo tra invio request e ricezione risposte?





Per dare una risposta, osserviamo le scale temporali. Riportiamo le seguenti *scali temporali* per un calcolatore:

- 1 Ciclo CPU: 0.3 ns
- Accesso alla DRAM dalla CPU: 120 ns
- Ping: Un numero variabile, da 40ms a 183ms

Se normalizziamo questa scala rispetto ad un ciclo CPU (quindi assumendo che duri un secondo), si avrebbe

- 1 Ciclo CPU: 1 s
- Accesso alla DRAM dalla CPU: 6 min
- Ping: Un numero variabile, da 4 anni a 19 anni (!)

Quindi confrontando le *scale* dei tempi, vediamo che è necessario implementare *server* e *client* che siano in grado di gestire comunicazioni parallele e concorrenti.

Indirizzo TCP

X

Metodo di identificazione dei processi nelle reti. Indirizzo TCP: definizione, indirizzo IP e port number. Notazione IP: dotted decimal notation. Relazione tra indirizzo IP e la relativa posizione geografica del calcolatore. Scelta dell'indirizzo TCP dei processi server. Assegnazione dell'indirizzo TCP dei processi client.

X

0. Voci correlate

- Comunicazione tra Processi

1. Indirizzo TCP (semplificato)

Q. Supponiamo di avere due processi che vogliono comunicare tra di loro. Come vengono *identificati* i processi sull'internet? In particolare, come garantiamo che la maniera con cui vengono identificati siano *univoci*?

Una soluzione è fornita dall'*indirizzo TCP*.

DEFINIZIONE. Ogni *processo* collegato a Internet è identificato univocamente dal *suo indirizzo TCP*, una coppia formata dall'*indirizzo IP* e il *port number*.

- L'indirizzo IP è un numero naturale compreso nell'intervallo $[0, 2^{32} - 1] \approx [0, 4 \cdot 10^9]$ e identifica il *nodo* (inteso come *calcolatore* sull'internet)
- Il port number è un numero naturale compreso nell'intervallo $[0, 2^{16} - 1] = [0, 65535]$ e identifica il *processo* sul nodo

L'indirizzo IP è *assegnato* al nodo, il port number invece è "*scelto*" dai processi server e *assegnato* ai processi *client*.

X

2. Proprietà dell'Indirizzo IP

2.1. Notazione Indirizzo IP (IPv4)

Un modo comune per rappresentare un indirizzo IP è la *dotted decimal notation*. Essa consiste in prendere la rappresentazione binaria dell'indirizzo IP (quindi ho 32 bit), suddividerlo in 4 byte e poi di rappresentare ogni byte in decimale; infine li mettiamo assieme, separandoli con un punto.

Esempio: **11000000|10101000|00000001|00000001** diventa **192.168.1.1**

2.2. Relazione (approssimativa) con la Posizione Fisica

Premettiamo che l'indirizzo IP non è un *localizzatore*, in quanto essa serve principalmente ad *identificare*. Tuttavia, vedremo che in certi casi è possibile "*dedurre*" delle informazioni sulla posizione geografica

- L'indirizzo IP rappresenta una posizione geografica *molto approssimata*, di solito con grande errore. Ad esempio, se l'indirizzo IP ci punta al campus principale dell'Università degli Studi di Trieste, non sarà possibile sapere in quale aula esatta si trova il nodo
- Dei "*nodi molto vicini*" possono avere indirizzi IP "*molto simili*", soprattutto nei byte a destra
- Ogni volta che un nodo si sposta a distanza sufficiente, deve cambiare l'indirizzo IP

Approfondiremo questo aspetto nella parte 2 del corso

X

3. Determinazione dell'Indirizzo TCP

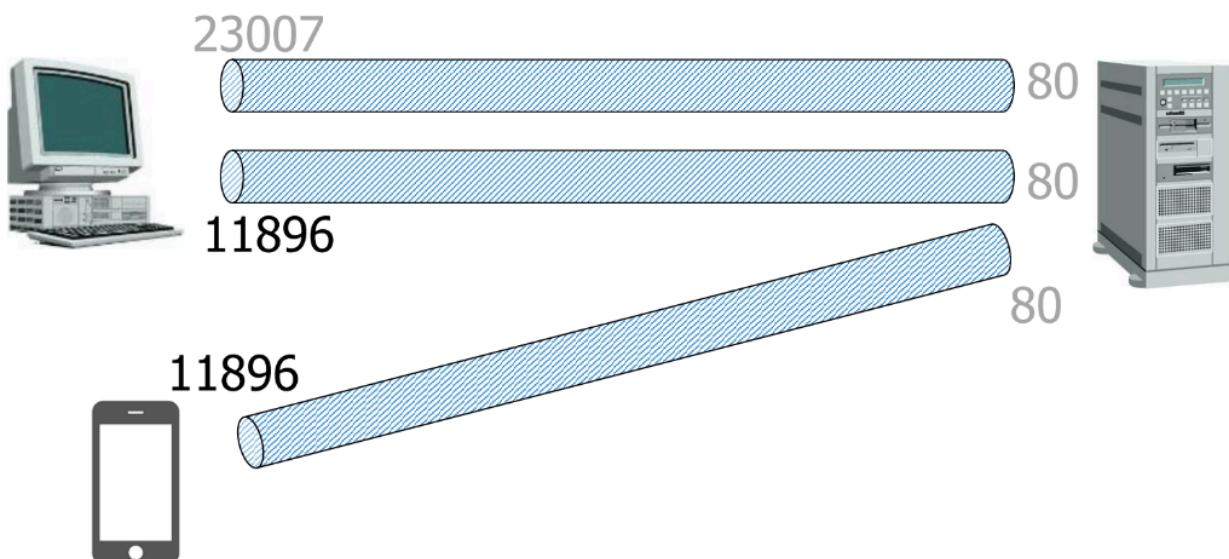
Client:

- Il port number viene scelto dal *sistema operativo* e *sul momento*. Esso è diverso da ogni altro port number già in uso, e dev'essere superiore a 1024 (per motivi storici, si vuole evitare di occupare i port che userebbero un server)
- Può (e di solito succede) cambiare ad ogni esecuzione

Server:

- Si sceglie il port number in base al *protocollo* da implementare e viene fissato

Osserviamo che, dato un server, se due processi client su nodi diversi si collegano al server allora *possono* avere lo stesso port (in quanto gli IP saranno sicuramente diversi)



Ricerca di Indirizzi IP

X

Ricerca di indirizzi IP per client. Esempi introduttivi: Protocollo HTTP, protocollo SMT/POP. Schema generale.

X

0. Voci correlate

- Indirizzo TCP
- Comunicazione tra Processi

1. Esempi Introduttivi di Ricerca Indirizzo IP

Q. Supponiamo che un client voglia collegarsi ad un server. Come primo passo, deve identificare l'indirizzo IP del server. Come fa?

Adesso vediamo con un paio di esempi.

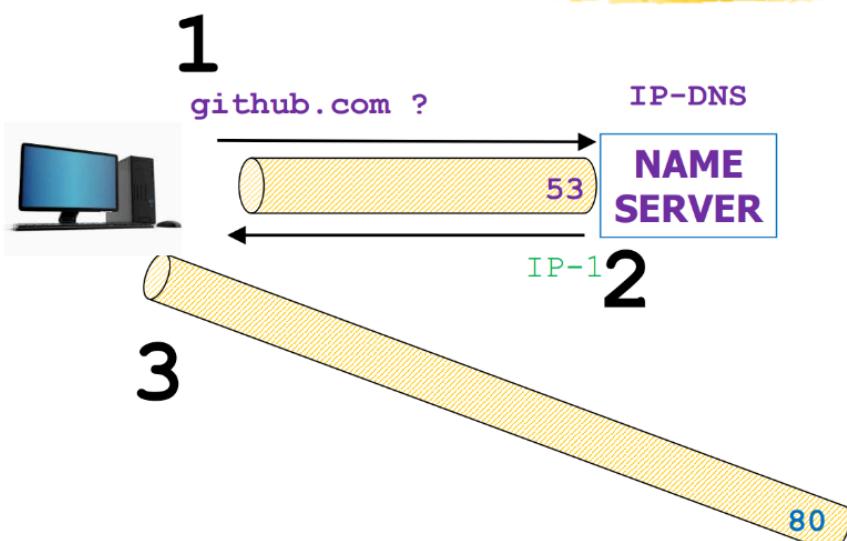
1.1. HTTP

Step 1: Conoscendo l'**URL** della *pagina web*, traiamo il *nome server*

- *Esempio:* **https://www.units.it/catalogo-della-didattica-a-distanza** è l'URL, otteniamo il nome server **www.units.it**

Step 2: Conoscendo il *nome server*, ottenerne l'**IP-server**. Per farlo, bisogna interagire col *protocollo DNS*. Essa consiste in inviare una *richiesta* del tipo "**indirizzo IP di <nome_server>?**" e ricevere la risposta contenente l'**IP-server**, da cui concludiamo.

- Daremo per scontato l'**IP-DNS** e la sua *port number* conosciuti. In particolare il port sarà 53.



1.2. Email

Step 1: Conoscendo il *proprio indirizzo email* (conosciuto a priori, diciamo che è fa parte della *configurazione*), ottenere il nome server

- *Esempio:* **bartoli.alberto@units.it** diventa **mx.units.it**
- Vedremo come funziona nei dettagli dopo

Step 2: Come prima, interagire col server DNS per ottenere l'*IP-Server*

X

2. Schema Generale

Step 1. Dipendente dal protocollo e dalla configurazione

Step 2. Dal *nome server* otteniamo l'*IP-Server*, basandoci sulla DNS (quindi ci servirà l'*IP-DNS* in configurazione)

Osserviamo che la *step 2* è la parte cruciale dello schema di ricerca di indirizzi IP.

Socket

X

Socket per implementare IPC ad alto livello. Definizione di socket, interfaccia TCP semplificata (primitive). Struttura di un processo server (idea e pseudocodice). Struttura di un processo client (idea e pseudocodice).

X

0. Voci correlate

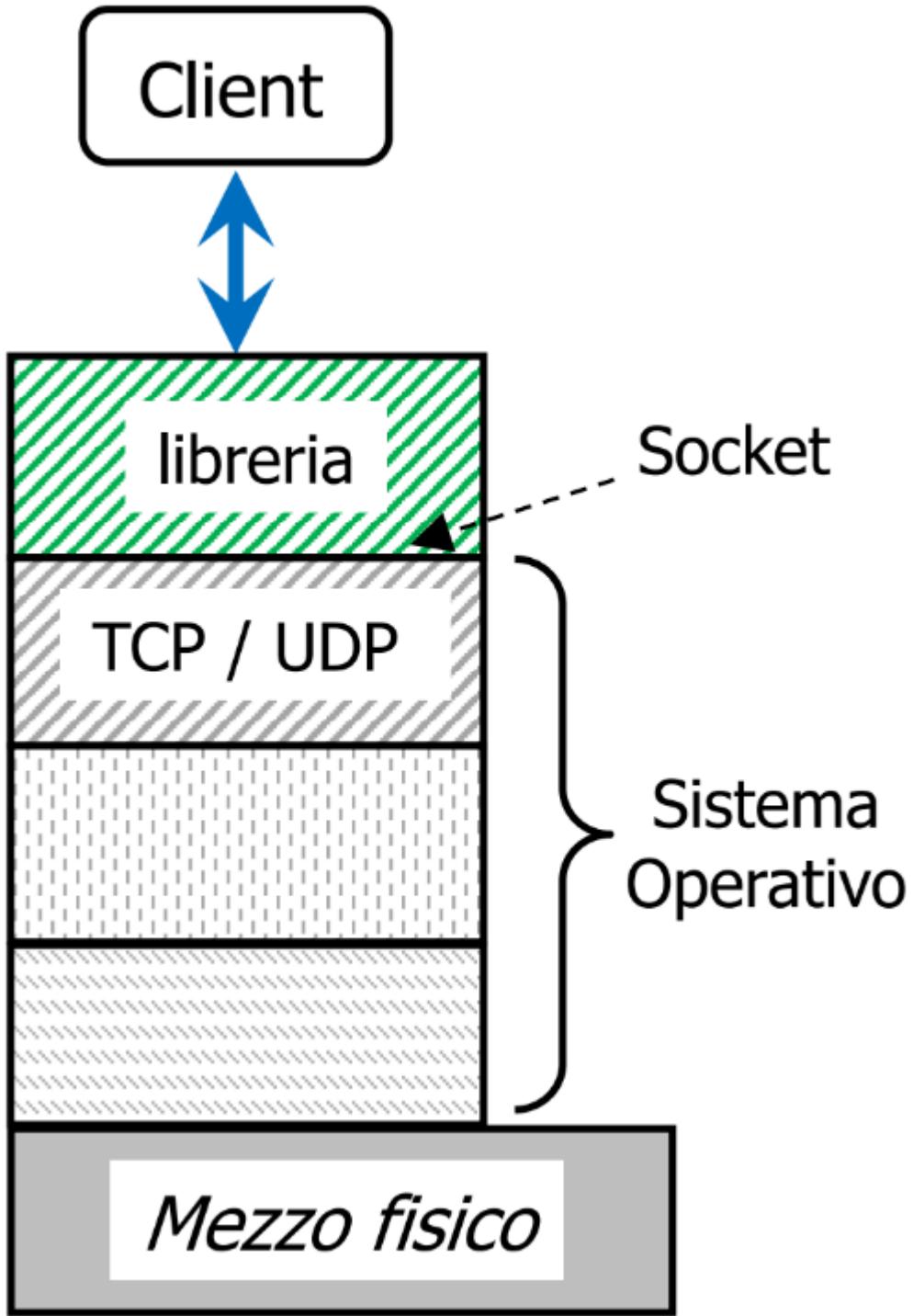
- Comunicazione tra Processi
- Socket

1. Socket

Dato un *collegamento* tra due *processi*, abbiamo che i sistemi operativi realizzano questo "*tubo*" che li permette di inserire e trasmettere messaggi. Vediamo nei dettagli come funzionano questi meccanismi, in particolare le *System Call*.

Chiamiamo i *socket* le *interfacce di programmazione* (insieme di funzioni/primitive) attraverso le quali si accedono *TCP* o *UDP*. Sono considerate *quasi standard* e molto complicate.

Molto spesso delle *librerie* (e.g. Python) implementano l'interfaccia socket con un livello di astrazione ancora più alto, tuttavia non sono più considerati standard (e dipende dal linguaggio).



2. Interfaccia TCP

Vedremo come, in una maniera semplificata, i socket implementino l'interfaccia **TCP**. Si hanno le seguenti system call (primitive):

- **SOCKET**: Va a creare l'estremità del collegamento
- **SEND**: Mandare dati nella connessione
- **RECEIVE**: Ricevere dati dalla connessione
- **CLOSE**: Chiudere la connessione

Queste primitive vengono utilizzate sia da processi client che server. Vediamo adesso le primitive specifiche necessarie per implementare la struttura di un client e di un server.

2.1. Server

Ricordiamo che il server:

Server:

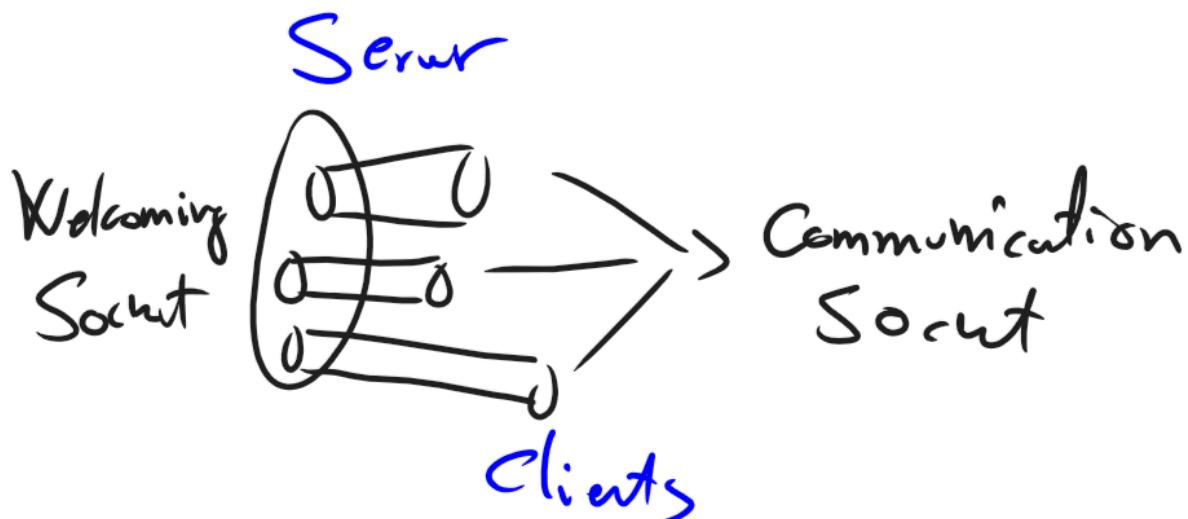
1. Sceglie il proprio identificatore
2. Attendere richieste di apertura verso il proprio id, quindi è in "*standby*"
3. Ogni volta che mantiene la connessione, finché non si chiude:
 1. Ricevere una *Request*
 2. Elaborare la *Request* internamente
 3. Trasmettere una *Response*

Adesso vediamo come implementiamo gli step 2 e 3.

PSEUDOCODICE.

- Creare socket s_1 (*SOCKET*)
- Scegliere un port number (da fare seguendo i protocolli)
- Associare s_1 al port number scelto (*BIND*)
- Dichiara disponibilità a connessioni su s_1 (*LISTEN*)
- Attendere connessione su s_1 . Non appena si ottiene una connessione, accettare (*ACCEPT*):
 - Ottenere un altro socket s_2 e comunicare su s_2 col client

Notiamo che in effetti ci sono più server socket: abbiamo un *welcoming socket* che attende le connessioni, e più *communication socket* per ricevere dati



Osservazione:

- Il server non ha bisogno di conoscere l'indirizzo TCP dei client, anche se può conoscerli con determinate funzioni in casi particolari

- Il server usa il proprio port number *soltamente una volta*, ovvero nel *bind* col welcoming socket
- Il server non conosce i port number dei *communication socket*, e non ha necessità di conoscerli. Infatti, hanno lo stesso numero di porta; vedremo che il sistema operativo è in grado di distinguerli, ma non in questo corso

2.2. Client

Ricordiamo la struttura di un client:

Client:

1. Determina l'identificatore del server (vedremo nella parte successiva del corso)
2. Apre la connessione verso l'identificatore del server
3. Finchè non si decide di chiudere la connessione,
 1. Trasmettere una *Request*
 2. Ricevere una *Response*

PSEUDOCODICE.

1. Crea socket *s* (*SOCKET*)
2. Connnette *s* con (IP-SRV, PORT-SRV) (*CONNECT*)
3. Comunica su *s* in base al protocollo usato

Osservazioni:

- Il client non si accorge che sul lato server ci sono *due socket*
- Il client non conosce il proprio port number, e non avrà necessità di conoscerlo in nessun caso (può farlo lo stesso)

Protocolli di Testo e Binari

X

Breve descrizione qui

X

0. Voci correlate

- Comunicazione tra Processi

1. Definizione di Protocollo

Abbiamo visto che abbiamo una *collegamento* tra due processi è un "*tubo*" in cui possiamo trasmettere dei dati. Ma come sono rappresentati questi dati?

Vengono trasmessi sotto forma di *byte*, ovvero un numero naturale in $[0, 255] \cap \mathbb{N}$. Questo ha conseguenze importanti! Infatti, enunciamo il *principio di openness*

Openness: I client e server possono essere sviluppati da organizzazioni diverse e in luoghi e tempi diversi.

Dal principio sorge il problema di *interpretare* effettivamente i dati, ovvero come i processi possano "*capire*" tra di loro.

Protocolli: La soluzione immediata e la più semplice è quello di usare i *protocolli*, quindi un insieme delle *regole note a priori*. In particolare possiamo avere le seguenti tipologie di regola:

- Sintassi (*struttura*)
- Semantica (*significato*)
- Tempistiche

I processi che partecipano in una applicazione devono implementare il *protocollo di quell'applicazione*.

Di solito vengono gestiti apertamente dalla *IETF* (Internet Engineering Task Force) e sono descritti nei documenti chiamati *Request for Commenters* (RFC). Ciò rende libero lo sviluppo di processi client e/o server.

Tuttavia, ci sono casistiche rari in cui i protocolli vengono sviluppati privatamente e sono mantenuti riservati e/o coperti da copyright (ex: Skype).

X

2. Categorie di Protocolli

Partiamo dal presupposto che *ogni byte* trasmesso viene rappresentato in sistema *esadecimale*.

Ci sono principalmente *due categorie di protocolli*: *testo* e *binari*.

Testo:

- *Ogni byte viene rappresentato sotto forma di un carattere.* Solitamente si usa la convenzione ASCII. Per codificare la *linea di testo*, si usa il *line feed* e *carriage return* (codificate da **0A** e **0D**)
- Esempi: HTTP, SMT POP

Binario:

- Ogni *bit* ha un significato specifico e non può essere tradotto in testo. Alcuni byte hanno valori che corrispondono a caratteri ASCII, ma il messaggio rimane comunque non interpretabile. I *software* come *Wireshark* permettono di dare una interpretazione.
- Esempio: DNS

Proprietà dei Servizi di Comunicazione

X

Proprietà dei servizi di comunicazione. Definizione di servizio connection-oriented o connectionless; byte-oriented o message-oriented; reliable o unreliable. Combinazioni tipiche: TCP, UDP.

X

0. Voci correlate

- Comunicazione tra Processi

1. Proprietà Servizi di Comunicazione

Dato un *servizio di comunicazione*, lo immaginiamo come un intermediario che manda messaggi. Nell'esempio fisico, si avrebbero le poste. Quindi non tutto è garantito, tra cui che *arrivino* effettivamente i messaggi, oppure che *arrivino in ordine*. Quali garanzie possiamo offrire a chi *usa* il servizio di comunicazione?

1.1. Orientamento con la Connessione (collegamento)

DEFINIZIONE. Un servizio di comunicazione si dice *communication oriented* (o *connection*) se essa segue questo schema:

- Si apre la connessione
- Si *trasmette o riceve* nella connessione
- Si chiude la connessione
(in parole semplici, va ad usare un "*tubo*" da aprire prima delle trasmissioni)

Quindi è necessaria un'apertura *esplicita* prima dell'utilizzo, in cui specifico il destinatario. Notiamo che implicitamente essa garantisce l'*ordinamento* dei messaggi, i.e. l'*ordine di ricezione* è l'*ordine di trasmissione*.

DEFINIZIONE. Un servizio di comunicazione è invece *connectionless* se invece non richiede un'apertura preventiva di nessuna connessione. Ogni *trasmissione* indica il destinatario e trasmittente; in questo caso, non manteniamo l'*ordinamento* in quanto ogni trasmissione è indipendente dall'altra.

1.2. Orientamento su Messaggi o Byte

DEFINIZIONE. Un servizio di comunicazione è:

- *Byte-oriented* se delle trasmissioni consecutive creano un *unico flusso di byte*, quindi il destinatario può ricevere più trasmissioni allo stesso tempo

- *Message-oriented* se ogni trasmissione è un messaggio separato dagli altri

Osserviamo quindi che in un servizio *byte oriented*, si ha che i byte mandati e ricevuti *sono diversi!*

Inoltre, in un protocollo *byte-oriented* potrebbe essere anche necessario avere delle *regole* per partizionare il flusso di byte in messaggi.

1.3. Reliability

DEFINIZIONE. Un servizio di comunicazione si dice *reliable* se tutti i dati sono ricevuti e ogni dato viene ricevuto *solo una volta*. Invece è *unreliable* se non è *reliable*, quindi o i dati possono essere persi o i dati possono essere ricevuti più volte.

N.B. In realtà la nozione di *unreliability* è *molto più complessa*, e va a tenere altri fattori. In questo corso ci faremo andare bene con questa definizione.

2. Combinazioni tipiche delle Proprietà

TCP:

- Communication-oriented
- Reliable
- *Byte-oriented* (!)

UDP:

- Connectionless
- Unreliable
- Message-oriented

Naturalmente, *TCP* è sempre l'alternativa migliore (anche se leggermente più costosa, da un punto di vista computazionale). L'*UDP* è esistito per motivi storici.

OSSERVAZIONE. Nessuna delle proprietà danno una garanzia di ordine temporale: per i servizi di comunicazione, il tempo "*non esiste*".

"DNS"

Prime Nozioni sull'Internet

X

Prime nozioni sull'internet, per il DNS. L'internet a grandi linee: dispositivi che si collegano ai router. Definizione di organizzazione, router di frontiera. Esempi di organizzazioni. Definizione di Host.

X

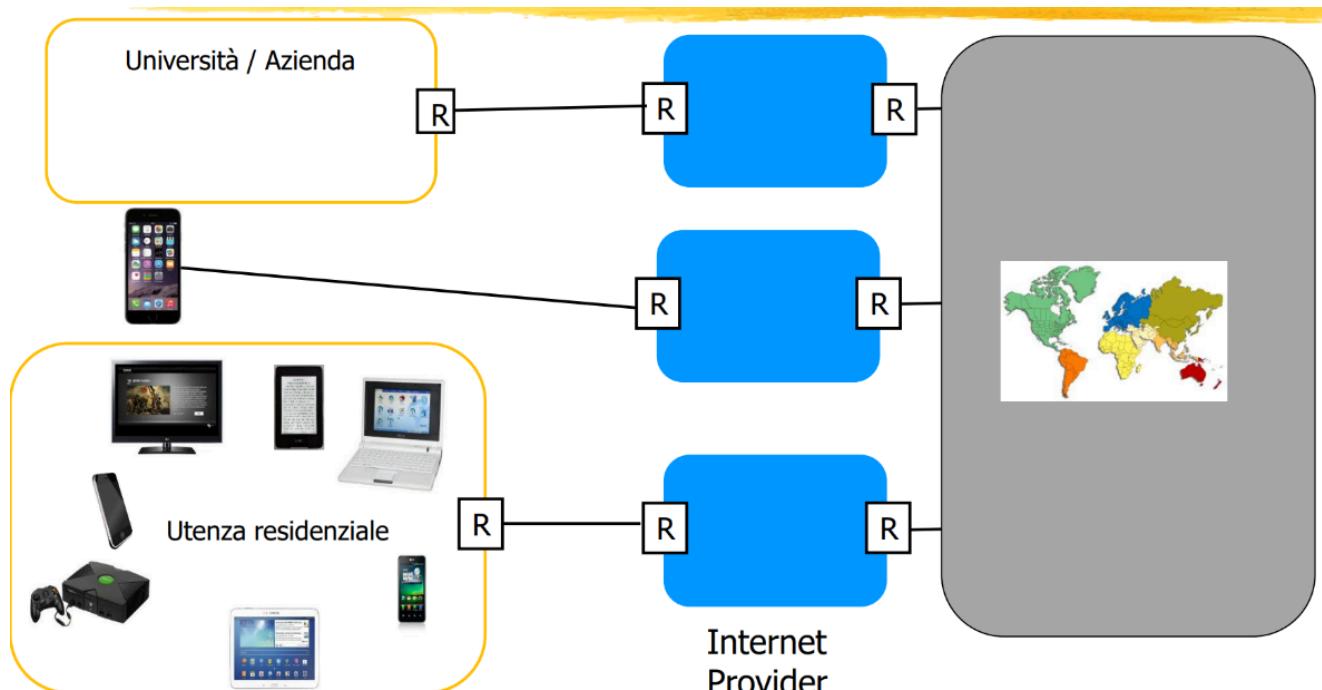
0. Voci correlate

- Comunicazione tra Processi

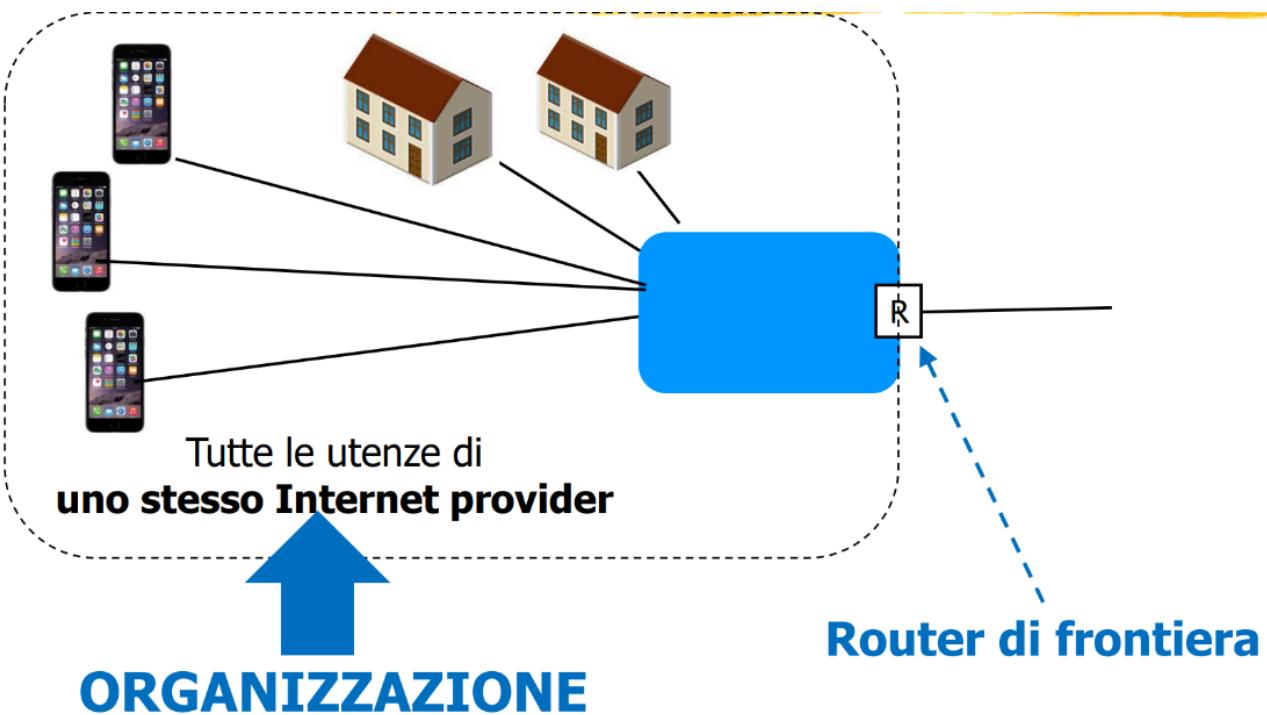
1. Prime Nozioni sull'Internet

A grosso modo, era stato definito l'*internet* come un'infrastruttura fisica che permette ai calcolatori di comunicare tra di loro. Adesso, diamo un paio di cenni di com'è strutturato l'internet.

A ogni dispositivo si collega ad un *router di frontiera*, che è un calcolatore da cui transita tutto il traffico all'internet. I router di frontiera sono gestiti dagli *internet provider*, organizzazione di mestiere.



Diciamo un'*organizzazione* tutte le utenze di uno stesso *Internet provider*, quindi che sono collegati allo stesso router di frontiera.



Esempi:

- PC collegato ad Internet da casa: appartiene al provider a cui si è iscritto, come Fastweb
- Lo stesso PC collegato ad Internet con eduroam: adesso appartiene all'Università a cui è collegata

Q. Come mai ogni dispositivo deve collegarsi ad un'organizzazione?

Nella pratica, i *router di frontiera* possono essere configurati per bloccare certi tipi di traffico (analisi sui numeri di porta) e anche alcuni indirizzi IP (vedremo con DNS).

DEFINIZIONE. Diremo *host* un calcolatore collegato a Internet. Il termine "*nodo*" può creare ambiguità, in particolare nel conteso delle *DNS*. Ogni host ha un *indirizzo IP* e può avere un nome.

X

DNS: Motivazioni, cos'è. Come riesce ad effettuare il Name Resolution. Osservazioni sul DNS.

X

0. Voci correlate

- Prime Nozioni sull'Internet
- Ricerca di Indirizzi IP
- Comunicazione tra Processi

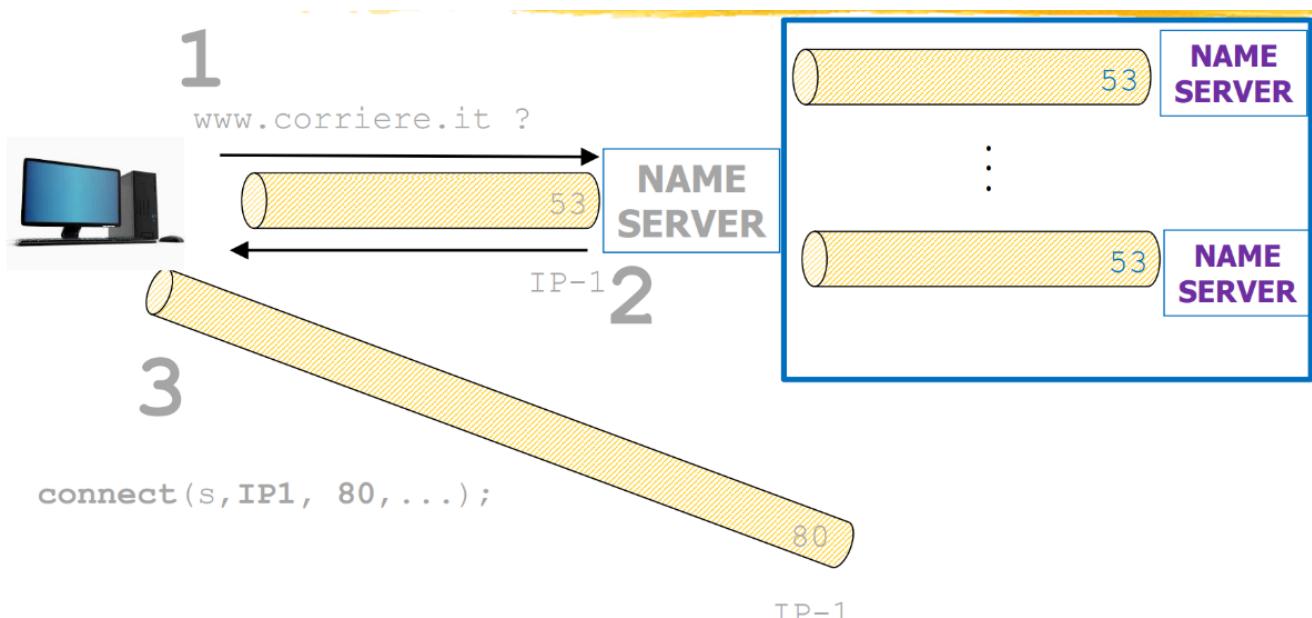
1. Nomi degli Host

Ogni *host* ha un *indirizzo ip* e può avere un *nome*. Il nome è utile per *aiutare* a individuare gli host, in quanto individuarli tramite indirizzo IP (anche indispensabili) è *scomodo*.

DOMAIN NAME SERVER. Comunque, per collegarsi ad un altro calcolatore, è comunque necessario risalire al suo indirizzo IP. I *Domain Name Server* (DNS) servono principalmente per realizzare una *tavella di traduzione* tra *name* e i loro corrispondenti *value* (indirizzo IP).

- I *DNS* sono centinaia di migliaia di name server distribuiti *geograficamente*
- Ognuno è "circa" gestito da un'organizzazione diversa
- Tuttavia ogni client avrà l'illusione di vedere un solo *DNS* (server DNS)

DEFAULT NAME SERVER. Ricordiamo il processo per *trovare* l'indirizzo IP di un host, bisogna avere l'indirizzo IP di un "*server DNS*" (Ricerca di Indirizzi IP). Questo *server DNS* andrà in realtà (tipicamente) a contattare *altri name server*.



Non vedremo nei dettagli *come* si effettuano le altre chiamate su DNS. Facciamo tuttavia un paio di osservazioni più utili a fine pratici:

- L'indirizzo IP del *Default Name Server* dev'essere noto a priori!
- Ogni *Name Server* opera sia da *server* che *client*, fornendo un esempio di simultaneità
- Ogni *host* contatta *sempre e solo* il proprio name server. La ricerca effettiva viene fatta *solo* dal *Default Name Server*, non gli altri *DNS* (intesi come *Domain NS*).
- Nel DNS i nomi terminano sempre col carattere `.`, da tenere conto per gli esercizi

Notiamo infine che *DNS* è un termine ambiguo, in quanto può riferirsi:

- Alla *intera infrastruttura mondiale* che realizza la tabella di ricerca *name-value*
- Ad un *name server specifico*, i.e. il *Default Name Server* o un'altro
- Al *protocollo di comunicazione* usato per interagire coi name server

Architettura DNS

X

Cenni di descrizione fisica dei DNS. Modi di configurare i DNS degli host, a seconda delle organizzazioni. DNS Centralizzati, DNS non centralizzati.

X

0. Voci correlate

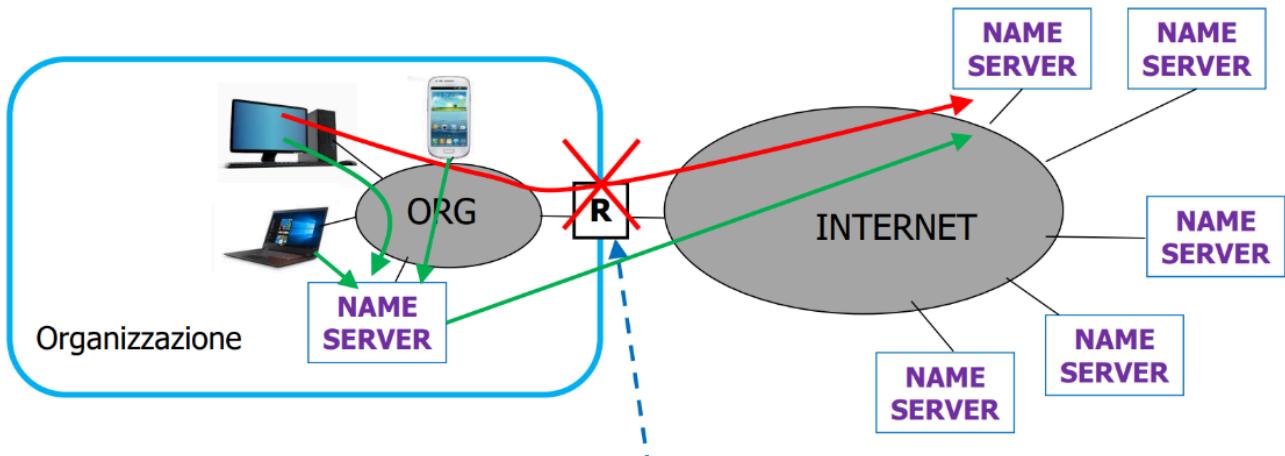
- Prime Nozioni sull'Internet
- Name Server

1. Architettura Name Server

Sappiamo che ogni host è *configurato* con *l'indirizzo IP* del proprio *DNS*. Ci sono due casi in cui configuriamo il *DNS*:

Caso 1 (tipico "aziende/università", centralizzati)

- Il DNS è "*interno ad organizzazione*" e *tutti gli host* appartenenti dell'organizzazione hanno quel *DNS*
- Si permette *solo* il traffico DNS a quel name server

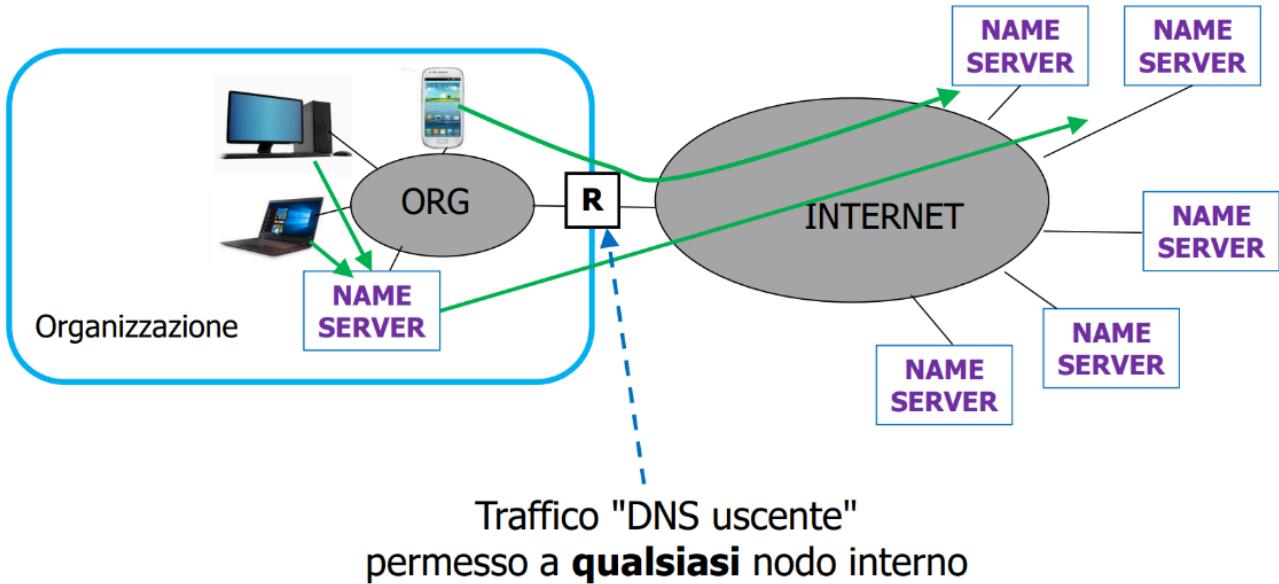


I motivi per cui si sceglie il *Caso 1* sono molteplici, tra cui:

- Motivi di prestazioni: *DNS Caching*
- Motivi di cybersecurity: Blocciamo risoluzione dei nomi "*associati ad attacchi*", ovvero che sono noti a distribuire malware o a comunicare con nodi che hanno malware
- Motivi strategici: vedere gli aspetti strategici dell'infrastruttura DNS! (Aspetti Strategici del DNS)

Caso 2 (tipico Internet Provider)

- Come *Caso 1*, solo che si permette di configurare il *proprio host* per usare un *name server esterno* (come il DNS della Google). Quindi, il router di frontiera non blocca niente
- Permettiamo traffico "*DNS uscente*" a qualsiasi host dell'organizzazione



Osservazione. Come mai alcuni offrono il proprio name server come DNS, utilizzabile da chiunque? Facendo ciò consumano un numero elevato di risorse...

- Il motivo più realistico è quello di *guadagnare più denaro* raccogliendo i dati sugli interessi degli utenti, che permette alle organizzazioni di eseguire operazioni profittabili (i.e. pubblicità mirata).

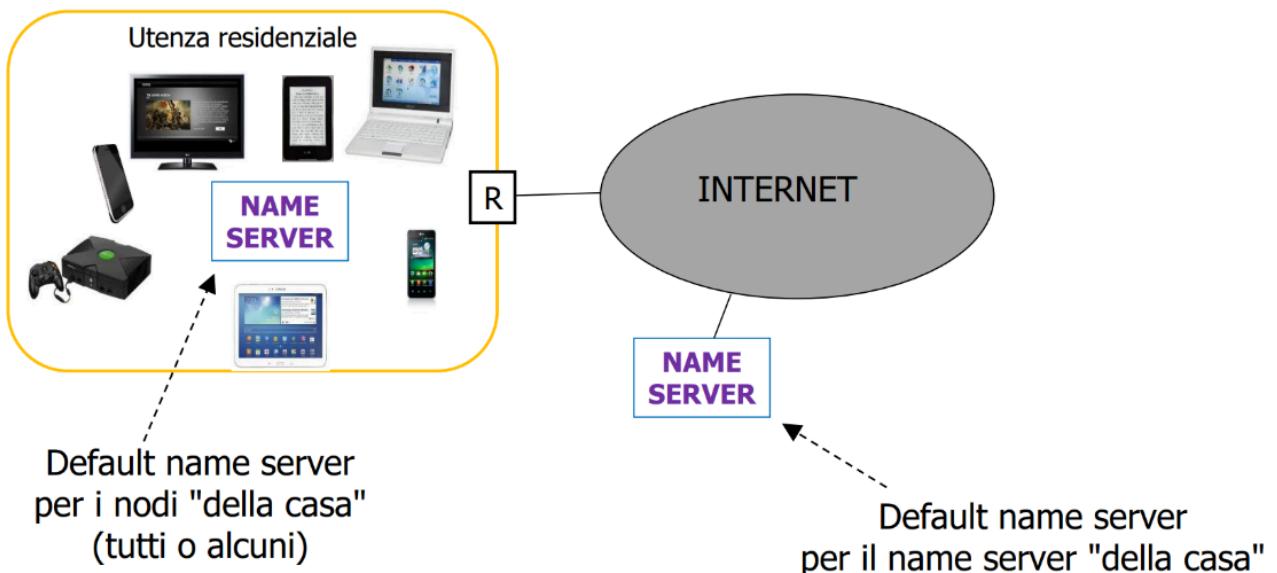
 X

2. Altri scenari DNS

Caso 3 (DNS "della casa")

In certi casi è possibile installare un *NS* sul dispositivo personale e configugarlo.

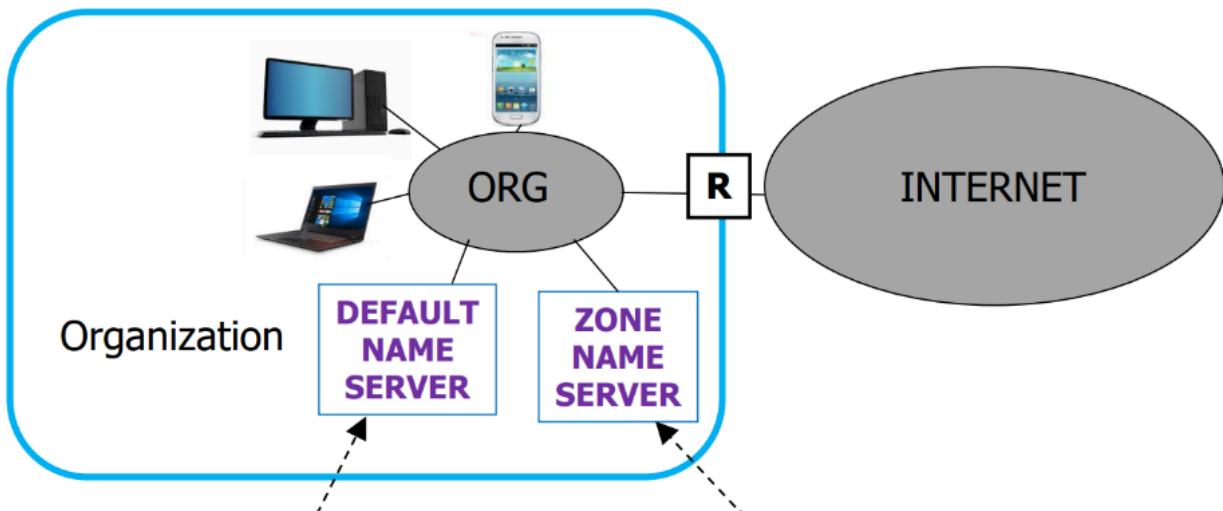
- *Esempio:* Su linux si può usare **dnsmasq**



Caso 4 (Recursive e Authoritative DNS)

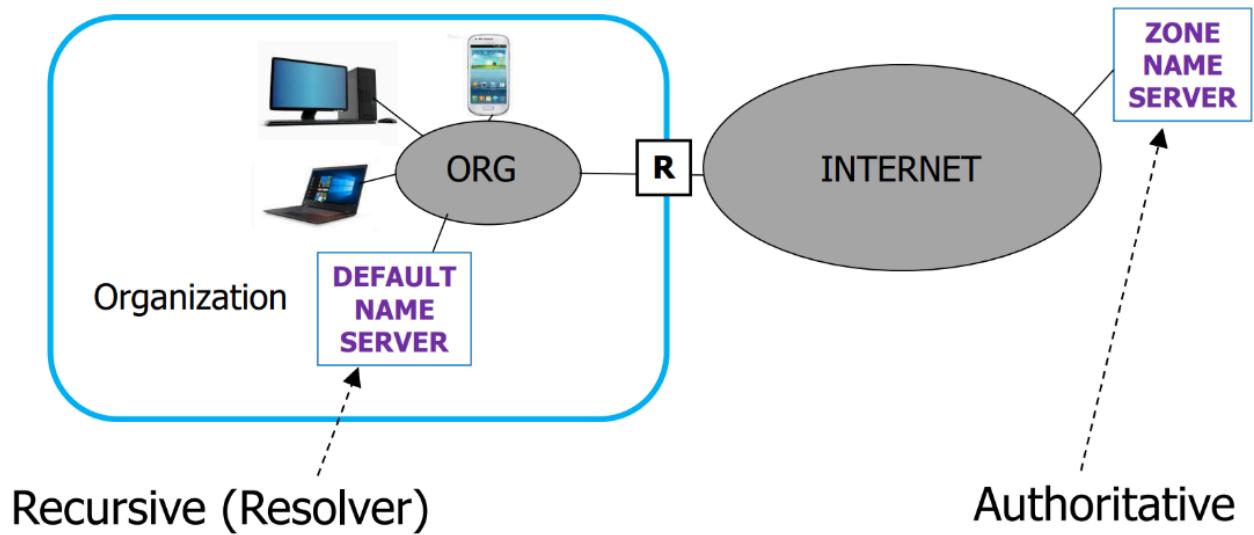
Avvolte all'interno dell'organizzazione possiamo avere *due DNS* distinti, una per *tutti i nodi interni* e l'altra contiene gli *RR* dell'organizzazione (vedremo dopo cos'è un RR, intuitivamente è una riga della mappatura name \mapsto value).

- Il primo si chiama *Recursive Name Server* e ottiene i RR richiesti dai nodi interni
- Il secondo si chiama *Authoritative Name Server* e contiene i RR della zona



Caso 4.1.

In certi casi è possibile che l'Authoritative NS si trovi all'esterno dell'organizzazione (caso comune: web hosting)



Name Resolution

X

Processo di Name Resolution. Local Resolver: precisazioni sul processo della ricerca di indirizzi IP, definizione di local resolver. Schema logico dei local resolver. Presenza RR in Local Resolver: Caching e hosts file.

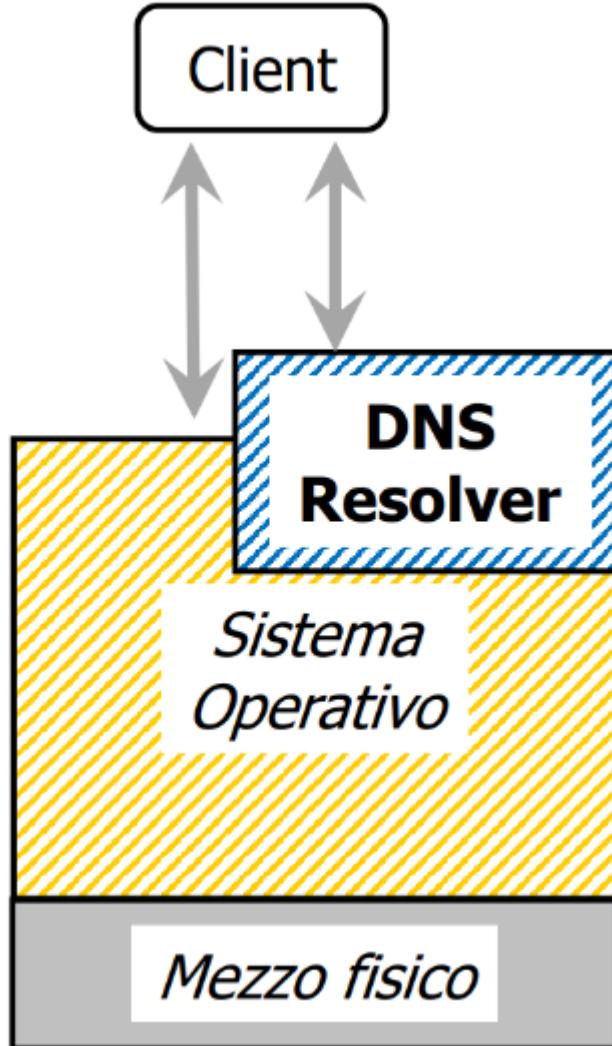
X

0. Voci correlate

- Ricerca di Indirizzi IP
- Name Server

1. Local Resolver e Name Resolution

Prima supponevamo che per trovare l'*indirizzo IP* associato ad un *calcolatore*, si andava a contattare direttamente il proprio *default name server*. Tuttavia, nella realtà abbiamo più componenti nel sistema operativo: si ha il *modulo local resolver* contenente delle funzioni che servono per effettuare il *name resolution*.

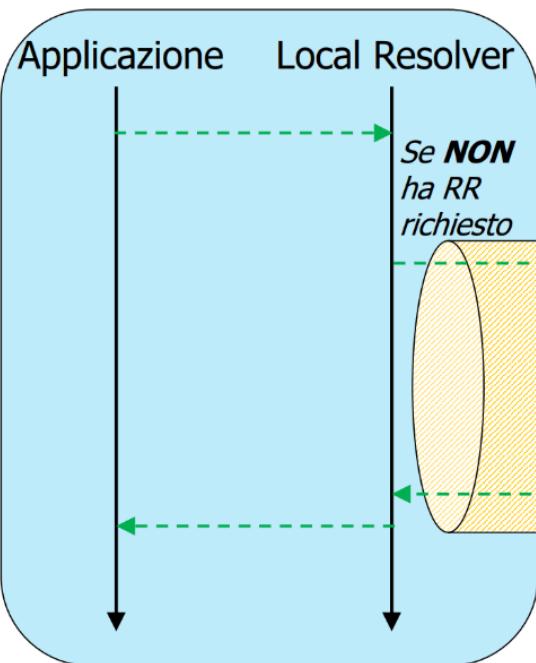
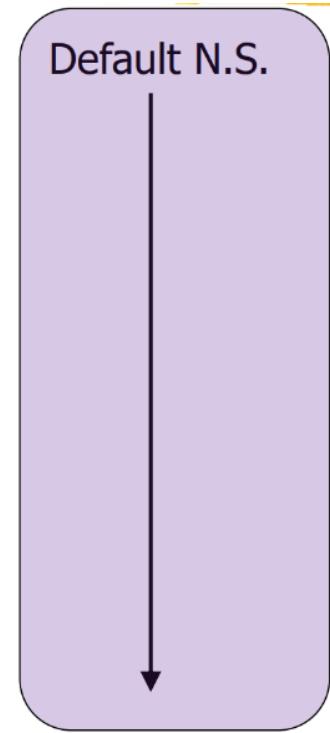
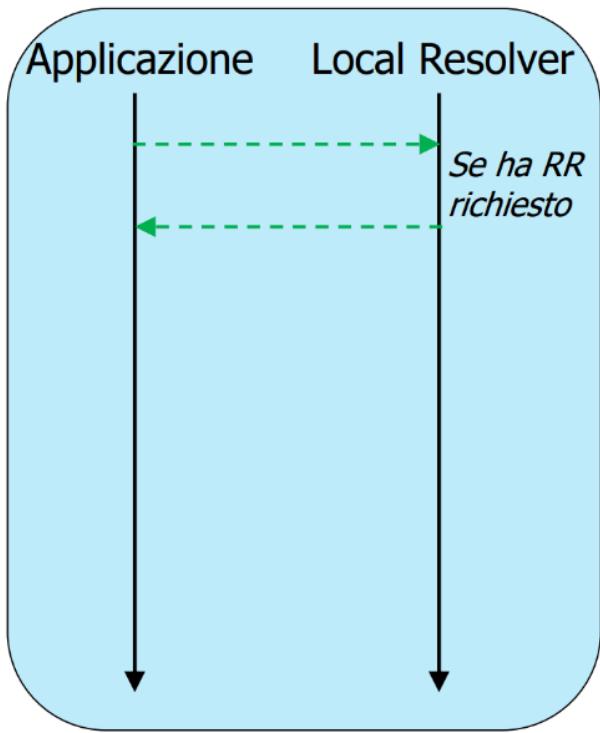


Il *name resolution* segue questo schema logico:

1. Se si conosce il *RR*, ritornare tale valore
2. Altrimenti contattare il proprio *DNS* per cercare l'*RR*

Il *local resolver* può già conoscere l'*RR* nelle seguenti casistiche:

- Ogni resolver *memorizza nella cache* per un po' di tempo gli RR ricevuti (DNS Caching)
- Si trova già la risposta nel *Hosts file*, ossia un file nel sistema operativo che traduce alcuni nomi in indirizzi IP



Cenni all'Implementazione DNS

X

Cenni all'implementazione storica del DNS. Domande: a quali DNS chiedere, come conoscere i loro indirizzi IP, come sapere che il nome non esiste, come posso creare/modificare indirizzi IP. Aspetto storico: 1985, Jon Postel. Requisiti per il sistema DNS.

X

0. Voci correlate

- Name Server

1. Requisiti dei Domain Name Server

Q. Quali proprietà deve avere un domain name server? Come faccio sapere quali DNS contattare, come conosco i loro indirizzi IP? Come raggiungo la certezza che un nome non esista? Se volessi modificare le tabelle sui DNS? Quali DNS devo modificare, e posso?

Per poter rispondere alle domande poste in una maniera soddisfacente, i DNS dovranno principalmente rispettare i seguenti requisiti:

Scala: Il sistema dev'essere in grado di "*scalare bene*", in quanto si ha a che fare con tabelle contenenti centinaia di milioni di righe (circa 4 miliardi di indirizzi IP possibili totali!)

Performance: L'accesso dev'essere chiaramente *sufficientemente vecole*, anche nel caso di ricerca fallita

Robustezza: Il sistema dev'essere accessibile anche in caso di *guasti*, quindi non ci dev'essere il c.d. "*single point of failure*".

Autonomia Amministrativa: Ogni organizzazione deve gestire "*i propri RR*" in autonomia, senza dover chiedere a nessuno.

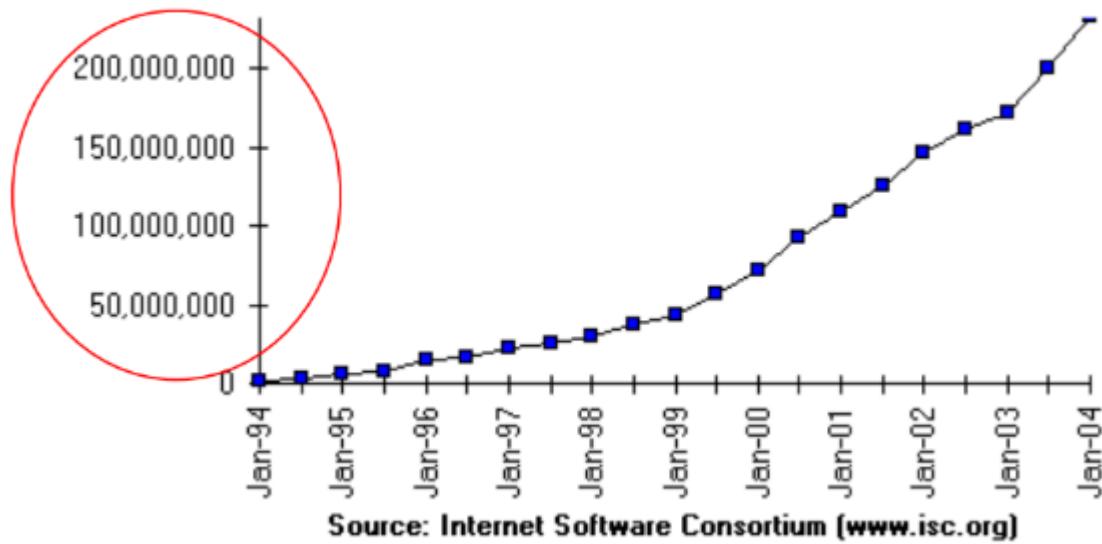
X

2. Cenni Storici al sistema DNS

1985. Jon Postel manteneva uno "*host file*" e ogni nodo aveva una *copia locale* dell'host file. Ogni modifica doveva essere comunicata a Jon Postel e poi essere ricopiata in tutti i nodi internet

Essendo che ai tempi i calcolatori erano *pochi*, questo sistema era fattibile.

1987. Primi protocolli che descrivono il DNS, RFC 1034 e 1035, che oggi funzionano ancora. La si considera come un "*miracolo*" in quanto dopo 40 anni il sistema è ancora in grado di funzionare, nonostante i "*carichi*" siano cresciuti in 6-7 ordini di grandezza



Aspetti Strategici del DNS

X

Criticità dell'infrastruttura DNS, e aspetti strategici dell'infrastruttura DNS.

X

0. Voci correlate

- Name Server

1. Criticità dell'Infrastruttua DNS

Oggi giorno, il *l'infrastruttura DNS* è una infrastruttura *enormemente importante*, una *infrastruttura critica*. Possiamo pensare che l'infrastruttura operi un ruolo analogo agli acquedotti o aeroporti nella società odierna.

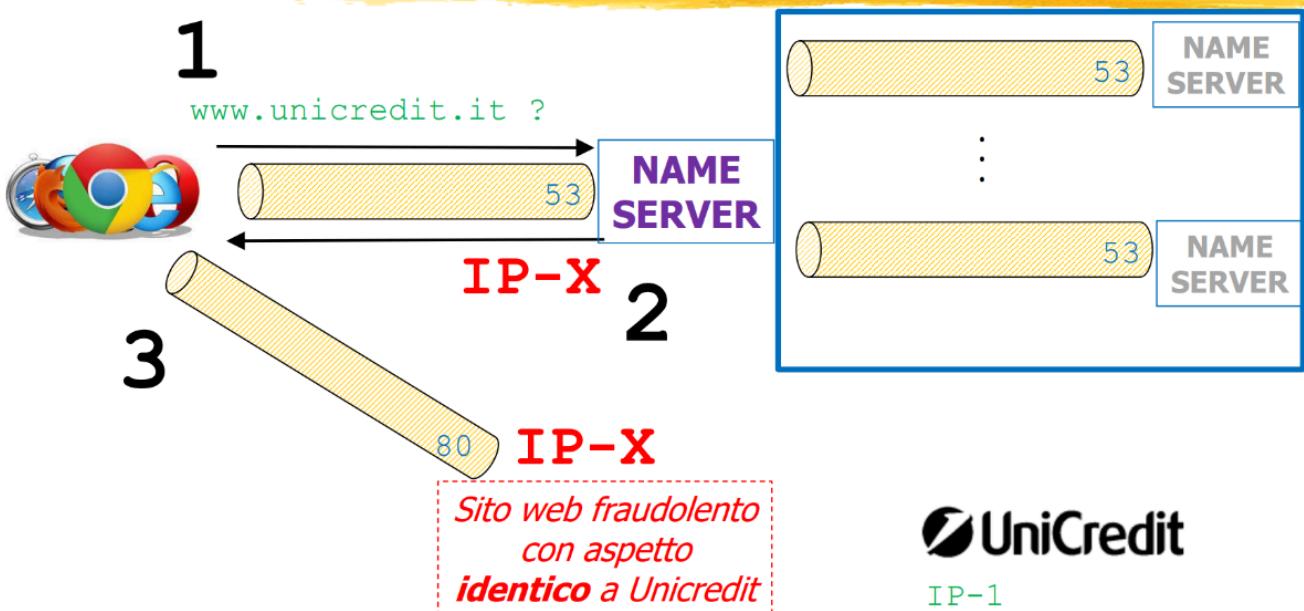
Pensiamo a due esempi che ne esemplificano la criticità:

Availability: Se non fosse più possibile tradurre i nomi che terminano con **.it**, come **inps.it**, le conseguenze sarebbero gravissime. Questo è un problema reale, con possibili *guasti* o *attacchi informatici mirati*

Hijacking: In certi casi è possibile *"manipolare"* un server DNS per ritornare *"indirizzi IP fasulli"*. Ad esempio, l'utente chiede il sito della sua banca; il DNS *"malevolo"* gli ritorna l'indirizzo IP di un sito che assomigli alla banca ma è un fake che raccoglie i dati. Anche questo è un problema reale, con tantissime cause possibili, tra cui:

- Attaccante modifica il contenuto a insaputa dell'amministratore
- L'amministratore è l'attaccante stesso
- L'attaccante modifica la configurazione DNS dell'utente, con tanti modi

Questo è un scenario realmente accaduto tante volte, coinvolgendo ad esempio il registrar GoDaddy e delle piattaforma di criptovalute.



Tuttavia, nel corso, assumeremo il seguente principio:

- Ogni nodo si comporta "*onestamente*"

Questo è dovuto al fatto che *tutti i protocolli Internet* seguono quel principio, e ai tempi non era contemplato che ci fossero degli eventuali pericoli.

 X

2. DNS: Strumento Strategico

OSS. Notiamo che accedere a servizi per *indirizzo IP* invece che per *nome* è praticamente "*infattibile*"; ciò rende il *DNS* come un'infrastruttura indispensabile per accedere a servizi. Da ciò emergono delle caratteristiche strategiche del DNS

Censura: Un governo può imporre ad ogni *Internet Provider sul territorio nazionale* di *bloccare DNS request* per certi nomi e di *bloccare traffico DNS verso indirizzi IP esteri*.

Sorveglianza: Un governo può imporre agli *Internet Provider* di tenere traccia gli *indirizzi IP* di un utente e le sue relative richieste DNS

Profilazione: Le aziende ottengono una *conoscenza "molto accurata"* su moltissimi utenti

Resource Record

X

Definizione di Resource Record. Cenni al DNS protocol: notazione intuitiva. Type principali: A, MX e CNAME. Osservazioni sul DNS response: additional RR.

X

0. Voci correlate

- Name Server

1. Definizione di Resource Records

Definizione. Un *resource record* è una tripla composta dal:

- *Nome* che identifica l'host (il nodo)
- *Type* che "caratterizza" la tipologia del valore
- *Valore* (l'indirizzo che identifica il calcolatore)

In altre parole, le *resource records* vanno a comporre la tabella di ricerca che viene realizzata dall'infrastruttura *DNS*. Fin'ora avevamo visto gli RR di tipo *A*, che identificano gli indirizzi IP.

Osservazione. In molti casi, ai nomi degli host associamo un'organizzazione, e pensiamo che l'indirizzo IP legato al nome siano gestiti dalla medesima organizzazione. Tuttavia, in realtà non c'è *nessun legame* tra l'organizzazione associata il nome e l'indirizzo IP (valore). Per il DNS, tutto questo è *irrilevante*.

X

2. Cenni al Protocollo DNS

Per il *protocollo DNS*, useremo solo la seguente notazione intuitiva:

Request: NAME TYPE ?

Response: il Resource Record completo, se c'è; altrimenti **NXDOMAIN**, che vuol dire una roba del tipo "RR inesistente" (vedremo come mai quando tratteremo i *domini*)

- Contiene una *response record*, che contiene o la *RR* richiesta o **NXDOMAIN**
- Può contenere ulteriori *additional record* (ulteriori RR), a discrezione del name server

X

3. Type Principali

MX. Gli *RR* di tipologia **MX** identificano il *dominio di una email* al *nome di host mail server di "quel dominio email"*.

CNAME. Gli RR di tipo **CNAME** identificano un name di un host in un altro nome *equivalente*.

Definizione di Dominio

X

Definizione di dominio. DNS response per domini con più RR. Osservazione: un dominio di CNAME dovrebbe essere solo composto da RR CNAME. Domain Tree: costruzione grafica, proprietà di sottodominio. Definizione di TLD e SLD.

X

0. Voci correlate

- Resource Records

1. Dominio

Definizione. Un *dominio* è un *insieme di RR con lo stesso nome*. Quindi, ogni RR appartiene esattamente ad un dominio.

Esempio. Supponiamo di avere gli seguenti RR:

```
google.com A IP1
google.com A IP2
google.com A IP3
```

Allora ho più host con lo stesso nome, ed essi appartengono allo *stesso dominio*.

Q. Se ho un dominio con > 1 componenti, come dovrebbe rispondere un *DNS* quando gli viene chieso **NAME A ?**?

Semplicemente è tutto a discrezione del *DNS*, quindi può decidere di dare *tutti*, *alcuni* o *solo uno* e i fattori che potrebbero motivare una delle scelte sono le seguenti:

- Scegliere posizioni geografiche più vicine
- Distribuire carichi a rotazione

Q. Se ho un dominio che contiene sia *CNAME* e non, cosa succede? Ho definizioni in conflitto...

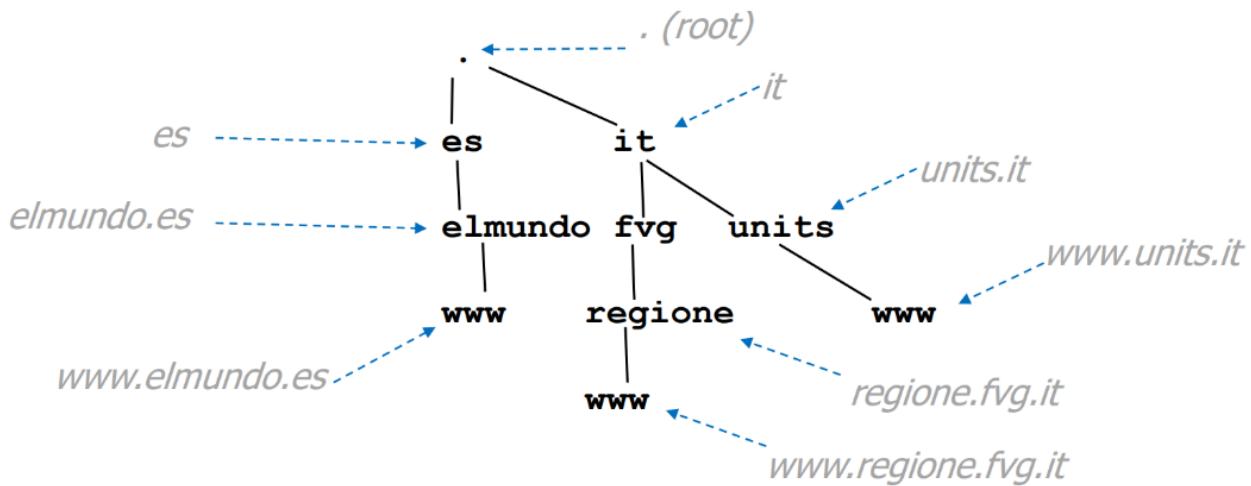
Generalmente questo non *dovrebbe accadere*, siccome per convenzione un dominio con *RR CNAME* non può contenere altri *RR di tipo diverso*. Non che sia impossibile, ma sconsigliatissima.

X

2. Domain Trees

Immagino di avere *tutti i domini esistenti*, come posso rappresentarli graficamente? Un modo per farlo è con i *domain tree*. In particolare, abbiamo un *grafo ad albero* dove:

- I *nodi* sono i *domini stessi*, e il nome è il "*label path*", ottenuto facendo la concatenazione con i loro parenti fino al nodo radice `.`.
- Gli *archi* rappresentano quindi un collegamento tra i domini (vedremo come nell'esempio)

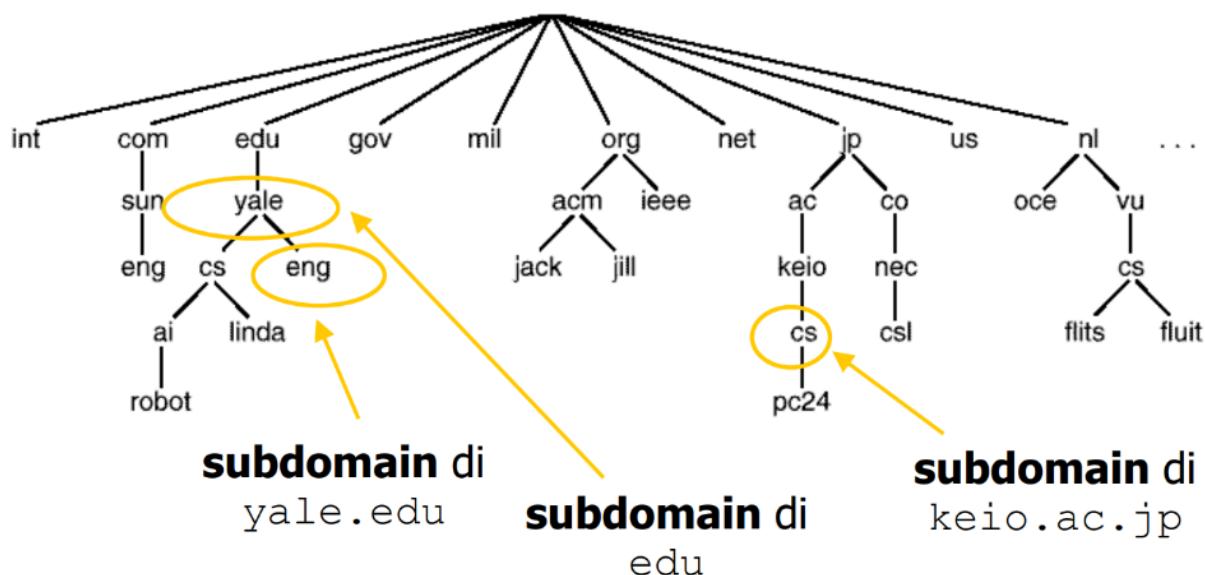


Notiamo che con questa rappresentazione *non vediamo gli RR stessi*, non sapremo dedurre *quanti e di quali tipi* RR contengono ogni nodo. Per farlo, bisognerà contattare il DNS.

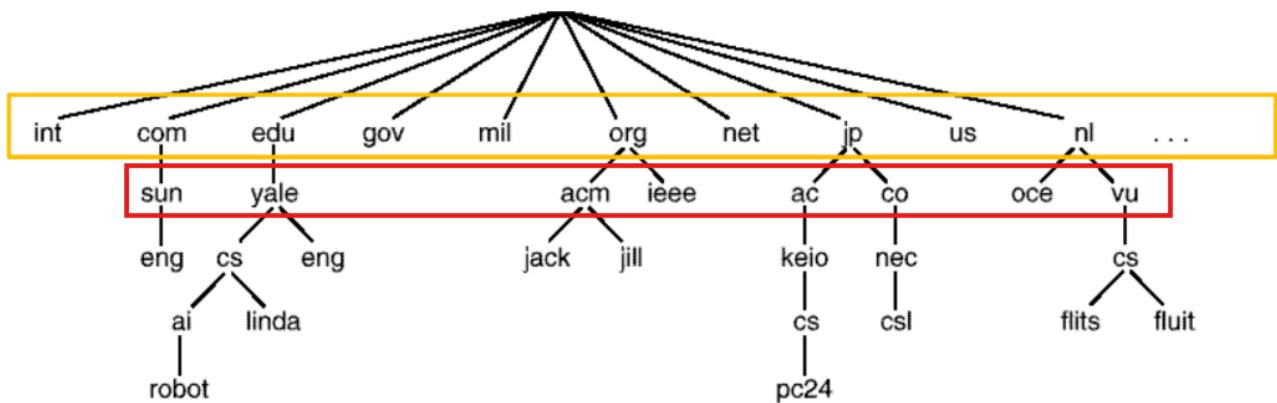
X

2. Subdomains

Principio. Ogni *dominio* è un *sottodominio* del *dominio padre*. Notiamo che ciò non implica che sono sottoinsiemi affatto, in quanto *RR di domini diversi* sono *disgiunti per definizione!*



Definizione. Un dominio si dice essere *Top-level domain* se è un subdomain della root. Se invece è subdomain di una TLD, allora si dice *Second-level domain*.



Creazione e Gestione dei Domini

X

Aspetti pratici sulla creazione e gestione di domini. Proprietari di domini: proprietà ricorsiva, delegazione di proprietà. Definizione di zona. Creazione di un dominio: caso comune e semplice. Vincoli sui nomi del dominio. Risalire al proprietario dal dominio: problemi pratici.

X

0. Voci correlate

- Definizione di Dominio

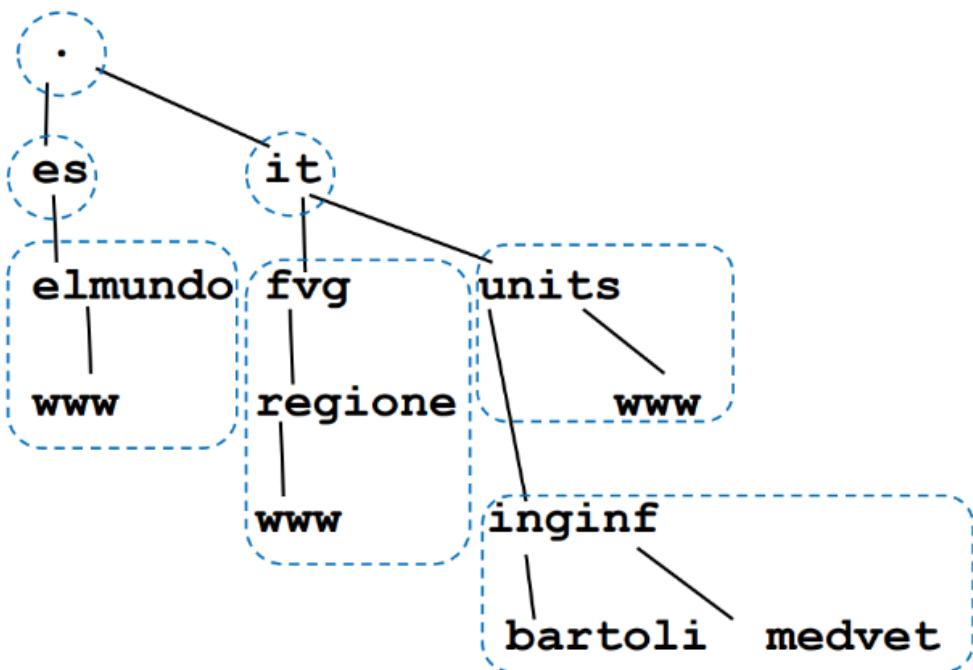
1. Proprietario di un Dominio

Ogni dominio ha un *proprietario* (individuo o entità giuridica), che ha *controllo completo su tutti gli RR*. Esiste quindi *quali* e *quanti* RR devono esistere nel dominio.

Proprietà. Ogni proprietario di un dominio è anche proprietario dei suoi *sottodomini* (ricorsiva).

Quindi, si avrebbe che il proprietario . è in realtà il proprietario di tutti i domini esistenti. Tuttavia, si ha che il *proprietario di un dominio può delegare ad altri la proprietà di un sottodominio*, spezzando quindi la ricorsività della proprietà.

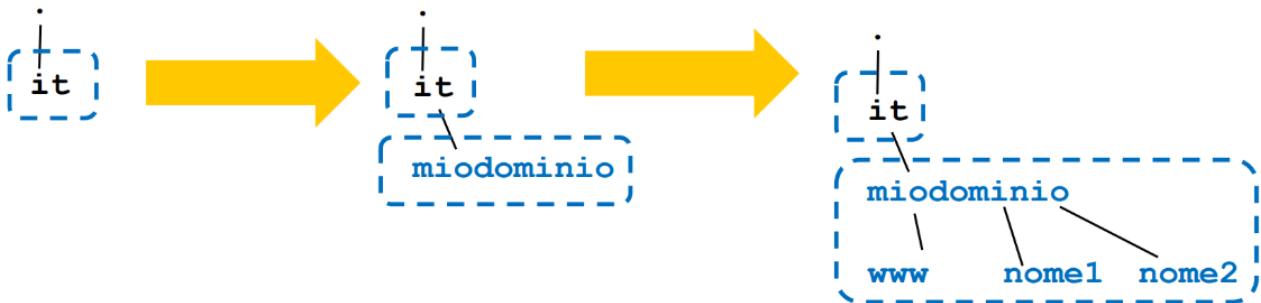
Da questo si ha un *partizionamento* della *Domain Tree* in *zone*, che costituisce un "pezzo contiguo" con lo stesso proprietario. Fisicamente, ci sono degli *RR* che descrivono queste suddivisioni in *zone*; per quanto riguarda il proprietario, si ha un metodo per identificarlo *molto complesso* ed è oggetto della seconda parte del corso.



X

2. Creazione di Domini

Per creare un dominio, assumiamo il *caso più semplice* (e comune): ovvero vogliamo creare un *dominio SLD* senza la possibilità di delegare la proprietà di sottodomini.



Il procedimento per farlo è banale. Si *"acquista"* il dominio da un *"DNS provider"*, che gestisce i dettagli tecnici e commerciali con i proprietari delle *TLD*. In particolare, si occuperanno la creazione e la delegazione della proprietà dietro le quinte.

Definiamo l'acquirente del dominio il *"registrar"*, e il *"DNS provider"* il *"registrar"*.

Q. Ci sono vincoli per scegliere il *nome* del dominio?

La convenzione delle DNS dettano che le *DNS* non forniscono nessuna garanzia a riguardo, ovvero il *nome del dominio* non pongono vincoli sull'*identità del proprietario*. Quindi, ogni nome può essere acquistato da chiunque (naturalmente con delle eccezioni, alcuni nomi non sono assegnabili o sono assegnabili con procedure di autenticazione offline).

Tuttavia, è possibile *opporsi all'assegnazione di un dominio* solo dopo quando è stata fatta l'assegnazione; in questo caso si va caso per caso, in certi casi si va ad effettuare procedimenti

legali, in altri si disattiva il dominio.

Osservazione. Quindi è possibile comprare un *nome dominio* che sembri che sia associata ad un'organizzazione o persona (ma in realtà non lo sono), e quindi poter ingannare le persone sull'internet.

2.1. Risalire il Proprietario dal Dominio

In certi casi è importante poter *risalire al proprietario* dal *dominio*. Per esemplificare:

- Rilevazione attività fraudolente
- Attività illecite, come installare malware "Command and Conquer"
 - In parole brevi, un malware C&C funziona mandando segnali "*segreti*" camuffate sotto richieste DNS verso un *dominio fasullo*. In questo caso, posso codificare richieste e risposte...

Per farlo, ricordiamo che le proprietà vengono *passate*, quindi in genere il *proprietario di un dominio padre* dovrebbe *"conoscere"* l'identità del dominio *"figlio"*. In pratica, possiamo:

1. Contattare **owner(root)** e identificare **owner(TLD)**
2. Contattare **owner(TLD)** e conoscere **owner(SLD)** oppure conoscere **DNS provider(SLD)**
3. Contattare **DNS provider(SLD)** e conoscere **owner(SLD)**

Per procedere a ulteriori livelli, solo le autorità giudiziare possono iniziare procedimenti per conoscere i proprietari.

I passaggi 1., 2. e 3. possono essere fatto con i *servizi web "whois"*, che *"navigano tra delegati"* a partire da root.

Osservazione. Notiamo che in certi casi, per motivi di svariata natura, i *Domain provider* possono decidere di offuscare intenzionalmente i dati. Le autorità giudiziare possono comunque obbligare questi registrar a fornire il registrant, ma è un procedimento *lungo e complesso*... (in quanto è probabilmente un procedimento internazionalmente e non sicuramente si ha la collaborazione del registrar)

Da questa osservazione deduciamo che l'identità del *registrant* tende ad essere *facilmente offuscabile* o *falsificabile*... gli unici casi in cui l'identità è certa sono gli TLD e alcuni SLD *"importanti"*.

Concludiamo dicendo che questa infrastruttura è critica e *basata su nomi*, che rende facile l'impersonazione sull'internet.

"Web"

Nozioni sul World Wide Web

X

Introduzione al World Wide Web. Definizione del World Wide Web, Web server e Browser (web client). Definizione di URL e le sue componenti. Azioni di un browser. Intuizione del protocollo HTTP. Azioni di un web server. Trarre documenti da URL. Osservazioni sull'URL: ruolo di localizzazione, confronto con i QR code, rappresentazione degli url negli address bar e HTTP redirection.

X

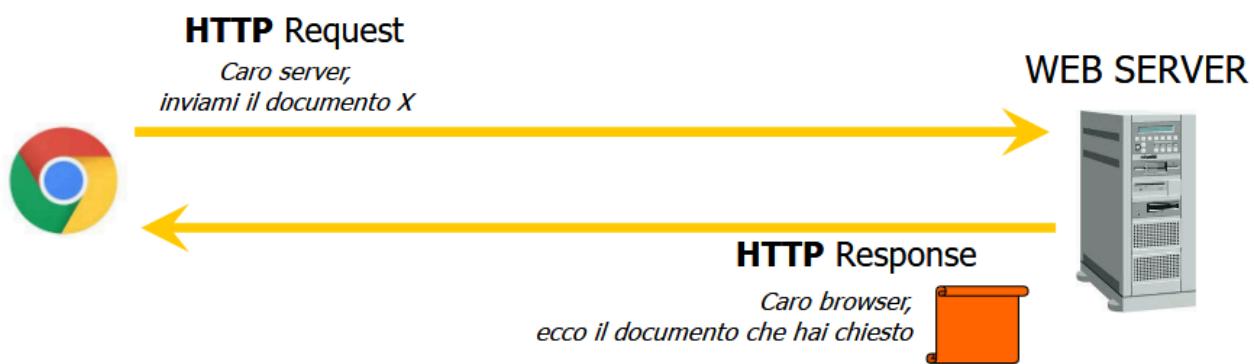
0. Voci correlate

- Comunicazione tra Processi

1. Definizione di World Wide Web

Il *World Wide Web* è un *insieme di documenti "collegati" tra di loro*, in vari formati. Il più comunemente usato è *HTML* (vedremo questo aspetto nei dettagli successivamente).

Nel *Web*, chiamiamo i *server* come i *web server* ed essi forniscono il servizio di *prelevare documenti*; invece i *browser* (*web client*) prelevano e visualizzano i documenti. In altre parole, nel Web si ha che i documenti vengono inviati dai web server e vengono visualizzati nei browser.



X

2. URL

L'*URL* (*Uniform Resource Locator*) è un *identificatore univoco* (anche se, dopo vedremo che in realtà sarebbe più accurato dire che è un *localizzatore*) di ogni documento del World Wide Web e consiste in una sequenza di caratteri. In particolare, la sequenza è suddivisa in tre parti:

- **Protocollo**: il protocollo usato per comunicare col web server, ad esempio HTTP, HTTPS, FTP, eccetera...
- **Nome del Server**: a cui effettuare la richiesta. NATURALMENTE bisognerà risolvere il nome mediante il local resolver nel sistema operativo
- **Nome del Documento**: identifica il documento stesso da trovare; quindi un file

Per convenzione l'URL è formattato nel seguente modo:

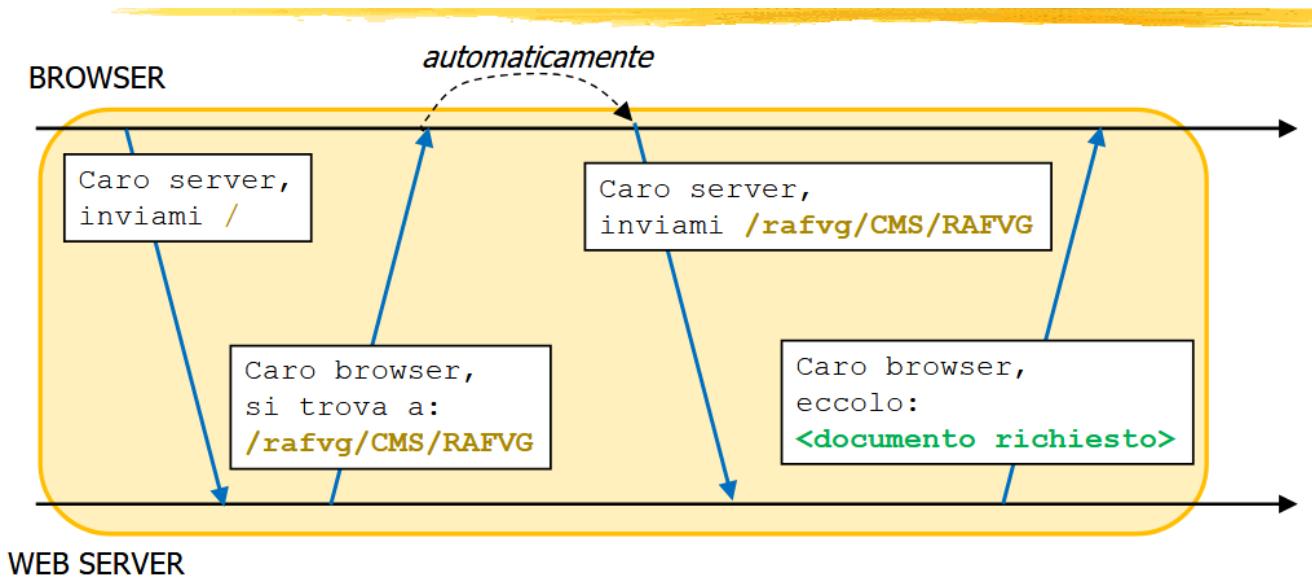
<protocol>://<server_name>/<document_name>

- Osserviamo che nel **server name** non è necessario iniziare per **www**.

QR Code. Il QR code è un'immagine che codifica una *sequenza di caratteri*, e la sua tecnica di codifica è *"robusta"* rispetto alle possibili trasformazioni di gruppo nelle immagini. Tipicamente i QR code vengono utilizzati per codificare gli URL; tuttavia ciò potrebbe essere pericoloso, in quanto l'occhio umano non è in grado di trarre un'idea dal QR code. Almeno, con gli URL si ha un'idea vaga!

Address Bar. Gli address bar dei browser sono dei campi che contengono una rappresentazione dell'URL. Nel passato gli URL venivano rappresentati *esattamente*, oggi invece si ha una *"rappresentazione semplificata"* che dipende dal browser. Ad esempio, nei browser dei dispositivi cellulari, si offusca la parte del **protocol** e viene rimpiazzata con un'icona. Oppure, un'altro caso frequente è quando il **browser** inserisce il carattere **/** alla fine del document name alla fine quando non c'è.

Inoltre, notiamo che i browser potrebbero *"decidere"* di mostrare documento con URL diverso in address bar. Questo è dovuto al fenomeno di **HTTP redirection**, che vedremo bene dopo. Per ora, la illustriamo intuitivamente con il seguente diagramma.



3. Azione Processi e Cenni al Protocollo HTTP

3.1. Web Browser

Il web browser seguirà i seguenti passaggi. Dato *in configurazione* l'URL:

- Trovare l'*IP server*, chiamando la DNS resolver
- Connetersi all'IP server con la porta specificata
- Mandare una richiesta HTTP per "*chiedere il documento*"
- Ricevere la risposta
- Chiudere la connessione
- Visualizzare il documento se possibile, o fare qualcos'altro (dipende dal tipo di documento!)

Osservazione. Notiamo che quindi *un documento* richiedere una *coppia* di *HTTP request* e *HTTP response*. Tuttavia, in realtà certe tipologie di documenti possono essere "*mandati a pezzi*", come nel caso di streaming o trasferimento di file più grandi.

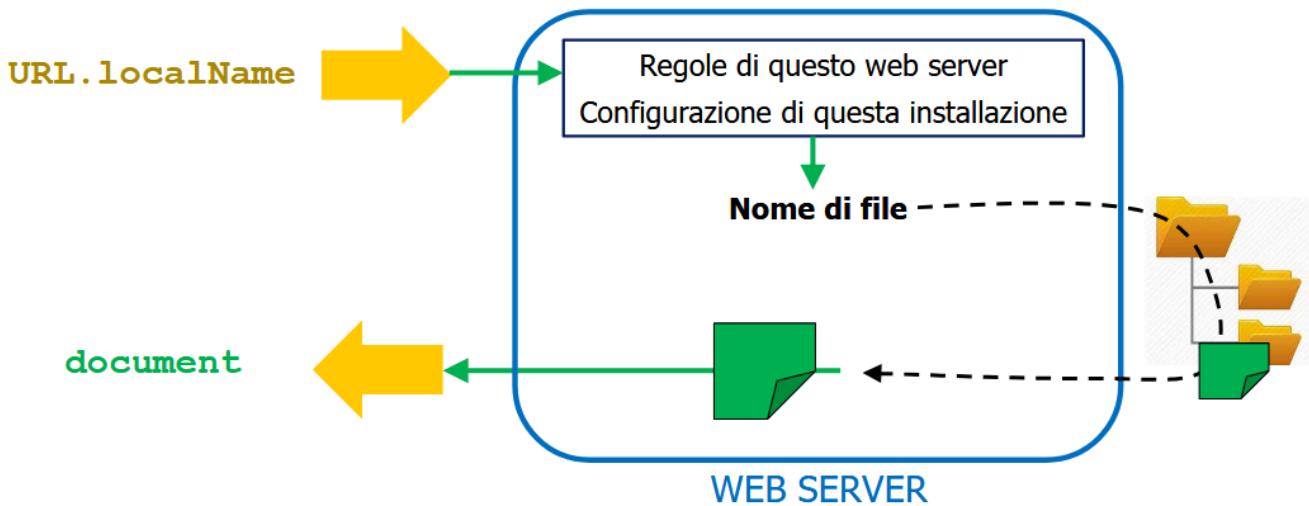
3.2. Azioni Web Server

Il web server seguirà i seguenti passaggi:

- Creare un socket e aspettare connessioni
- Quando ne riceve una:
 - Aprire il communication socket
 - Ottenere il *document name* specificato nell'URL
 - Costruire una risposta, ad esempio nel caso positivo costruire la risposta che contiene il documento
 - Inviare risposta

Osservazione. Notiamo che il passaggio di *associare il document name al documento effettivo* non è un passaggio banale, per cui esistono *molte possibilità*. Questo processo dipende da come viene configurato ogni web server, e non ci sono dei standard particolari. Comunemente si ha il *caso intuitivo*, ovvero il *document name* è interpretato come un *path relativo* su una directory del web server. Ci sono altre convenzioni aggiuntive, tra cui:

- Se non esiste l'estensione, allora viene interpretata come un file html
- Se è una directory vado a cercare il file **index.html** della directory
- Se non c'è niente (quindi c'è solo il , cerco **index.html**



Notiamo che, come conseguenza, si ha che in realtà l'*URL* è un *localizzatore*: infatti, ogni volta che sposto il file il suo URL deve cambiare.

3.3. Intuizione del Protocollo HTML

Il protocollo HTML è un protocollo text-based. Per inviare richieste o risposte, basta formattare il "*messaggio*" nei seguenti modi:

Richiesta:

```
GET <URL> HTTP/1.1
<varie righe di testo>
<linea vuota>
```

Risposta:

```
HTTP 1.1 <codice_esito>
<varie righe di testo>
<linea vuota>
<documento richiesto (se c'è)>
```

Le "*varie righe di testo*" servono per facilitare le interpretazioni nelle richieste o risposte.

Vedremo dopo cosa precisamente sono...

Struttura dei Contenuti Web

X

Struttura dei contenuti Web. Tre tipologie di file: HTML, CSS e Javascript. Metodi per hostare web server in pratica. Cenni a HTML, CSS e Javascript.

X

0. Voci correlate

- Nozioni sul World Wide Web

1. Formato dei Contenuti Web

Il contenuto dei documenti visualizzati dai browser può essere diviso in tre parti:

- *HTML*: Contenuto e struttura
- *CSS*: Stile (contenuto VISIVO)
- *JavaScript*: Azioni

HTML e CSS sono dei linguaggi "*markdown*", JavaScript è invece un vero e proprio linguaggio di programmazione (maledetto e complicatissimo...)

Per hostare un *web server*, abbiamo due metodi:

- *Self-hosting*: sul "nostro PC", accedendo il nome "localhost" oppure l'IP 127.0.0.1
- *Remoto*: Un "vero e proprio" sito web, tuttavia è necessario avere un nome DNS da qualche parte

Adesso vediamo un paio di cenni ad ogni "*parte logica*" di un documento, come si struttura.

X

2. HTML

HTML è un linguaggio per descrivere "*simple structured documents with in-lined graphics*" e ipertesti, immagini, eccetera...

Come *visualizzatore HTML* si può semplicemente usare un *browser*, oppure anche l'interfaccia di un *IDE*. Un visualizzatore HTML semplicemente interpreta il documento e nè da un'interpretazione visiva.

HTML è formato da due elementi: *testo* (content) e *tag* (structure). Ogni tag fornisce delle informazioni aggiuntive sul testo, tra cui la *struttura* o la sua *formattazione*. I tag vanno *aperti* e *chiusi*.

```
<!doctype html>
<html>
  <head>
    <title>This is the title of the webpage!</title>
  </head>
  <body>
    <p>This is an example paragraph. Anything in the
    <strong>body</strong> tag will appear on the page, just like this
    <strong>p</strong> tag and its contents.</p>
  </body>
</html>
```

Come si vede nell'esempio, ogni documento è strutturato in due parti:

- Inizia l'**head** e contiene il **titolo** e altri elementi "**background**" (come script)
- Poi c'è il **body** che contiene tutto il contenuto della pagina, quindi titoli, testo, liste, eccetera...

Inoltre, ogni **tag** può contenere uno o più **attributi**. Per sapere *quali attributi* e a quali valori assegnare, bisogna consultare l'RFC apposita.

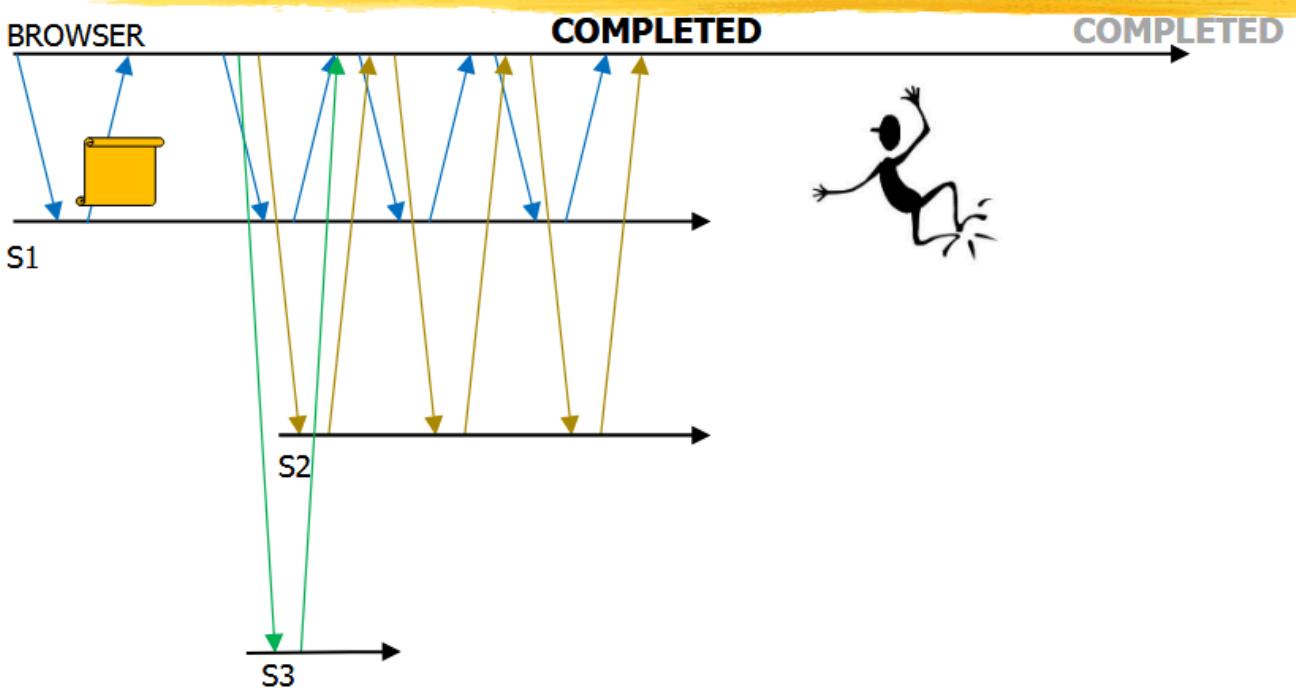
2.1. Tag Importanti

Vediamo alcuni tag importanti.

Hyperlink. Il tag ` ... ` è il **tag** che va a creare un **hyperlink**, ovvero un collegamento ad un altro sito esterno sul web.

Image. `` va a inserire un'immagine nel documento. Osserviamo che ogni volta che incontriamo un tag del genere, il web browser DEVE prelevare l'immagine effettuando ulteriori richieste a web server. Teoricamente è possibile inserirli direttamente, ma non lo faremo mai.

Iframe. `<iframe src='...'></src>` inserisce un altro documento nel documento mostrato. Come con le immagini, il browser deve prelevarlo tramite richieste http (e DNS).



Q. Cosa fa un browser?

Si vede il seguente pseudocodice che descrive le azioni di un browser:

- Per ogni "tab" aperta,
 - Interpretare il codice *HTML*
 - Prelevare *automaticamente* tutti i tag HTML che prevedono richieste ad altri server web, come le immagini, iframe, eccetera...
 - Ricordiamo che è necessario contattare il proprio *DNS*, poi aprire una connessione col web server, fare una request, ottenere la response e infine di chiudere la connessione
 - Eseguire script
 - ...
 - Disegnare in base ai documenti correnti

Osserviamo che l'azione vale *per ogni tab aperta* e quindi si parallelizza i processi (il come dipende da browser in browser). Inoltre, la visualizzazione dei documenti è *incrementale*, quindi carica i contenuti "poco appena".

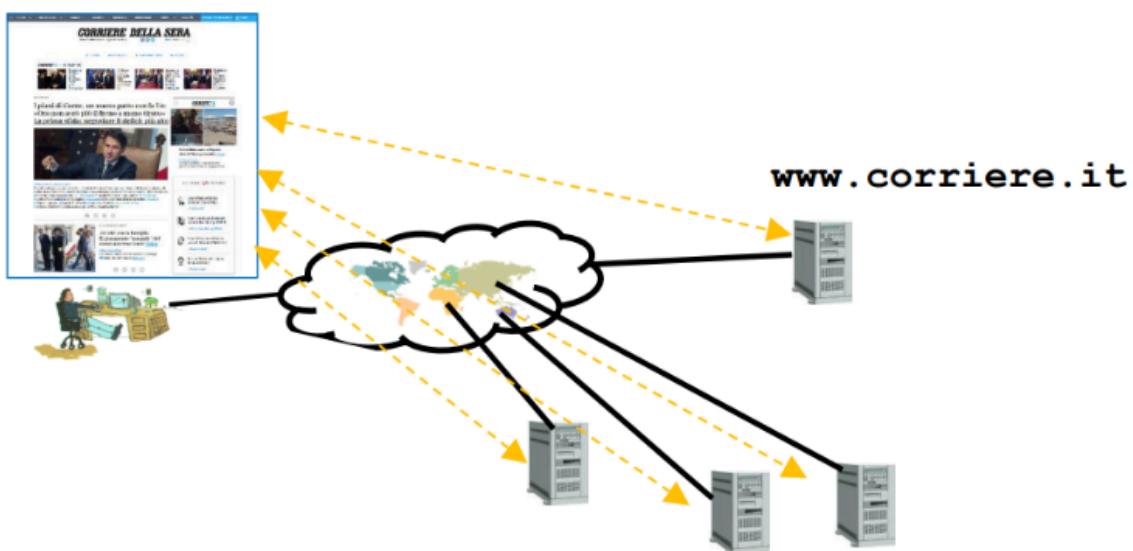
Nozione di pagina e sito web. Necessità di contattare più server per una pagina web. Chiarimenti sulle terminologie: documento/risorsa, pagina e sito. Implicazioni. Complessità dei siti web reali: come un browser gestisce un URL con solo name server. URL relativi e assoluti.

0. Voci correlate

- Nozioni sul World Wide Web

1. Pagina Web

DEFINIZIONE. Una *pagina web* è il *risultato di molti documenti prelevati automaticamente*, tra cui immagini, iframe, script/css. La barra degli indirizzi del browser contiene solo l'URL del "documento contenitore"

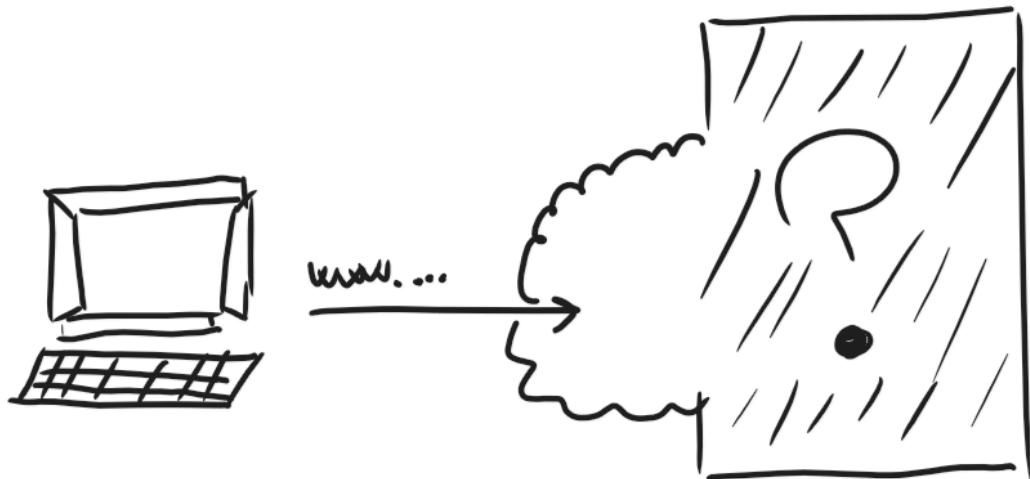


Notiamo che ciò vuol dire che una pagina web risiede su *molti web server* organizzati da enti diverse. Quindi, il browser deve sempre contattare un *insieme di server*, mai prevedibile o controllabile dal lato user.

Ciò ha delle implicazioni, e sono le seguenti:

- *Analytics*: grazie a questo sistema è possibile raccogliere informazioni su *chi va* sul sito
- *Supporto pubblicità e profiling*

Quindi, se una organizzazione è "*presente*" su molti siti web e può riconoscere lo stesso browser, allora riesce a "*conoscere tutto di noi*".



X

2. Terminologie Web

Non c'è uno *standard universalmente accettato* sulla terminologia dei contenuti web, tuttavia nel corso definiamo alcuni termini:

- *Documento/Risorsa*: Ciò che identificano gli URL
 - *Pagina*: Ciò che il browser visualizza in un tab
 - *Sito*: Insieme di pagine con lo stesso server name
-
- X
-

3. URL Relativi

Vediamo un paio di *aspetti tecnici* dei siti web, in quanto per quanto concerne gli URL.

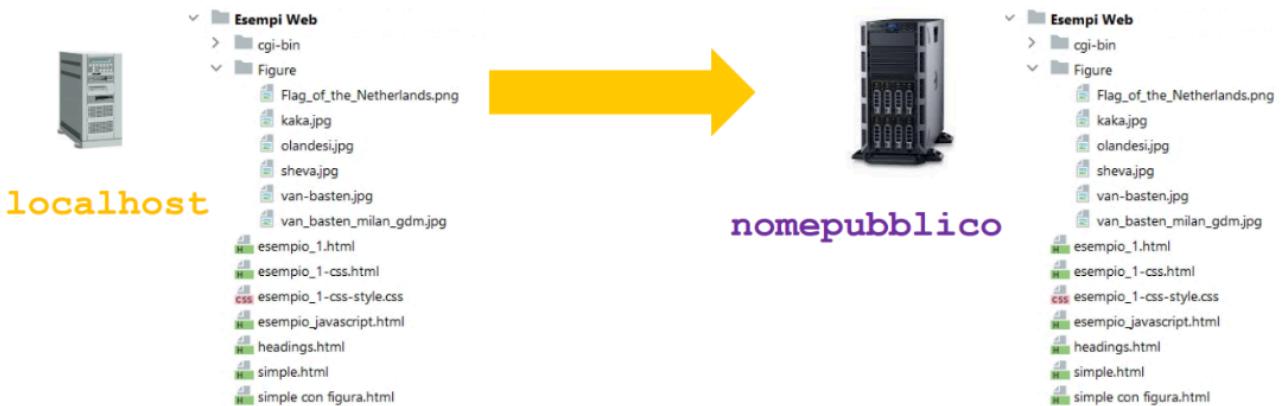
Osservazione. Quando in un file html si ha un tag con attributo `src`, è possibile mettere un URL del tipo `<tag src='myfile.x'> <\tag>`. Come fa il browser a sapere *da quale* web server prelevare? Dell'URL conosce solo la terza parte (ovvero il document name)!

Molto chiaramente si tratta di un *URL relativo*, quindi semplicemente basta copiare le prime due parti (protocollo e name server) da "*dove ho prelevato il documento*".

Vediamo un esempio dove gli *URL relativi* sono utili. Supponiamo di avere un *sito web* sul server S_1 , e poi di voler spostarlo su S_2 . Se ogni URL fosse *assoluto*, allora questo spostamento creerebbe degli problemi in quanto diventano "*invalidi*"; quindi bisognerà rilocarizzare tutto da capo...

- Si può farlo automaticamente, ma è complicato
- Farlo a mano è ancora peggio

Un classico caso di questo esempio è lo spostamento dal *server web test* al *server web produzione*: usando gli URL relativi, ci togliamo la maggior parte del lavoro da fare.



Una cosa da **ASSOLUTAMENTE NON FARE** è creare i cosiddetti "*URL relativi assoluti*", in quanto è concettualmente sbagliata.

Dettaglio. Se un URL relativo inizia con , allora l'URL relativo è "*ancorato*" alla directory del server name.

<https://www.S2.com/docs/index.html>

```
...  
<IMG SRC="MyGif/fig.gif">  
..."
```

<https://www.S2.com/docs/MyGif/fig.gif>

<https://www.S2.com/docs/index.html>

```
...  
<IMG SRC="/MyGif/fig.gif">  
..."
```

<https://www.S2.com/MyGif/fig.gif>

Tutorial Sito Web

X

Gestione di siti web. Ruolo del web developer. Come fare testing: sul proprio pc o su un altro pc. Come "pubblicare" un sito web: self-hosting e acquistare un dominio DNS, affittare un calcolatore e acquistare un dominio DNS, usare un servizio di web hosting o usare un website builder. Domain custom e non custom.

X

0. Voci correlate

- Pagina e Sito Web

1. Come fare un Sito Web

Nel contesto dei siti web, ci sono due ruoli:

- Il *web developer* sviluppa il sito, quindi crea i file necessari e poi configura, esegue e mantiene il web server; anche se in realtà i due compiti sono spesso effettuati da ruoli diversi
- L'*utente* semplicemente accede al sito, quindi inserisce l'URL nel browser

Vedremo come *fare un sito web* come *web developer*

Step 1. (*Fare prove*)

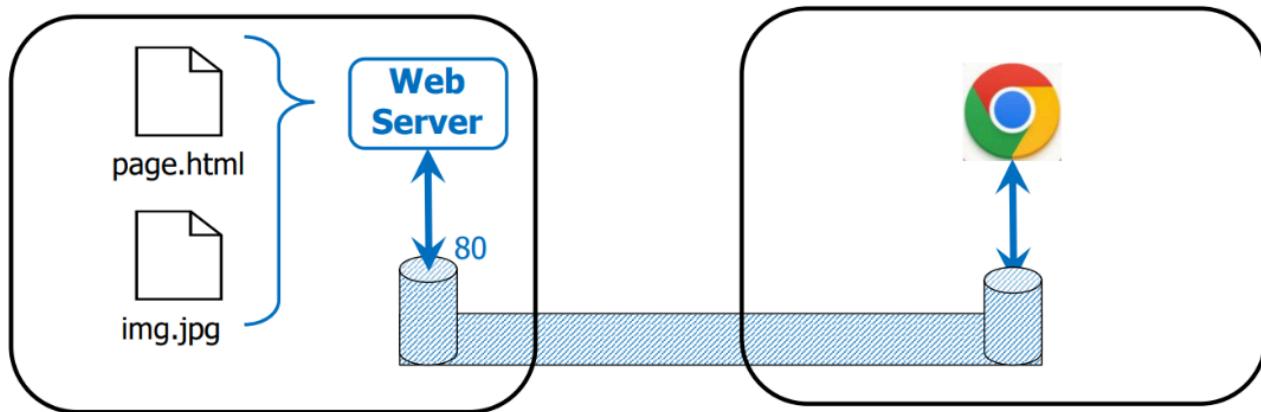
Un primo passo da fare è quello di effettuare delle prove sul sito web, quindi fingersi un utente. Ci sono più modi per fare il testing:

- *Stesso PC*, quindi il web server viene hostato localmente
- Un altro PC collegato alla *stessa rete Wi-Fi Personal*, quindi con le stesse "*credenziali*" (esempio: Eduroam no, stessa casa sì)

Vediamo il primo caso, quindi avere un *web server sul proprio PC*. Per testare, effettuare il seguente procedimento:

1. Aprire una shell
2. Posizionarsi nel folder con i file del sito
3. Lanciare il web server
4. Usare un browser e mettere url `http://localhost` oppure `http://127.0.0.1` oppure un nome a piacere sul `hosts` file

PC



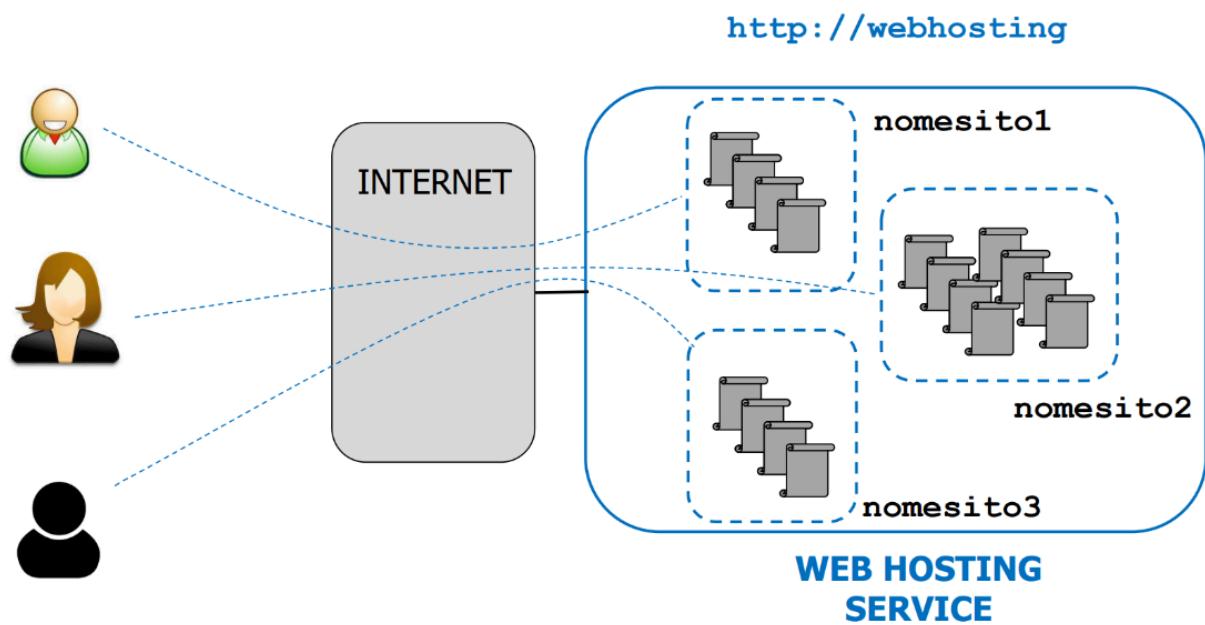
Step 2. (*Renderlo accessibile*)

Una volta soddisfatti del sito che si ha, bisogna "*pubblicarlo*" e quindi renderlo accessibile a tutti. Anche qui ci sono svariati modi per farlo:

1. Ottenere un *dominio DNS* ed eseguire il web server su un calcolatore; solitamente lo acquisto da un DNS provider
 1. Il calcolatore è *mio*
 2. Il calcolatore è *affittato* da un'azienda / ente / organizzazione / qualcuno
2. Usare un servizio *web hosting*, ossia un processo web affittato. Solitamente basta copiare i contenuti e specificare il dominio di cui sono proprietario
 1. Alcuni servizi di web hosting forniscono la possibilità di usare un *proprio dominio (custom domain)*, creando degli RR di tipo CNAME. Notiamo che comunque il "*proprietario*" del dominio è il web developer, quindi deve avere delle eventuali "collaborazioni"
 2. In altri casi il dominio viene *scelto dal servizio di web hosting (non-custom)*; in questo caso il nome è vincolato parzialmente dal servizio, siccome il dominio può essere proprietà del servizio. La parte "*comoda*" consiste nel fatto che non bisogna acquistare nessun dominio
3. Utilizzare un servizio di *website builder*, che fornisce sia il web server che un editor per modificare il sito. Quindi non bisognerà possedere nessun file di nessun tipo per creare il sito stesso.

Facciamo delle osservazioni:

- 1.1) Non posso usare questo metodo con un "*collegamento da casa*", siccome quel tipo di indirizzo IP tende a cambiare frequentemente (è *privato e dinamico*); occorrerà invece configurare un *indirizzo IP pubblico e statico*
- 2.2.) Siccome i clienti effettuando le richieste GET sullo stesso server web del servizio di web hosting, bisognerà capire a quale sito si riferisce la *terza parte dell'URL* (document name)



Client Side Scripting

X

Elementi di client-side scripting. Cosa può fare JavaScript allo browser. Dove sono gli script. Cenni a JavaScript. Osservazione: implicazioni di sicurezza (eseguire script JS sui browser), meccanismo di sandbox.

X

0. Voci correlate

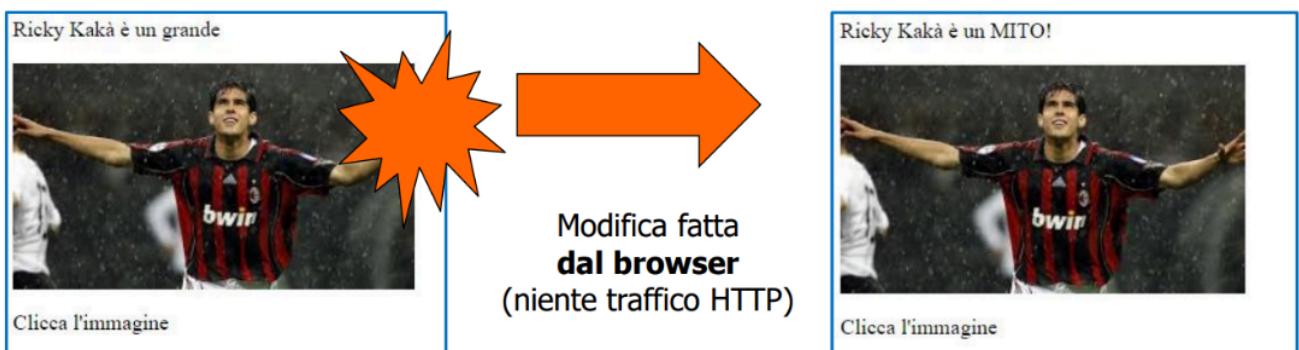
- Struttura dei Contenuti Web

1. Concetto di Client-side Script

Ricordiamoci che una *visualizzazione* ottenuta di un documento è *immutabile* e *indipendente da azioni utente*. Allora come posso aggiungerci delle interazioni, modificando la visualizzazione della pagina, senza dover generare ulteriore traffico?

Client-Side Scripting: Abbiamo degli *script* che sono in grado di *descrivere azioni* che *il browser deve effettuare*, come ad esempio aggiungere/eliminare/spostare elementi HTML, modificare attributi o proprietà CSS. Quindi, l'esecuzione degli *script* può controllare completamente l'aspetto visivo del documento!

Quindi, il concetto di *client-side scripting* consiste in creare dei *codici* da essere eseguiti (*interpretati*) dal browser, usati tipicamente per modificare l'aspetto di un documento visualizzato.



Ciò è possibile grazie al concetto di *render tree* dei browser, ricordiamoci che ogni browser esegue gli seguenti step:

- Per ogni "*tab*" aperta,
 - Interpretare il codice *HTML*
 - Prelevare *automaticamente* tutti i tag HTML che prevedono richieste ad altri server web, come le immagini, iframe, eccetera...

- Ricordiamo che è necessario contattare il proprio *DNS*, poi aprire una connessione col web server, fare una request, ottenere la response e infine di chiudere la connessione
- **Eseguire script**
- **Disegnare in base ai documenti correnti**

Negli ultimi due step si mantiene il *render tree corrente*, e gli script dicono al browser di modificare la rappresentazione nel susseguirsi di certi "eventi".

Gli script possono essere trovati in più "luoghi":

- All'interno del documento *HTML* stesso, racchiusi tra tag `<script> ... </script>`. Sono associati a "event handler" di elementi HTML specifici
- In documenti separati linkati dal documento HTML, si usa lo tag `<script src="...">`; il browser si cercherà lo script automaticamente. Quindi in realtà può trovarsi *ovunque*

X

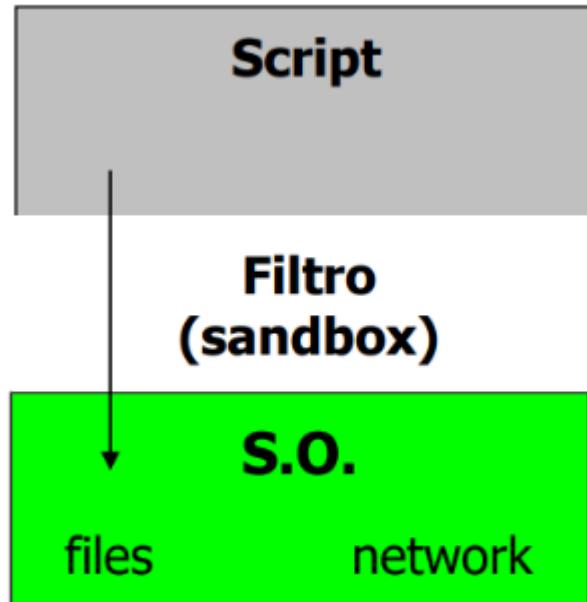
2. Java Script

Il linguaggio standard di fatto per scrivere gli script è *JavaScript*. E' un linguaggio interpretato, quindi i programmi sono sempre in *forma sorgente*; questo per rendere conveniente far eseguire gli script ai browser, siccome i loro calcolatori su cui vengono eseguiti possono essere *diversi*.

Inoltre *Java Script* può fare molte altre cose (non approfondiremo), intuitivamente si può pensare che di fatto uno script in JS è "*quasi equivalente*" ai programmi che si installano sui computer.

Ciò implica delle questioni di sicurezza, in quanto ogni volta che visualizzo una *pagina*, in realtà eseguo anche "*programmi*" sulla mia macchina e scritto da qualcun altro.

Quindi è stato designato il meccanismo di sicurezza "*Sandbox*", per cui uno script non può accedere direttamente alle risorse locali. In un certo senso, è un *filtro* per gli script JS.



Tuttavia, ciò non risolve tutto in quanto gli script JS possono comunque compromettere siti (e.g. mostrare dei dati che in realtà non corrispondono ai dati nel database); la libreria ha *controllo completo* sul web!

Documenti Statici e Dinamici

X

Documenti statici e dinamici. Remind del sistema di document retrieval. Documento dinamico: layout generale, vari punti di vista. Esempio: web app. Architettura back-end. Osservazione: nessuna differenza tra documento statico e dinamico per il browser. URL con query string.

X

0. Voci correlate

- Pagina e Sito Web
- Nozioni sul World Wide Web

1. Documenti Dinamici e Statici

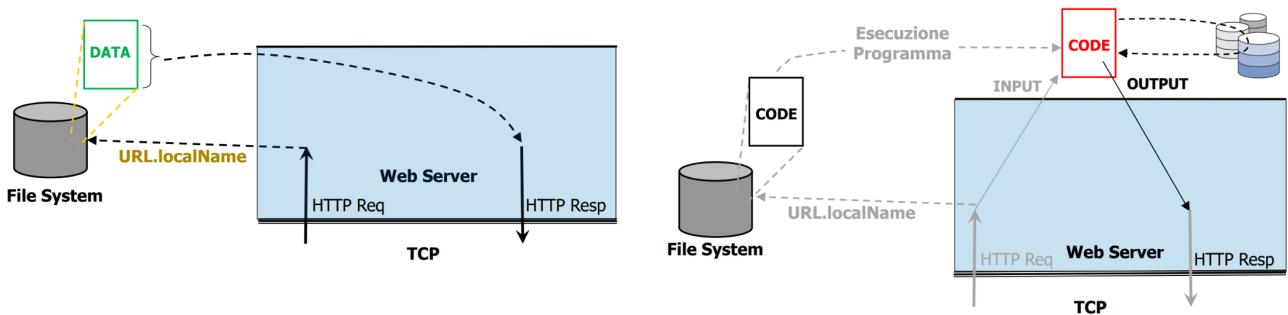
Ricordiamo che generalmente un "*document*" è un *file* sul web server, di cui il nome si ottiene da *URL.localName* con *regole* che dipendono dal web server.

Q. Quindi il sito ha preparato già *tutti i documenti possibili*? Ad esempio, è possibile che il sito di Trenitalia abbia tutti i documenti comprensivi di tutte le ricerche possibili?

No, un caso molto più comune è quello dei *documenti dinamici*; ovvero, il *document* è *costruito* da un *programma*. In particolare, *estrae parametri dalla HTTP request*, cerca delle informazioni opportune e crea il documento.



Quindi, se con un documento statico un *URL* localizza un "*documento già esistente*", invece con un *documento dinamico* la si crea alla ricezione della HTTP request.



Non ci sono convenzioni particolari per "*codici*" che creano le pagine, sono *specifici per ogni sito web*.

ESEMPIO. (*Web app*)

Nel caso delle *web app*, abbiamo che il documento è costruito da un programma indipendente dal web server e che deve seguire le convenzioni del web server. Ci sono molte tecnologie back-end per farlo, e non li vedremo. Nel corso assumiamo che esista un *programma web server* che "*fa tutto*".

Osservazione. Nel caso in cui un *web server* riceve una HTTP request per un URL che identifica sia una pagina statica che dinamica, allora essa usa le *sue regole* per decidere se tornare il contenuto statico o dinamico.

Osservazione. Notiamo che un *documento statico* non è *immutabile*, siccome posso comunque *avere degli script* che modificano il suo aspetto

Osservazione. Osserviamo che per il *browser* non importa se il documento tornato nella HTTP response è statico o dinamico; alla fine il web browser effettua le stesse zioni. Ci sono certamente degli "*indizi*" per dedurre se un documento è statico o dinamico, ma sono irrilevanti a fine pratici. Un primo esempio è quello di vedere l'estensione del local name.

Uno dei tanti indizi per capire se un documento sia statico o dinamico è quello di identificare le *URL query string*, alla fine. Essi vengono usati *tipicamente* su *web server*, e contengono dei *parametri input del programma*. Intuitivamente, la sintassi è qualcosa del tipo:

```
<protocol>://<name_server>/<my_file>?s1=t1;s2=t2;...;sn=tn
```

Di solito servono per effettuare query su database.

Protocollo HTTP

X

Protocollo HTTP. Introduzione al protocollo HTTP, le sue proprietà. Sintassi delle richieste e risposte HTTP. Concetto di headers. Osservazione: come viaggiano le trasmissioni HTTP su connessioni TCP. Approfondimenti sugli header da sapere: header host, referer, location, user agent, set-cookie/cookie, content-type/length.

X

0. Voci correlate

- Nozioni sul World Wide Web

1. Introduzione a HTTP

Il protocollo **HTTP** è estremamente complesso, ne vedremo solo un suo sottoinsieme e semplificheremo alcuni aspetti.

HTTP (*Hyper Text Transfer Protocol*, RFC 2616) è un protocollo *a richiesta/risposta* text-based in ASCII. Il protocollo è indipendente dal livello di trasporto, solitamente nel WWW si usa **TCP**; in altri casi si può usare anche **TCS/TCP**.

HTTP è uno dei protocolli più usati, infatti nel 1997 il 75% del traffico internet erano richieste HTTP; oggi ne sono circa il 98%.

X

2. HTTP Request e Response

Come accennato, le richieste HTTP sono strutturate nel seguente modo:

```
GET <URL> HTTP/1.1
<varie righe di testo>
<linea vuota>
```

Nelle "varie righe di testo" ci sono le *request headers*, ossia un insieme di linee contenenti delle coppie *Header: Value*. Ogni specifica del protocollo HTTP definisce gli insiemi di Header e Value permessi, e quali sono i loro significati.

Esempio.

- *User Agent*: Indica su quale piattaforma, dispositivo e altre informazioni sta il web browser
- *Accept Language*: Indica la preferenza della lingua

Osserviamo che quindi il *web server* costruisce il documento *anche* in base agli header in richiesta!

Adesso vediamo le risposte HTTP:

```
HTTP 1.1 <status_code> <reason_phrase>
<headers>
<linea vuota>
<documento richiesto (se c'è)>
```

La *Status Code*, accompagnata dalla *reason phrase*, indica l'esito della richiesta e sono suddivisi in 4 tipologie:

- **200**: Esito positivo, OK
- **3XX**: Redirection ad un altro sito, ovvero ha trovato il documento da un'altra parte
- **4XX**: Client error, ad esempio il client non è autorizzato a vedere certi contenuti
- **5XX**: Server error, problematiche col web server

Come si hanno gli *request header*, abbiamo gli *response header* e funzionano allo stesso modo; servono per fornire informazioni aggiuntive. Essi sono:

- Content-type
- Content-length
- Content-encoding

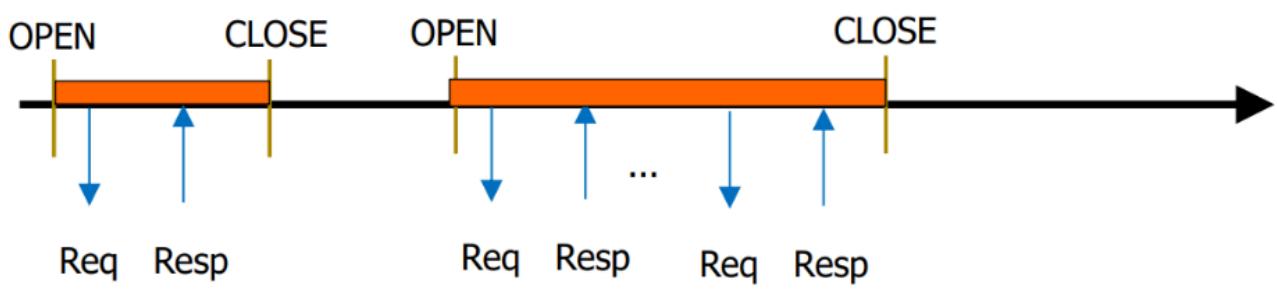
Osservazione. Nella response *non* si ha il nome del documento!

 X

3. HTTP e TCP

Facciamo un breve detour su *come* viaggiano le trasmissioni HTTP su TCP:

- Ogni connessione TCP invia *una o più* coppie di HTTP request-response
 - In realtà la situazione è più complicata, siccome possono verificare altre situazioni (request pipelining, HTTP2, QUIC, ...); non li vedremo
- Si chiude quando lo decide il *Browser* o il *Server*, a loro discrezione. Se lo decide il server, il browser viene *"avvisato"* nell'ultima response con l'header *Connection*



4. Header HTTP

Adesso approfondiamo degli header da sapere

4.1. Header Host

Request Header

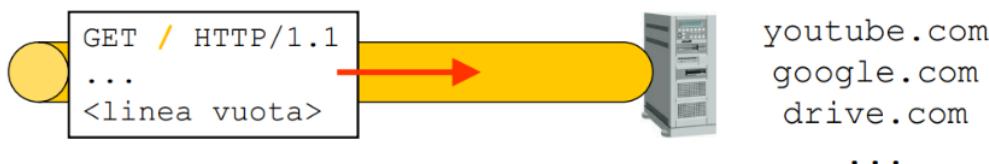
Header: Host

Value: Web Server Name (quindi la seconda parte dell'URL!)

Questo header serve per evitare alcuni casi ambigui, come i casi in cui *un web server* ha più *name server* (come ad esempio Google, Youtube e Drive); quindi per risolvere questa sorta di disambiguità, si specifica il *host name* nell'header.

Altri esempi di casistiche simili sono *web hosting* e *proxy* (vedremo i proxy dopo).

youtube.com.	299	IN	A	172.217.9.78
google.com.	299	IN	A	172.217.9.78
drive.google.com.	299	IN	A	172.217.9.78
calendar.google.com.	86399	IN	CNAME	www3.l.google.com.
www3.l.google.com.	299	IN	A	172.217.9.78



4.2. Referer

Request Header

Header: Referer

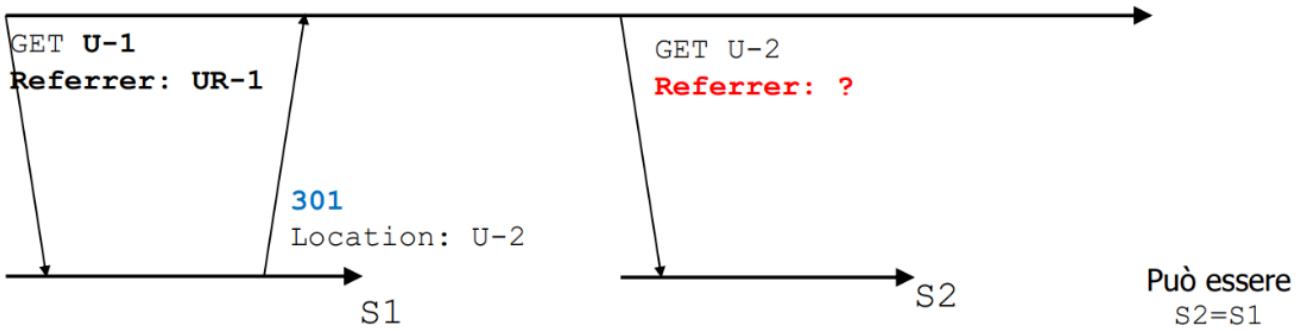
Value: URL

Il *referer* identifica il documento in cui è stato trovato l'URL richiesto, diciamo quindi l'URL che "*mi ha spinto a cercare*" il link.

(*approfondimento personale, non fidarsi!*) Questo è uno degli header più controversi; viene visto come un compromesso tra privacy e utilità fatto male.

Osservazione. Nel caso delle HTTP redirection, il referer può cambiare a seconda del browser (quindi NON specificato nel protocollo HTML):

- Dare l'URL originale
- Dare l'URL che ha "*detto*" di effettuare la redirection
- Non dare nulla, omettere quindi il referer



4.3. Location

Response Header

Header: Location

Value: URL

Questo header identifica il *"vero posto"* del documento. Questo viene utilizzato nei casi di **HTTP redirection**, infatti se il browser riceve una response con *location header* allora essa invierà automaticamente un'altra request verso l'URL specificato.

4.4. User-Agent

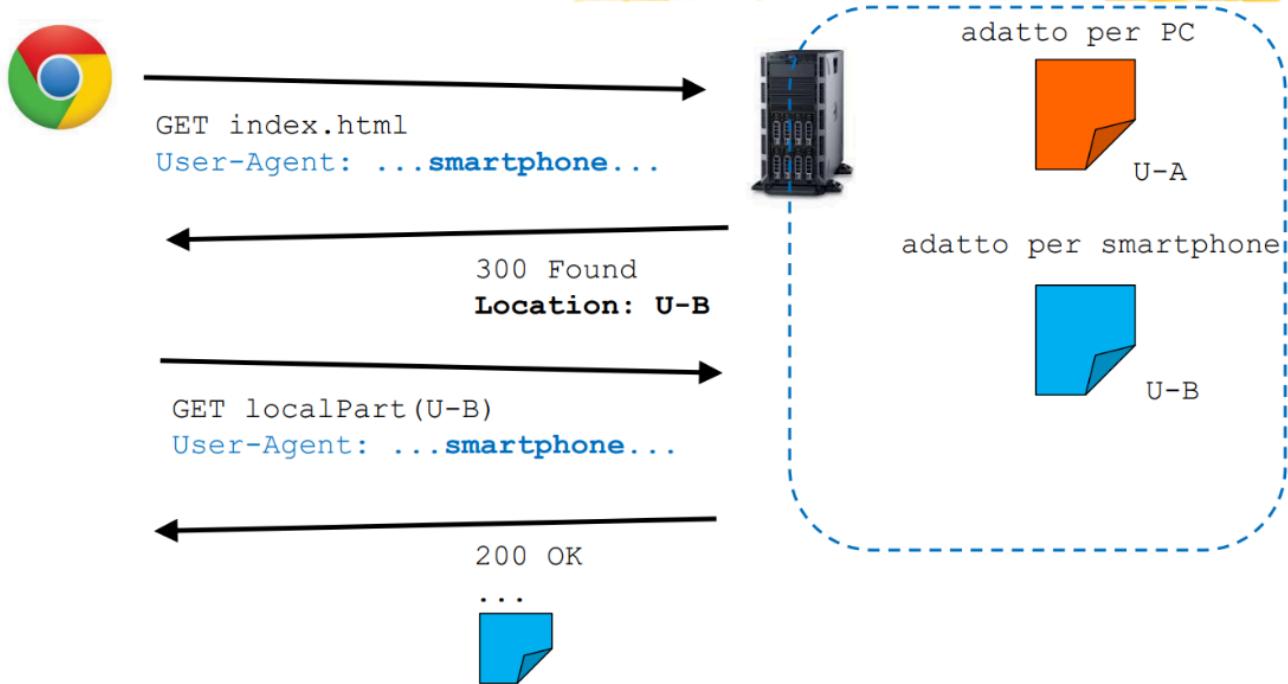
Request Header

Header: User-Agent

Value: String (formattato con la convenzione specificata)

Questo header descrive il *browser usato* e *quale dispositivo* si sta usando per effettuare la richiesta. Il web server può:

- Ignorarlo
- Rispondere con un *"documento ottimizzato"* per quel particolare browser. Ci sono due modi per farlo:
 - Mediante una *redirection*, quindi rispondere con 3XX; approccio abbastanza diffuso fino a qualche anno fa
 - Designare una *responsive web site* (quindi dinamico!), quindi di scrivere dei script CSS/Javascript opportuni per *"adattare"* la pagina allo schermo in cui si trova. Ciò non crea traffico aggiuntivo, ed è l'approccio attuale



4.5. Altri Response Header

Response Header

Header: Content-type

Value: Stringa (un formato o un insieme di formati, tipo .jpeg, .html, eccetera...)

Osservazione: Vedremo che può essere anche una Request header, vedremo in dettagli dopo...

Response Header

Header: Content-length

Value: Intero (scritto sotto forma caratteri ASCII)

Proxy HTTP

X

Proxy HTTP: definizione, dinamica reale. Motivazioni per i proxy, configurazione delle proxy.

X

0. Voci correlate

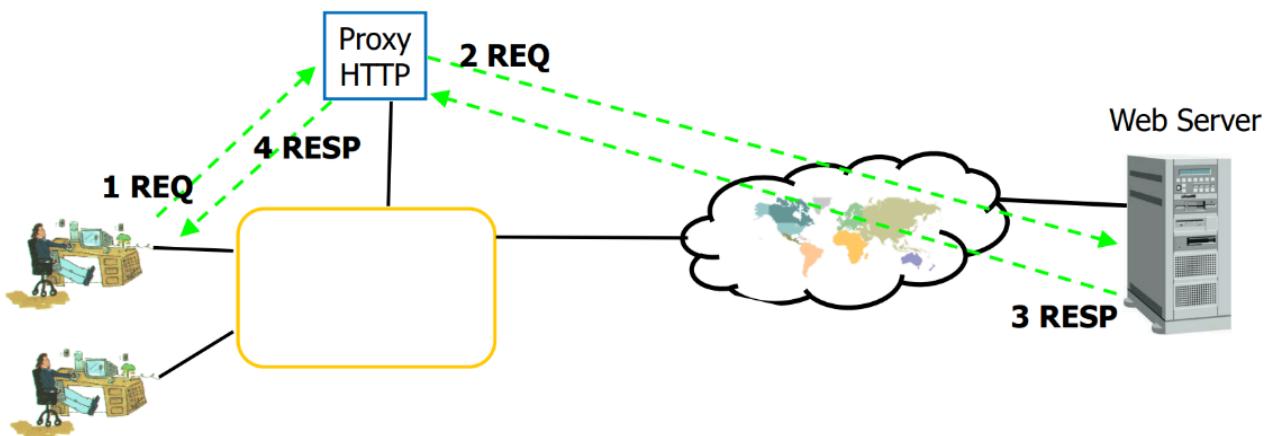
- Name Server
- Nozioni sul World Wide Web

1. Proxy HTTP

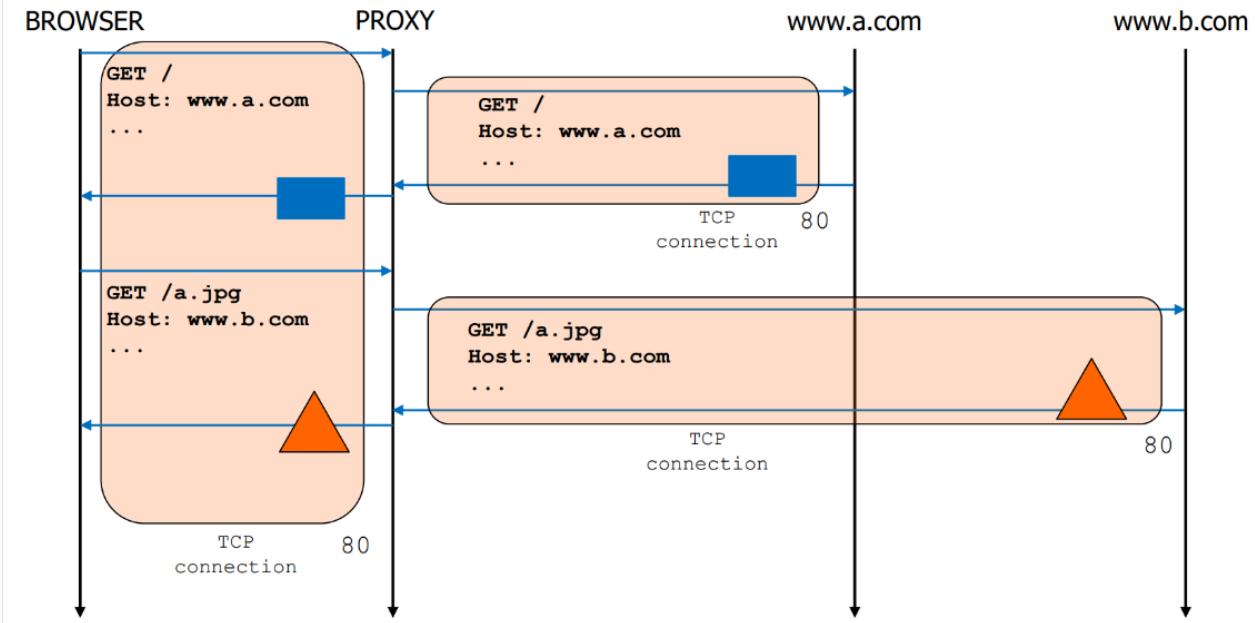
Ricordiamo che il *web server* effettua le seguenti azioni, dato un URL:

- Trova l'indirizzo IP associato alla seconda parte dell'URL (il server name)
- Si connette e comunica col web server
- ...

In realtà, molte organizzazioni non permettono il *collegamento diretto* tra *web server* e *client*. C'è un *processo intermediario* che funge da *server* per il client, e da *client* per il web server. Questo processo si chiama *proxy*.



Quindi il vero procedimento dovrebbe iniziare con la *risoluzione* del *proxy name*. Notiamo che quindi il browser non andrà mai a cercare il valore associato al name server!



Q. Perché?

Ci sono vari motivi di carattere tecnico. Ad esempio, uno dei ruoli del proxy è quello di monitorare gli URL visitati e i suoi contenuti, ed eventualmente proibirne alcuni per motivi di sicurezza. Notiamo che è indispensabile che si usi i *proxy HTTP!*

Q. Devo fare qualcosa se il web browser deve usare il proxy? Come ci assicuriamo che il browser vada a usare il proxy e non "*faccia il furbo*"?

Prima di tutto, sul *router di frontiera* blocchiamo tutti i traffici di porta 80/443 da client *non proxy*, quindi costringendo ai web browser di usare i proxy. Per quanto riguarda il web browser, o si configura direttamente il browser, o si configura il sistema operativo. Nella maggior parte dei casi, oggi la si configura in automatico.

Q. Come fa il proxy capire *a quale web server* collegarsi?

Lo si capisce dall'header host.

Invio Dati a Web Server

X

Inviare dati a web server. Procedura generale con form HTML: sintassi della tag **<form>** e usare query string. Metodo POST per inviare dati privati, sintassi. Differenza tra request GET e POST.

X

0. Voci correlate

- Pagina e Sito Web
- Client-side Scripting
- Documenti Statici e Dinamici
- Protocollo HTTP

1. Inviare Dati a Web Server

Fin'ora abbiamo *prelevato risorse* da web server. Come si effettua il viceversa, ossia *inviare informazioni* al web server?

Esempi di casi in cui è necessario l'uso:

- Verbalizzazione esami
- Email
- Acquisti
- Eccetera...

Il procedimento *generale* è come segue:

1. Inserire i *dati* su un "*campo*", specificato mediante un protocollo di invio dati (esempi: FORM, BASIC, vedremo dopo...)
2. Costruire una stringa contenente i dati
3. Inviare una HTTP request con la stringa

Per la 1., vediamo il tag FORM:

1.1. Tag Form

Tag: `<form ...> ... </form>`

Attributes:

- *Action*: URL (inserire l'URL a cui si intende di inviare i dati)
- *Method*: ??? (Vedremo dopo cosa mettere)

All'interno dello spazio tag `form`, invio più campi in cui definisco i metodi con cui si inviano i dati. Elenchiamo alcuni esempi:

- `<input type="..." id="..." name="..." value="...">`
- `<textarea id="..." name="...> ... </textarea>`

X

2. Metodi di Invio Dati

2.1. Query String

Un primo metodo semplice per *inviare dati* è quello di usare le query string. Quindi inviare una request `GET URL?S` dove `S` codifica gli input specificati.

2.2. Metodo POST

Il metodo delle query string non potrebbe essere un'ottima idea, siccome potrebbero contenere dei *dati sensibili* e le *browsing histories* sono memorizzati in più "posti"

Un altro modo è quello di inviare una richiesta di tipo `POST`, ovvero la richiesta sarà formata come il seguente:

```
POST url_name HTTP/1.1
  content_type: application/form
  content_length: ...
EMPTY LINE
Data
```

Notiamo che è sempre una *HTTP request*, ma di "*tipo diverso*". Vediamo delle differenze tra tipo GET e POST:

GET:

- Non trasporta documenti
- Caso comune
- Si aspetta delle risposte, tipicamente un documento dinamico

POST:

- Trasporta dei dati, quindi si DEVE specificare gli header `content_type` e `content_length`
- Invio informazioni raccolte nel HTML FORM

Sessioni HTTP

X

*Sessioni HTTP. Motivazione introduttiva. Definizione di sessione, implementazione con i Cookie.
Stato delle sessioni: tabella delle sessioni, variabili di sessione.*

X

0. Voci correlate

- Documenti Statici e Dinamici

1. Motivazione Introduttiva

Un documento è costruito da un *programma* che tiene conto della *HTTP request* e le precedenti inviate dallo stesso *browser e dispositivo*. Fino ad'ora, sappiamo come gestire la prima parte; e la seconda parte? Ovvero, come facciamo a realizzare un sistema che sia in grado di "*ricordare*" le request inviate precedentemente?

Vediamo dei primi approcci (che non avranno successo):

- Agganciare tutte le HTTP request provenienti dalla *stessa connessione TCP*: non funziona, siccome le connessioni TCP possono essere aperte e chiuse arbitrariamente
- Come prima ma le HTTP request provengono dallo *stesso indirizzo IP*: comunque posso usare dispositivi (e browser) diversi...

Quindi, bisognerà "*inventare*" un nuovo modo per fare tutto ciò, siccome il protocollo HTTP non è sufficiente per farlo; infatti, si dice che HTTP è un protocollo *stateless*, ovvero le request HTTP non conservano nessun riferimento alle richieste precedenti fatte dallo stesso browser e device.

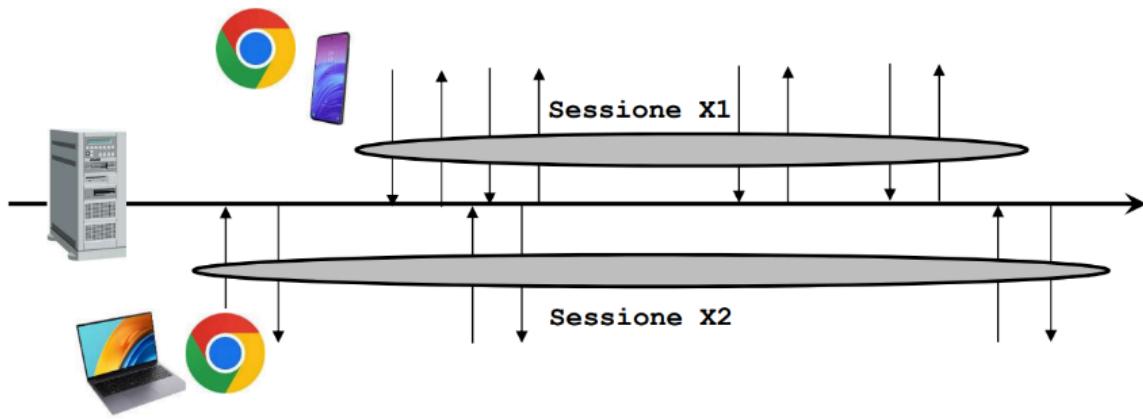
X

2. Sessione HTTP e Cookies

Definiamo il concetto della *sessione HTTP*:

DEFINIZIONE. (*Sessione*)

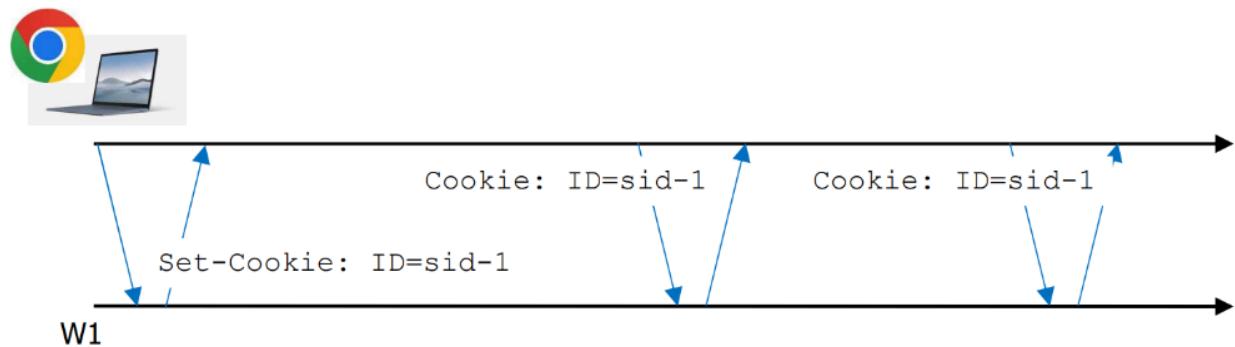
Una *sessione* è una sequenza di Request e Response HTTP scambiate da una coppia di *Browser@Device* e *Website*. Ogni sessione ha *identificatore univoco* sul Website.



In particolare, con l'implementazione mediante i **cookie** abbiamo che una HTTP request R appartiene ad una certa sessione **sid** con web server W *se e solo se* R contiene header **Cookie: name=sid**. Name e sid dipendono dal web server, sia dalla sua configurazione e dalla sua scelta fatta al momento.

Vediamo come si fa l'assegnazione di un cookie ad un **web browser**:

1. Web browser invia HTTP request senza header **Cookie**
2. Web server riceve la HTTP request, *crea* sessione identificata da **sid_1** e la registra nella sua **tabella delle sessioni**
3. Web server invia una HTTP response con header **Set-Cookie: ID=sid_1**
4. Web browser riceve la HTTP request, e salva il cookie nella propria **Cookie table**; nelle prossime HTTP request verso lo stesso host il WB inserisce il header **Cookie: ID=sid_1**



Approfondiamo sulle proprietà del **Cookie table**:

- Viene **condivisa tra tutti i tab**
- E' persistente, ossia fin quando non si decide di cancellarla rimane
- Non è condivisa tra browser diversi
- Può essere cancellata in qualunque momento

Q. Quanto possono durare i cookie?

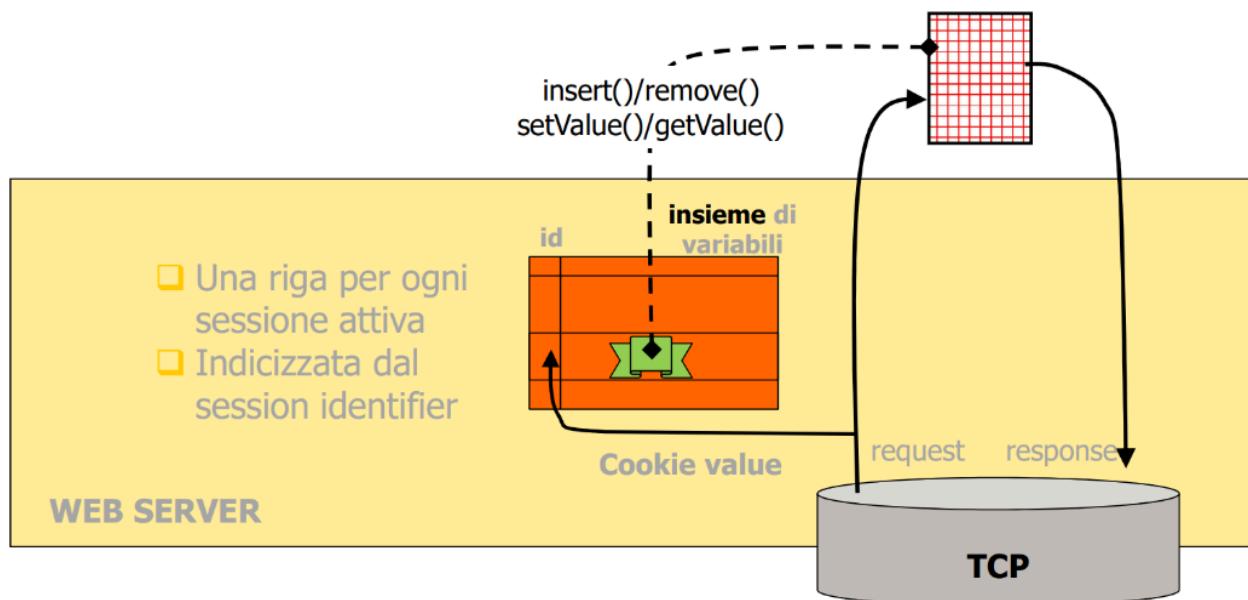
Può essere specificato dal web server, nel header **Set-Cookie: name=sid; Expires=date**

3. Stato delle Sessioni

Uno dei motivi per cui designiamo il concetto delle *sessioni* è proprio quello di associare delle *variabili* ad ogni sessione.

Una *variabile di sessione* è una *string*, che viene salvata in una *session state* gestita dal *web server*. Il web server dispone delle *funzioni* per agire sulla session state:

- *Creazione / Distruzione variabili*: `InsertSession("...", ...)`, `RemoveSession("...")`
- *Lettura / Scrittura variabili*: `SetValueSession("...", ...)`, `GetValueSession("...")`



Osservazione. Notiamo che abbiamo più aspetti delle variabili di sessione gestiti da livelli diversi:

- *Spazio di memoria e interfaccia di accesso*: Web server
- *Significato (interpretazione)*: Web app
- *Definizione header Cookie*: HTTP

X

4. Requisiti Cookie Identifier

Vedremo che ogni sessione può corrispondere ad un *acesso autenticato* ad una pagina web; quindi, se un giorno fosse possibile leggere la tabella dei cookie di qualcuno, le conseguenze sarebbero gravi! L'attacker può generare HTTP request che appartengono alle persone altrui.

Quindi il *cookie identifier* dovrà rispettare ulteriori requisiti, per essere più "*riservata*"

- *Non numerabile*: l'attacker non dev'essere in grado di poter "*indovinare*" la cookie id, sennò potrebbe tentare l'attacco eseguendo un ciclo while banale

- *Non predicibile*: non dev'esserci un pattern riconoscibile nella generazione delle session id

Authentication e Authorization

X

HTTP authentication. Concetti preliminari: assunzione dell'esistenza della sessione, definizione di subject e username, sessione autenticata e non autenticata. Protezione dei contenuti. Caso comune: Log-in con lo stesso URL. Azioni web server, pseudocodice. Concetto di authorization: diritto di accesso degli utenti, differenza dal concetto di authentication. Definizione di ACL (Access Control List), logica dell'authorization check. Generalizzazione degli ACL, definizione di Realm e Realm table. Osservazione: l'authorization va programmato in certi casi.

X

0. Voci correlate

- Sessioni HTTP

1. Concetto di Autenticazione

Supponiamo d'ora in poi l'esistenza della *sessione HTTP*.

Definizioni.

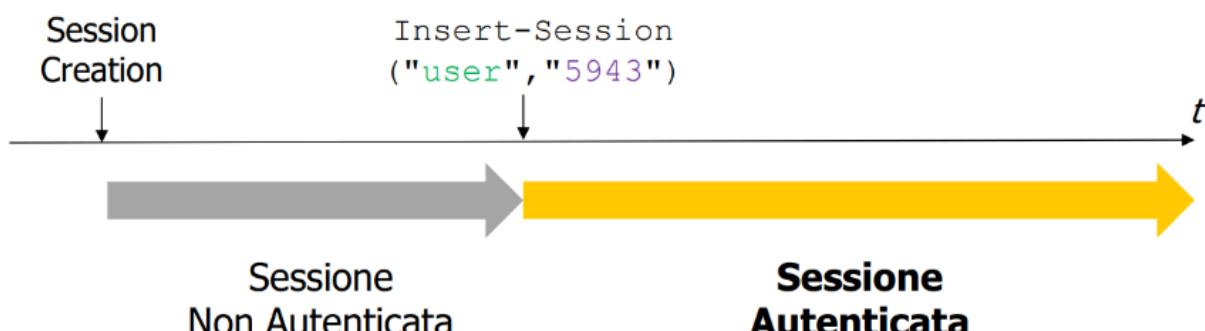
Subject: Utente di un sistema informatico (ESTERNO al calcolatore)

Username: Una stringa che identifica un'utente (INTERNO al calcolatore)

Un *subject* utilizza un *username* solo se è in grado di dimostrare di essere associato a tale username al sistema. Lo si fa mediante il possesso delle *credenziali*, e ci sono più modi per farlo; la più comune è la coppia Username-Password

Quindi ogni server si salva una *tabella delle credenziali*, dove ad ogni username si associa delle credenziali.

Definizione. Una sessione si dice *non autenticata* se non è associata a nessun username, altrimenti si dice *autenticata*. Ogni sessione è inizialmente *non autenticata* e può diventare *autenticata* mediante delle richieste HTTP. Di solito, la variabile di sessione **user** dipende dalla configurazione del web server.



Definizione. Un URL si dice protetto se richiede l'*autenticazione*, altrimenti si dice *non protetto*. Denotiamo gli URL protetti con URL-P e gli URL non protetti con URL-NP.

Per gestire il *log-in*, ci sono più casi:

1.1. Login Stessa Pagina

Analizziamo un caso comune: ovvero, se vogliamo accedere ad un URL-P di un sito web, dobbiamo effettuare il log-in (che si trova nello stesso URL-P)

Esempi: Facebook, Amazon

1. HTTP Request sessione non autenticata: **GET URL-P ...**
2. HTTP response con "*sollecita autenticazione*" (pagina log-in)
3. HTTP request con credenziali
4. HTTP response con documento dinamico personalizzato

1.2. Redirection verso Login Page

1. HTTP Request sessione non autenticata: **GET URL-P ...**
2. HTTP response con *3XX Redirection* verso **URL-LGIN**
3. HTTP request **GET URL-LGIN**
4. HTTP request con credenziali
5. HTTP response con documento dinamico personalizzato

Osservazione. In questo caso, come possiamo "*ricordarci*" a quale URL l'utente stava provando ad autenticarsi? Possiamo sfruttare la *session state* e salvare **URL-P** in una certa variabile della session state (che implicitamente assumiamo sia già presente, quindi il browser ha inviato header set-cookie eccetera...)

1.3. Login Diverso URL

1. HTTP Request sessione non autenticata: **GET URL-P ...**
2. HTTP response con "*sollecita autenticazione*" (pagina log-in)
3. HTTP request con credenziali **POST URL-VALIDATION**
4. HTTP reponse con *3XX Redirection* verso **URL-P1** predefinito
5. HTTP request **GET URL-P1**
6. HTTP response con documento dinamico personalizzato

Osservazione. Nei casi 1.2. e 1.3. che prevedono l'HTTP redirection, nella request **POST** in cui si trasmettono le credenziali come *Referer* si può inserire sia URL-P, URL-LGIN, ..., o anche ometterla siccome non è specificato in RFC

2. Protocolli di Autenticazione

Per mandare le richieste di "*solllecito autenticazione*" e di "*invio credenziali*", ci sono più modi. Vedremo i protocolli FORM e BASIC, le più comunemente usati

2.1. FORM

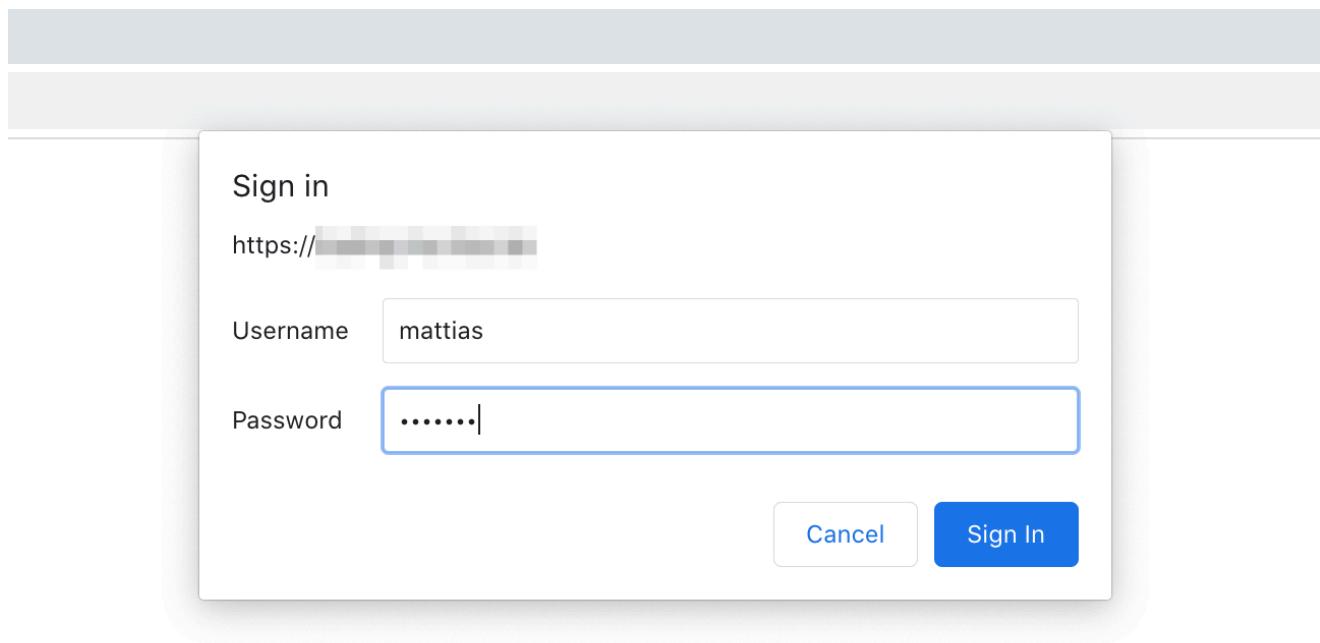
L'idea di base è semplicemente quello di richiedere le credenziali su un form HTML, in un *documento a parte*! (Ovvero non inserisco il FORM nel documento da proteggere, sennò sarebbe inutile...)

- *Solllecito*: 200 OK, inviare il documento in cui si sollecita ad inserire le credenziali
- *Invio Credenziali*: Request POST, con content-type "form" e verso un'URL di validazione

2.2. BASIC

L'idea di base è quella di aprire una *finestrella esterna alla pagina* per chiedere l'username e la password e inviare un'HTTP request specificando i dati in uno degli *header*

- *Solllecito*: 401 Unauthorized (!, faremo un'osservazione dopo), inserisco header **WWW-Authenticate: Basic realm = "<my_realm>"** e altre informazioni necessarie
- *Autorizzazione*: **GET URL_P** e metto header **Authorization: BASIC u:p** dove **u** e **p** sono codificate in BASIC64 (nota! non è una protezione vera e propria, in quanto è una codifica facilmente reversibile)



Notiamo che in nessun modo sono state usate delle funzionalità HTTP per l'autenticazione.

Inoltre, facciamo un'osservazione nel caso di BASIC authentication: quando il web server sollecita l'utente ad autenticarsi, si manda una response di codice 401 "Unauthorized" al client. Il termine *Unauthorized* è più che forvante, siccome in realtà si tratta di autenticazione e non c'entra niente con l'autorizzazione (vedremo dopo). D'altronde anche i termini usati negli header sono forvanti, in quanto si usa "*Authorization*" e "*Authentication*" intercambiabilmente.

3. Logica Web Server

Per fare tutto ciò, il server usa la seguente logica (in pseudocodice):

PYTHON

```
req_url <- extractURL(req)
if req_url in URL_NP:
    doc <- BuildDocument(req_url)
    Respond(doc)

elif req_url in URL_P:
    u <- getValue-Session("user")
    if not u:
        # sessione non autenticata
        user, pwd <- extractData(req)
        if user, pwd in DATA:
            doc <- BuildDocument(req_url, u)
            Respond(doc)

    else:
        Respond("You must login", ...)

else:
    doc <- BuildDocument(req_url, u)
    Respond(doc)
```

4. Authorization

Col concetto di *authentication* abbiamo distinto gli "*URL protetti*" da quelli non protetti. Questo basta? No, in quanto tutti gli utenti autenticati hanno gli stessi "*diritti d'accesso*".

Ad esempio, su Drive alcuni documenti sono visualizzabili *solo* da certi utenti. Come facciamo?

ACL. Prima di tutto, nella *configurazione del web server* definiamo gli *Access Control List*, che sono degli insiemi di utenti autorizzati a visionare la pagina dell'URL. Quindi, alla logica del *web-server* aggiungiamo un controllo aggiuntivo prima di restituire il documento: controllare che l'utente appartenga all'ACL dell'URL.

Tuttavia, è difficile definire individualmente gli ACL per ogni URL. Definiamo la seguente nozione per avere una rappresentazione più compatta degli ACL:

Realm. Un realm è un *insieme di risorse protette nello stesso modo*, formato da:

- *Realm Name*
- *Resources*: Descrizione molto compatta di risorse protette
- *ACL*: Insieme di username autorizzati
- *Protocol*: Vedremo dopo, si intende il protocollo di autenticazione (BASIC, FORM, eccetera...)

Realm Name	Resources	ACL	Protocol
Esami	/esami/*	alberto marco ruud	FORM
Documenti	*.pdf	alberto ricky andry	BASIC
Config	/admin/*	admin	FORM

In ogni realm è presente implicitamente la *Default Realm*, contiene tutti gli altri URL non protetti ed è accessibile senza autorizzazione.

Realm Name	Resources	ACL	Protocol
Esami	/esami/*	alberto marco ruud	FORM
Documenti	*.pdf	alberto ricky andry	BASIC
Config	/admin/*	admin	FORM
Default	Any other	-	No Auth.

Osservazione. Quindi con i *realm* l'authorization può essere un aspetto configurato nel *web server* in una tabella. Questo basta? No, ci sono certi casi in cui l'authorization va anche programmata. Esempio: dato un server web che usa un *programma* che esegue query a DB, bisogna stare attenti che l'utente sia effettivamente autorizzato ad effettuare la query.

Un caso notevole che esemplifica quanto sia necessario *programmare* i realm, è la vulnerabilità del sito web 18app: non è stato aggiunto il controllo di authorization nelle pagine individuali degli utenti, quindi ogni utente autenticato poteva accedere alle pagine (e consumare i voucher) di goni altro utente...

Q. Supponiamo che un browser@device sia autenticato ad un username. Cosa succede se cambio il *realm* durante la sessione? L'utente deve autenticarsi di nuovo?

Solitamente no, tuttavia se il nuovo protocollo di autenticazione è più "*forte*" allora si richiede di riautenticare.

X

5. Recap: Logica Web Server

Per fare un recap, si scrive un *pseudocodice* che descrive la logica del web server per gestire l'authentication e l'authorization (assumendo session).

```
let URL_P
let ACL(URL_1)
let ACL(URL_2)
...
let URL_NP

req <- GetRequest()
req_url <- ExtractUrl(req)

if req_url in URL_NP:
    doc = BuildDocument(req_url)
    SendResponse(..., doc)

elif req_url in URL_P:
    cookie <- req[cookie]
    if not cookie:
        # Creare sessione
        cookie = GenerateCookie()
        States[cookie] = {}
        SendResponse(..., header: {'set_cookie': f'id={cookie}'})

    elif cookie:
        usr = GetValueSession('user')
        if usr:
            # Se autenticato
            if usr in ACL(req_url):
                # Se autorizzato
                doc = BuildDocument(req_url, usr, ...)
                SendResponse(..., doc)
            elif usr not in ACL(req_url)
                SendResponse("Not Allowed", ...)

        elif not usr:
            # Se non autenticato
            if req['credentials']:
                # Se manda le credenziali
                u, p = req['credentials'].extract()
                if (u,p) in CREDENTIALS:
                    BuildDocument(u, doc, ...)
                    SendResponse(..., doc)

            elif not req['credentials']:
                SendResponse("Devi autenticarti", ...)
```


Modalità Incognito

X

Un paio di parole sulla navigazione in incognito. Definizione di modalità "incognito" di un browser.
Osservazioni pratiche.

X

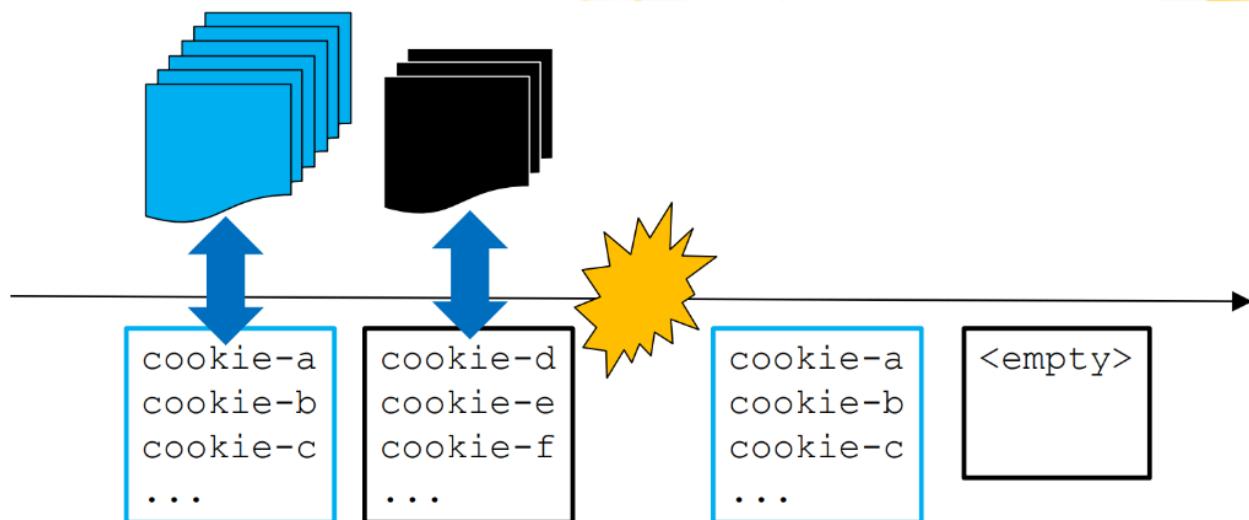
0. Voci correlate

- Sessioni HTTP

1. Modalità Incognito

Ricordiamo che una delle proprietà del *cookie table* è il fatto che sia *persistente* e *condivisa* tra i tab.

Definizione. La "*modalità incognito*" di un browser consiste in creare un'altra *cookie table separata* che non sia persistente. In altri termini, si tratta di un browser separato "*usa e getta*".



Esempi.

- Il browser naviga su un certo sito web www.somesite.it e la si accede come un certo utente U. Aprendo un altro tab e navigando sullo stesso sito, si rimane ad essere utente U; tuttavia, aprendo un tab incognito la si accede senza autenticazione
- Vale anche il viceversa, solo che la cookie table del browser incognito non persiste (rimane comunque tra i tab)

Q. Questo mi aiuta a navigare "*senza tracce?*"

Naturalmente no, siccome comunque ho molti fattori che "*lasciano tracce*":

- Comunicazioni DNS

- Associazioni a username
- ...

Quindi il termine *incognito* è un termine fuorviante, in quanto ci suggerisce l'idea di navigare anonimamente, anche se in realtà non stiamo facendo altro che usare un browser "*usa e getta separato*".

Per risolvere veramente il problema, si usano altre tecnologie (di cui non vedremo), come VPN o TOR browser. Inoltre, sarebbe definire *da chi* vogliamo nascondere le nostre tracce.

Web Analytics e Tracking

X

Aspetto pratico del web: web analytics e tracking. Problema di web analytics: contare accessi a siti, per pagina. Metodi per implementare web analytics. Tracking: cookie di terze parti, implicazioni..

X

0. Voci correlate

- Nozioni sul World Wide Web
- Pagina e Sito Web
- Sessioni HTTP

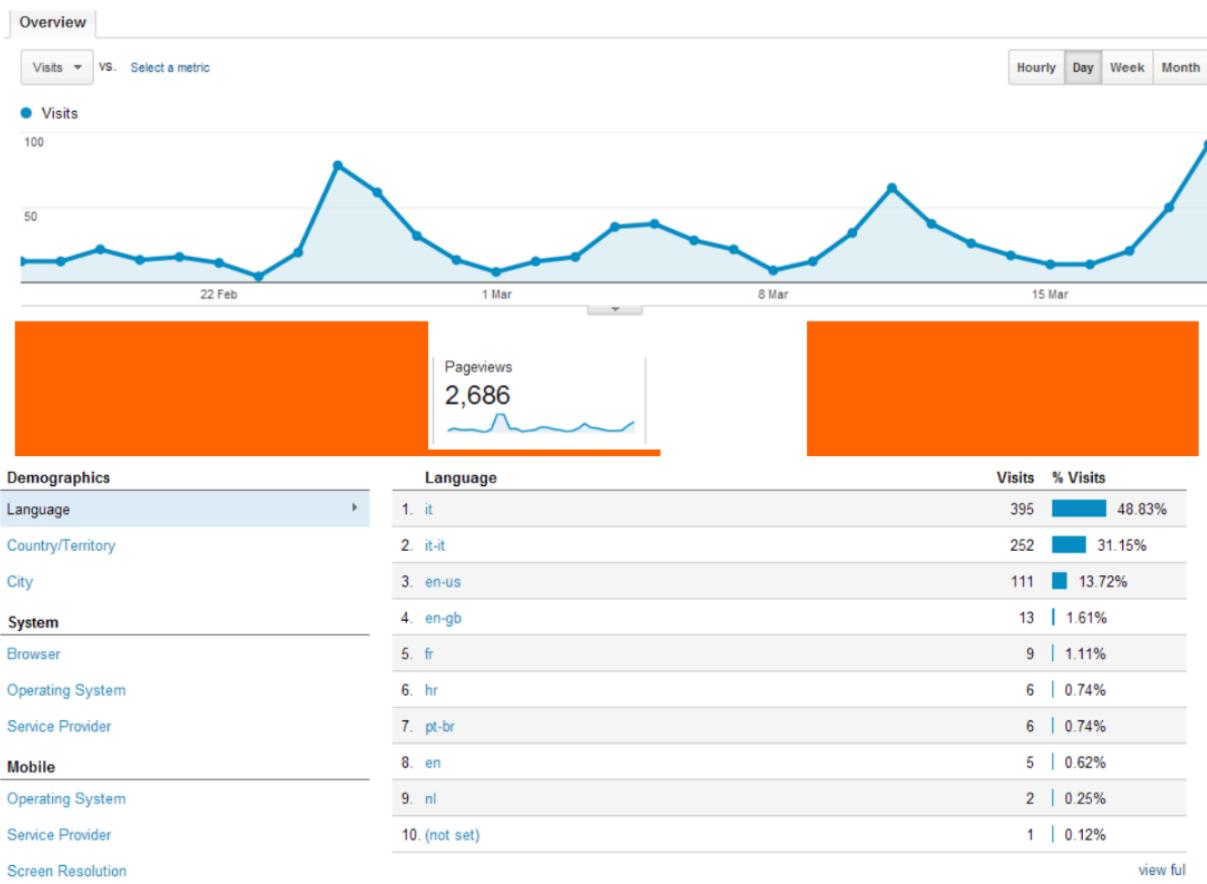
1. Web Analytics

Avendo tutti gli strumenti sul web, vediamo un paio di applicazioni.

PROBLEMA. Supponiamo di essere il proprietario di un sito. Come possiamo contare gli *accessi* al sito, dividendoli per *pagina*?

Il *conteggio* non sarà realizzato da noi, in quanto delegheremo questo lavoro ad un'altro web site A, noto come *servizio di web analytics* che è un *servizio specializzato* che facilita l'analisi dei dati.

Esempio. Un esempio noto di servizio analytics è *Google Analytics*. Non solo è presente delle informazioni sulle visite, ma anche gli *header delle HTTP request*, gli *indirizzi IP lato client*, eccetera...

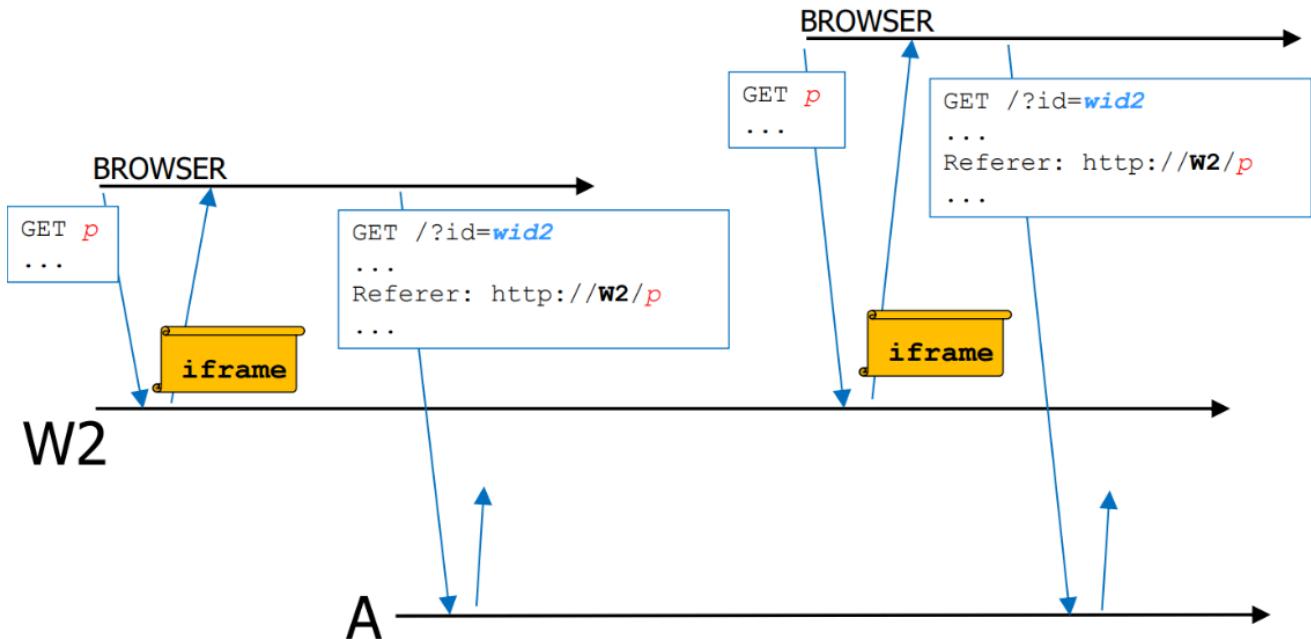


In pratica, per "*delegare*" questo lavoro al servizio A, ci sono tanti modi per farlo. L'unico requisito dev'essere che il *caricamento di una pagina P* implica automaticamente l'invio di una HTTP request ad A.

Esempio. A assegna un codice univoco (wid) ad ogni sito, e poi inseriamo un *iframe* "*invisibile*" (microscopicamente piccolo) in ogni pagina: `<iframe src="http://A/?id=wid" height=1 width=1>`

Notiamo che in questo esempio:

- Il codice WID è ridondante, siccome nel referer specificheremo l'URL "*sorgente*". Tuttavia, è più utile nel senso pratico per semplificare le implementazioni su database (in quanto le WID formeranno una chiave primaria eccetera)
- A conterà *solo le visite*, e non è in grado di identificare l'utente siccome non ha delle informazioni sufficienti. Con *tracking* vedremo che sarà possibile farlo lo stesso (con grossissime implicazioni...)



Esempi. Altri esempi:

- Stesso di prima ma con "immagini trasparenti o molto piccole"
- Script JS
- Sistemi di sviluppo leb, vengono inserite automaticamente
-

Osservazione. Ogni pagina web W può anche usare più servizi di web analytics A_1, \dots, A_n .

X

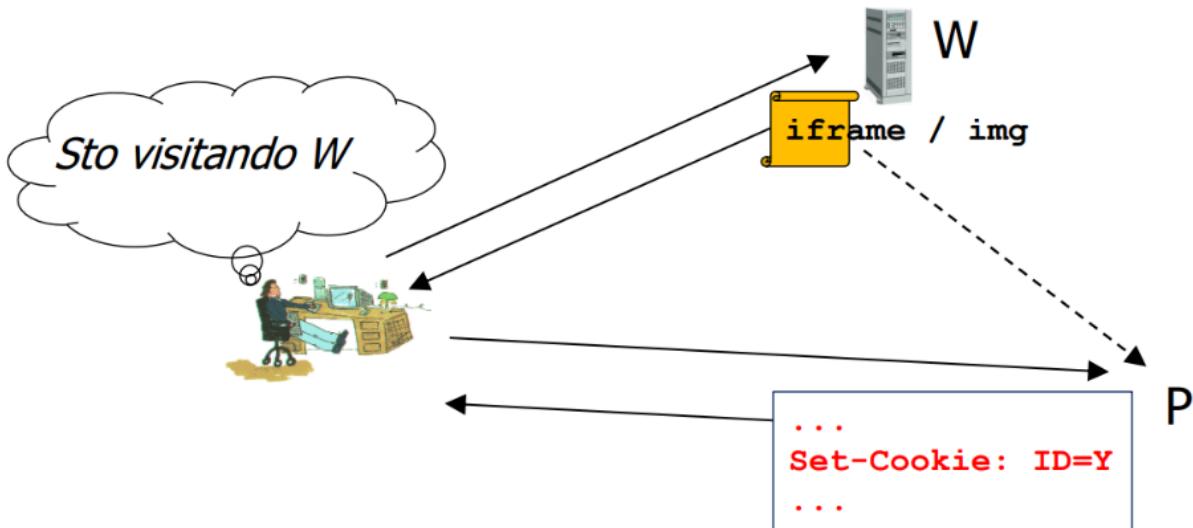
2. Third-Party Cookies

Andiamo ad *aumentare* l'obiettivo di prima:

PROBLEMA. Raccogliere *sequenze di pagine visitate da ogni browser* e sapere il numero di clienti "abituale"

La soluzione a questo problema consiste nell'usare i *cookies* per gli accessi ad A , con una scadenza "*quasi infinita*". Notiamo che con questa soluzione saremo anche in grado di *contare i browser!*

THIRD-PARTY COOKIES. Notiamo che l'utente *crede di visitare* il sito web W , ma in realtà anche P e P può creare un cookie da assegnare all'utente; questo cookie si dice *cookie di terze parti*. Questo scenario è presente anche in contesti più generali, oltre al web analytics.



Motivazioni. Le motivazioni per usare i *third-party cookies* sono le seguenti:

- *Web Analytics*: come visto prima
- *Tracking*: in questo caso P non aiuterà W a fornire dei dati, invece W permette a P di raccogliere i dati (in cambio di denaro) da raccogliere come suoi, e li venderà ad altri organizzazioni che vogliono creare *profili* di Browser@Device

Il secondo caso è *enormemente diffuso*, ogni W ha tipicamente molti P.

Implicazione. Notiamo subito l'implicazione dei *third-party cookies*: se un sito P è *"presente su molti siti web W"*, allora potrebbe in grado di riconoscere *"ogni"* Browser@Device. Ciò vuol dire quindi P sarà in grado di osservare la navigazione di *"ogni"* Browser@Device....

Per mitigare questa situazione, l'*informativa GDPR* obbliga ai siti W europei di *informare di ogni third-party cookie* a cui forzano gli utenti. In particolare, vengono specificate che le finalità devono essere *"chiare"* e il consenso deve essere *esplicito e consapevole*.

Web Advertising

X

Web advertising (cenni di cenni): idea di base del funzionamento, ruolo di Ad Network. Modi di fare web advertising: contextual advertising e targeted advertising. Implicazione di targeted advertising.

X

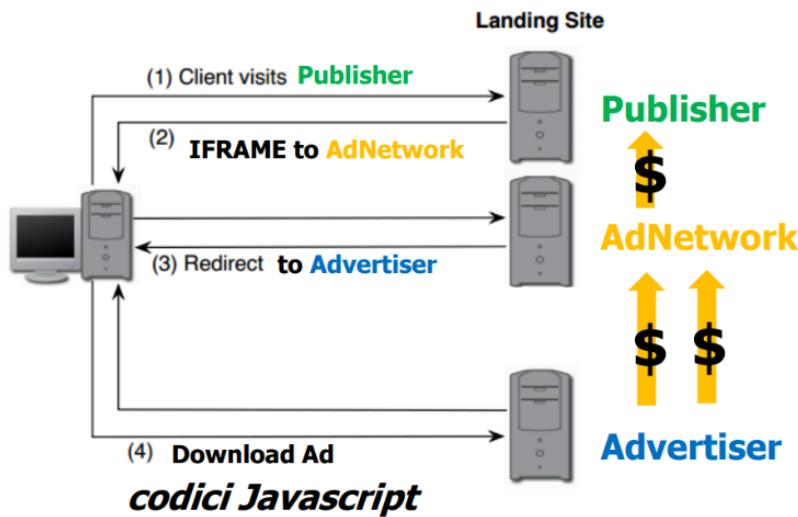
0. Voci correlate

- Web Analytics e Tracking
- Pagina e Sito Web

1. Web Advertising

Il *web advertising* consiste nello seguente schema:

1. Il client visita il sito web "*publisher*" P, che fornisce i contenuti e uno *spazio pubblicitario*. P si dice anche "*Landing site*"
2. Lo spazio pubblicitario non è altro che un tag *iframe* che andrà a prelevare la risorsa dal web server "*Ad Network*" N. L'*ad network* non è altro che l'intermediario tra gli *advertiser* e le *landing site*
3. Contattando N, si viene reindirizzati all'*advertiser* che in risposta invia la pubblicità effettiva, mediante dei codici JS



Il motivo per cui si fa questo schema sono *soldi*, in particolare si paga l'intermediario N per *click* e *per view*. In particolare, i click sono più "*profittabili*" di un'ordine di grandezza rispetto alle view; ciò incentiva gli advertiser A a creare *pubblicità interessanti*.

Osservazione. In questo schema, Ad Network fa delle scelte "*on the spot*" per decidere verso quale *advertiser* indirizzare. Ciò si basa su *algoritmi e infrastrutture complicate* che rispettino i

seguenti requisiti:

- Veloce e scalabile
- Ottimo (massimizza la "*probabilità*" di ottenere click)
- Mostrare tutti gli advertiser

Quindi ci saranno due modi di *scegliere* l'advertiser:

1. *Contextual*: Tengo conto solo del sito corrente come contesto, ossia scelgo l'advertiser più "*adatta*" alla pagina in cui viene visualizzata. L'algoritmo si basa quindi sull'analisi del contenuto di P e delle pubblicità
2. *Targeted*: Qui invece ci basiamo sul *profilo dell'utente* a cui si mostra la pubblicità. Ciò porta con se delle *implicazioni economiche, sociali, politiche e strategiche enormi*, dal momento in cui si cercherà di raccogliere più informazioni sul *Device owner* possibile per avere un targeting ottimale.

Concludiamo dicendo che questo è il *vero costo* dei "*servizi apparentemente gratuiti*" sul web: al posto di pagare con i soldi, paghiamo con i nostri dati che vengono raccolti.

"Mail"

Architettura Email

X

Architettura e-mail. Terminologia: definizione di mail, e-mail. Architettura dell'infrastruttura mail: definizione di Mail User Agent, Mail Transfer Agent, dominio Mail Server. Connessioni e protocolli (cenni): tipologie usate. Scenario comune di sistemi mail. Esempio completo: sender to receiver.

X

1. Significato di "e-Mail"

Nel gergo tecnico, il termine *e-mail* (o semplicemente *mail*) può assumere più accezioni:

- L'indirizzo mail, come ad esempio **john@gmail.com**
- Il messaggio contenuto nella mail
- L'infrastruttura mail di cui vedremo

Per noi, il termine *mail* andrà a significare il *messaggio stesso*, ovvero una sequenza di linee di testo come specificato da un RFC. Per ora, sappiamo che nell'email si ha gli indirizzi mail del mittente e del destinatario.

X

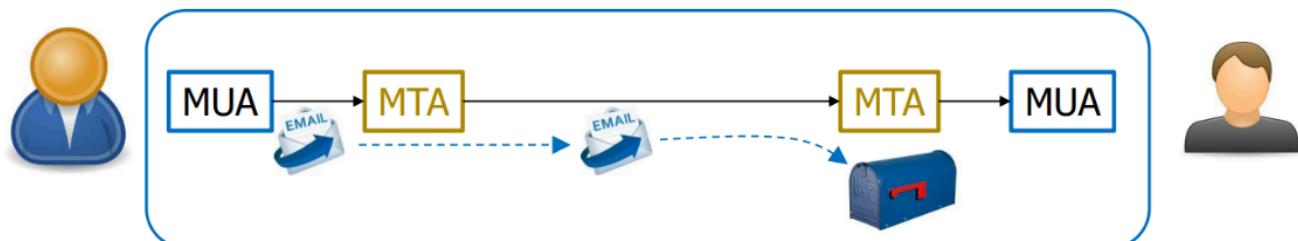
2. Architettura Mail

L'architettura mail è composta da due sottosistemi, e sono le seguenti:

DEFINIZIONE. Le *mail user agent* (MUA) sono i programmi utilizzati dagli utenti per gestire la propria email; quindi invio, ricezione, scrittura, archiviazione ed eccetera...

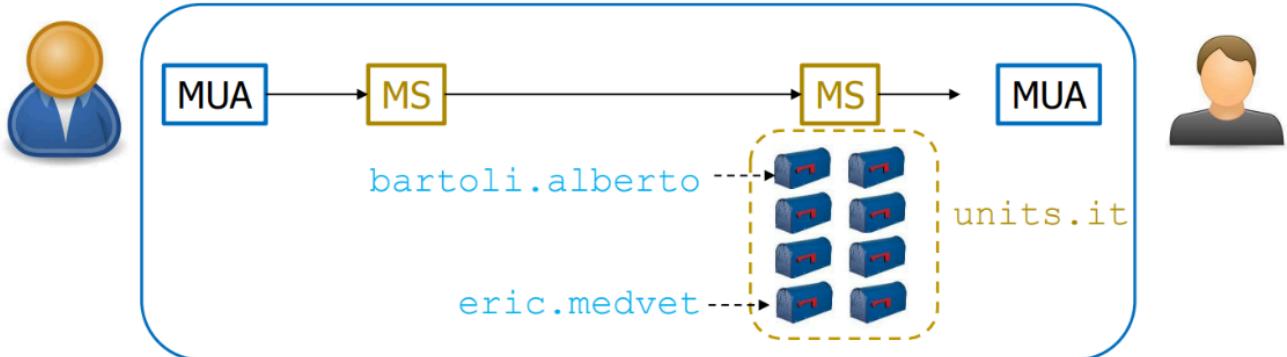
Una MUA può essere quindi un *programma installato sul computer* o una *web app* (web mail).

DEFINIZIONE. Le *mail transfer agent* sono invece dei *programmi server* che tipicamente forniscono due servizi: il trasporto delle mail e la memorizzazione degli email ricevuti (*mailbox*). Quindi, in approssimazione possiamo dire che sono delle *mail server* (MS).



Quindi una MUA interagisce col "suo" MTS (circa MS) per inviare delle mail.

Proprietà Mailbox. Ogni *mailbox* è identificata (e anche localizzata) *univocamente* dal suo *indirizzo email*. La sua sintassi è intuitivamente formata da due parti, divisa dal simbolo chiocciola @. La prima parte è l'identificatore della mailbox, la seconda è il *dominio e-mail*; un *dominio e-mail* è un insieme di mailbox e si trovano tutti sullo stesso MS.



Q. Dato un dominio e-mail, come possiamo rintracciare il suo (o uno dei suoi) mail server?

Semplice, basta interagire col DNS effettuando richieste su RR di tipo MX. Osserviamo che quindi le *mailbox* non centrano niente con gli RR!

Osservazione. Notiamo che tutte le mailbox di un dominio si devono trovare sullo stesso MS, ma è possibile che un dominio e-mail abbia più mail server. Esempio: vedere con la propria posta elettronica istituzionale, usando *Dig Online* per consultare gli RR

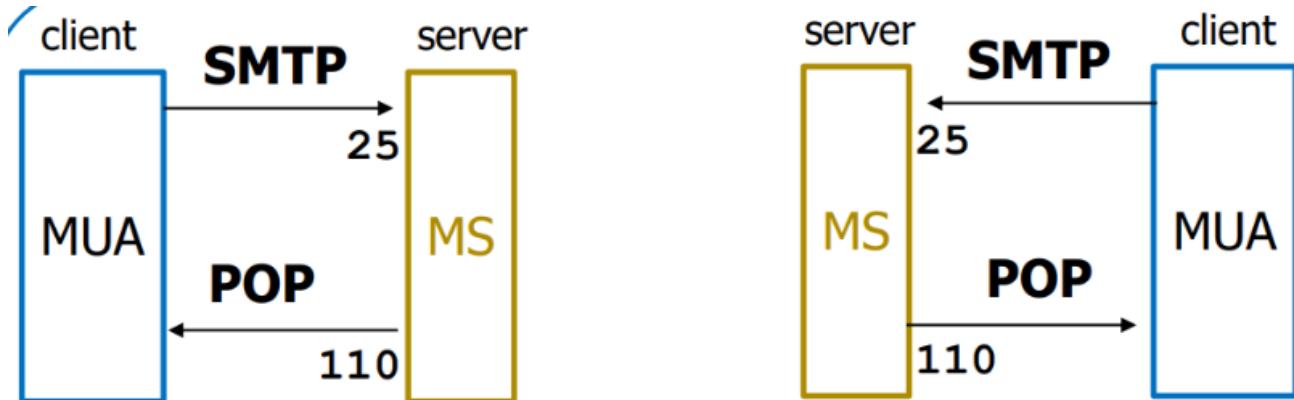
Osservazione. Dominio e-mail e mail server possono avere nomi completamente diversi tra loro.

X

3. Tipologie Connessioni

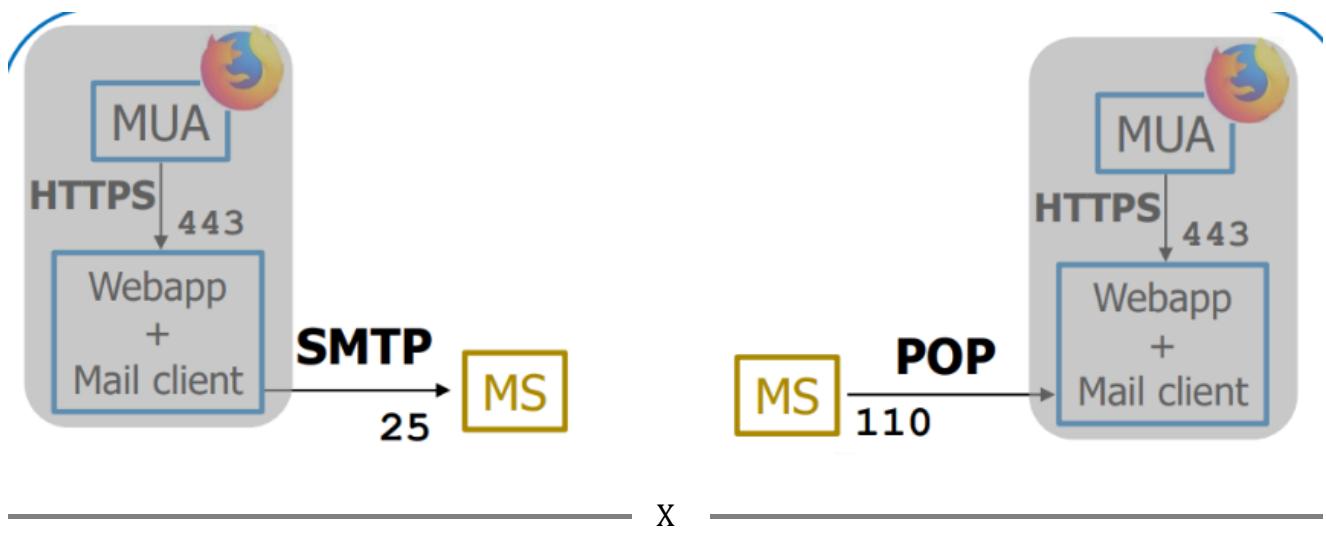
Accenniamo su come avviene la *comunicazione* in termini di *connessioni TCP e protocolli*.

1. *MS to MS per inviare le mail*: Si usa il protocollo *SMTP*, porta 25 lato server. Qui non c'è *mai* il bisogno di autenticarsi, altrimenti sarebbe da distribuire una tabella delle credenziali in *tutti i mail server* che sarebbe assurdamente impossibile da implementare
2. *MUA to MS (invio mail)*: Per inviare le mail, si usa il protocollo *SMTP* con porta 25 lato MS. Qui l'autenticazione è "*opzionale*", nel senso che come descritti nelle RFC non c'è bisogno di autenticarsi, ma praticamente oggi è quasi sempre necessario autenticarsi. Questo aspetto è problematico e si riconduce al caso dell'*e-mail spoofing* (vedremo dopo cosa vuol dire)
3. *MUA to MS (estrazione mail)*: Per consultare le mail ricevute, si usa il protocollo *POP* con porta 110 lato MS. Qui è sempre necessaria l'*autenticazione*.



Osservazione. Nel caso 2. notiamo che la MS non fa mai niente di propria spontanea volontà; è sempre la MUA a contattare la MS, anche nel caso in cui riceviamo mail.

Osservazione. Nel caso 2., se abbiamo che la MUA è implementata come *web app*, allora essa non ha connessione diretta con MS. Invece essa contatterà la *web app* e la *mail client* mediante protocollo HTTP, che a sua volta contatterà la MS.

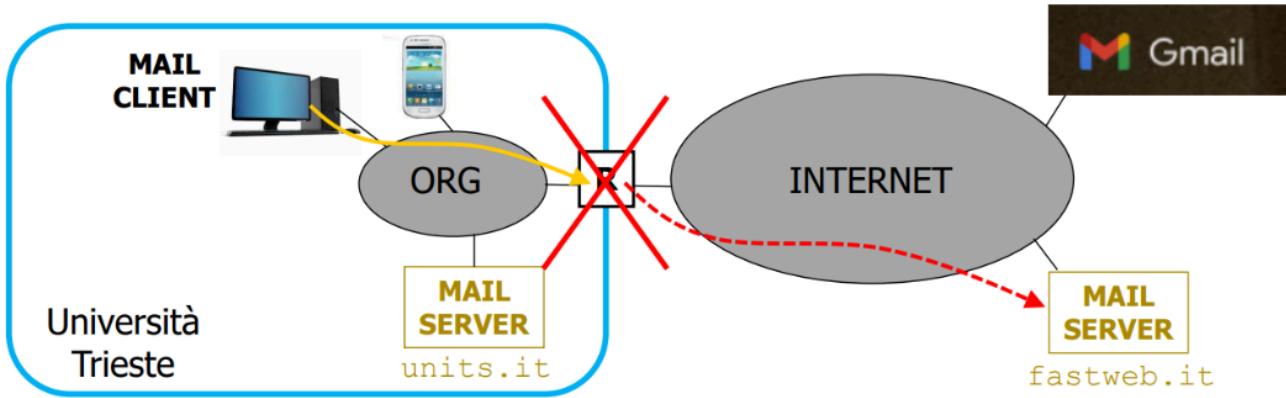


3. Configurazione Sistema Mail

Un scenario *comune* delle organizzazioni è quello di configurare il traffico nella seguente maniera:

- Non permettere traffico e-mail diversi da "*quello proprio*", ovvero il router di frontiera R bloccherà qualsiasi connessione di tipo SMTP verso un mail server esterno che non provenga dal proprio mail server interno.

Lo si fa per motivi pratici, per controllare "*cosa entra e cosa esce*".



Notiamo che in questo caso è possibile comunque usare Gmail o un'altra web mail, siccome la MUA si connette al mail client esterno mediante HTTP/HTTPs.

4. Sender to Receiver

Vediamo l'esempio completo: come *"viaggia"* un'email, quando viene inviato?

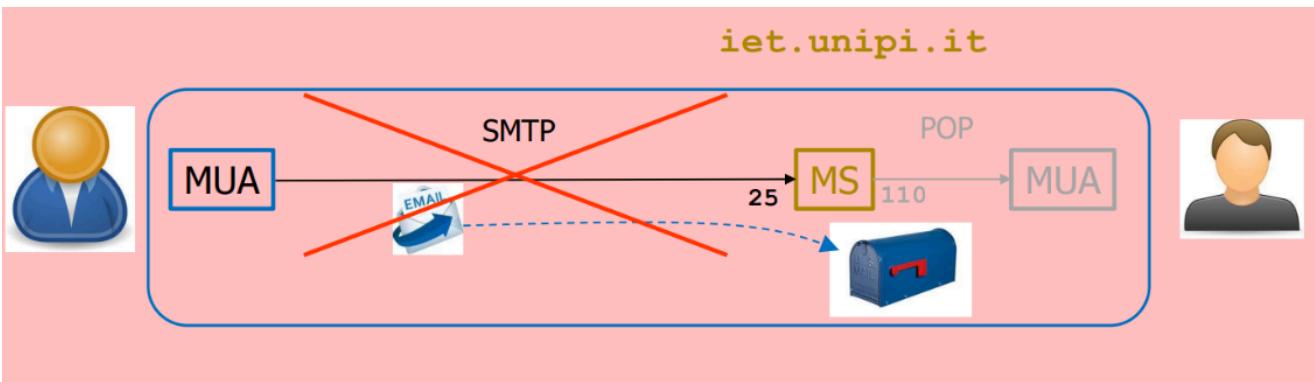
1. Partiamo dalla *mail client*, ovvero il programma che invierà richiese al MS. Essa dev'essere configurata:

- Da avere un *indirizzo e-mail*
- Tipicamente ha anche una *username + password* (di solito username è il nome della mailbox)
- Ogni mail client può quindi gestire più *indirizzi email*, per ora assumiamo che ne gestisca solo uno per semplicità concettuale

2. Supponiamo che adesso il mail client sia pronto per inviare la mail al destinatario. Abbiamo due casi:

1. Il *destinatario* ha lo stesso dominio e-mail del *mittente*: in questo caso è già *"arrivato"*, se lo deposita e basta
2. Se sono *diversi*, allora bisognerà invece fare richieste al DNS per determinare l'indirizzo dell'altro Mail Server (MX e poi A, oppure solo MX se il DNS dispone delle RR aggiuntive)

Osservazione. In questo paradigma il *mail client* si collega *soltamente* al proprio Mail Server, e *mai* ad altri *mail server* indipendentemente da dove si trova l'utente! Anche (e soprattutto) per trasmettere mail!



Questa era una cosa possibile tempo fa, ma non si fa più (a meno che non si vuole "*aggirare*" il sistema per qualche motivo strano...)

Protocolli e-Mail. Protocollo POP: definizione, paradigma e notazione. Comandi da sapere. SMTP: definizione, osservazioni su SMTP: versione autenticata e/o criptata. Comandi SMTP secondo RFC 821, formato delle mail secondo RFC 822.

0. Voci correlate

- Architettura e-Mail

1. Protocollo POP (RFC 1225)

Il protocollo **POP** (*Post Office Control*) è un protocollo a *comandi/risposte* ed è *text-based* su ASCII. Ovvero, i comandi e le risposte vengono codificati in caratteri ASCII. Una risposta viene codificata con:

- **+OK** se il comando inserito ha un esito positivo
- **-ERR** altrimenti, con informazione aggiuntiva

Una risposta può essere contenuta su più linee, e viene terminata col carattere punto **.**

NOTAZIONE. Per descrivere la comunicazione fatta mediante POP, denotiamo con **S** i comandi inviati dal server e **C** quelli dal client. In realtà non esiste, lo facciamo solo per chiarezza visiva

COMANDI. Vediamo i comandi inviabili dal lato client:

- **USER** **user@example.com**, **PASSWORD** **password**: Autenticarsi alla mailbox propria
- **LIST**: Elencare i messaggi. Il lato server mostrerà un elenco numerato di mail ricevute, con size dei messaggi relativi
- **RETR** **n**: Leggere la mail **n**-esima di quella ottenuta nell'elenco
- **DELETE** **n**: Eliminare la mail **n**-esima di quella ottenuta nell'elenco
- **QUIT**: Terminare la comunicazione

Esempio. (*Comunicazione tipica POP*)

```
S: +OK POP3 SERVER READY
C: USER alice@example.com
S: +OK USER ACCEPTED
C: PASS secret123
S: +OK PASSWORD ACCEPTED
C: LIST
S: +OK 2 MESSAGES
S: 1 1200
S: 2 850
S: .
C: RETR 1
S: +OK 1200 OCTETS
S: <mail>
S: .
C: DELE 1
S: +OK MESSAGE 1 DELETED
C: QUIT
S: +OK GOODBYE
```

Osservazione. In questo caso, è il *lato server* che inizia la conversazione: prima di poter procedere, deve dare "l'OK" al client.

2. Protocollo SMTP (RFC 821)

SMTP è un protocollo *molto intricato*, per motivi storici. Ricordiamo che ci sono due "*step*" in cui si va a comunicare col protocollo SMTP:

1. *MUA to MS*: Qui l'autenticazione e/o criptaggio è "*opzionale*". All'inizio non c'era proprio
2. *MS to MS*: Qui non c'è l'autenticazione

Quindi ci sono più "*versioni*" del protocollo SMTP che sono state aggiunte nel tempo, per poter aggiungere la possibilità di autenticazione e/o criptaggio. In totale, ci sono *3 numeri di porta diversi* con funzionalità auth/crypto diverse.

Nel corso *trascuriamo* l'uso della crittografia, e faremo finta di usare solo la porta 25.

SMTP. Similmente a *POP*, *SMTP* è un protocollo *ASCII text-based* a *comando/risposta*. Da un punto di vista funzionale, nelle risposte conta solo l'*identificativo numerico* (anche se c'è in realtà del testo descrittivo dell'esito, per aiutare l'utente).

COMANDI. Vediamo i comandi inviabili dal lato client:

- **EHLO/HELO <mail_domain>**: Il client si identifica come il *mail server* di un certo mail domain.
- **MAIL FROM: <s@sender.com>**: Iniziare a mandare una mail, dichiarando dapprima il mittente
- **RCTP TO: <r@receiver.com>**: Specificare il destinatario
- **DATA**: Dichiarare che si sta iniziando a comporre la mail.
- **QUIT**: Terminare la comunicazione
- **AUTH LOGIN**: Dichiarare che si sta iniziando ad autenticarsi. Da qui la mail server potrà chiedere l'username e la password, che verranno inseriti in *codifica Base64*

Vediamo un esempio tipico:

```

S: 220 mail.receiver.com ESMTP Postfix
C: EHLO mail.sender.com
S: 250-mail.receiver.com
S: 250-AUTH LOGIN PLAIN
C: AUTH LOGIN
S: 334 ...
C: ...
S: 334 ...
S: 235 2.7.0 AUTHENTICATION SUCCESSFUL
C: MAIL FROM:<s@sender.com>
S: 250 Ok
C: RCPT TO:<r@receiver.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: Server-to-server test
C: This is a test message sent without authentication.
C: .
S: 250 Ok: queued as 67890

```

X

3. Formato Mail (RFC 822)

Q. Con SMTP e POP possiamo vedere le *mail*. Come si scrive in effetti una mail?

Osserviamo innanzitutto che per *mail* - inteso come il *messaggio* - non si ha solo il contenuto, ma l'*involturlo in sé*. Ovvero, essa è composta da due parti:

Headers: Insieme di header lines **Name: Value**, in cui si hanno delle informazioni aggiuntive per gestire le mail.

Hanno un ruolo analogo alle headers HTTP; tuttavia, la differenza consiste nel fatto che in questo caso gli header possono essere generati da più processi:

- *Mail Client mittente*
- *Mail Server intermedi*
- *Mail Client ricevente*

Osserviamo che di solito nelle applicazioni MUA vediamo solo un *sottoinsieme degli header*, di solito c'è l'opzione per vederli tutti

Body: Il contenuto vero e proprio della mail. Essi sono codificati in *linee di caratteri ASCII 7-bit*, e l'ottavo bit dev'essere sempre impostato a 0 (per ragioni storiche e intricate di cui non conosciamo).

Per allegare dei file o per avere più bit, abbiamo delle *estensioni optional* da parte della MUA sender per trasformare il body originale in ASCII 7-bit, e poi nell'header si specificano *"come"* vengono effettuate queste trasformazioni; così la MUA receiver sarà in grado di effettuare la trasformata inversa del body.

Email Spam e Address Spoofing

X

Problemi pratici legati ai protocolli email. Spam e Address spoofing. Mitigazioni allo spam e address spoofing.

X

0. Voci correlate

- Protocolli Email

1. Problemi Pratici

Ci sono dei *problemi pratici molto importanti* legati all'architettura mail e ai protocolli usati. Questi problemi non hanno delle soluzioni in quanto sono intrinseche al modo in cui è stata designata l'architettura mail e configurazione delle mail server, bensì abbiamo delle *mitigazioni abbastanza "efficienti"*.

I problemi sono due:

Spam:

- Invio massimo di email "*indesiderati*"
- Fini commerciali e spesso illegali
- Oggi compongono la maggioranza degli email inviati

Spoofing:

- Falsificazione indirizzo dell'email del mittente.
- Questo è "*facilissimo*" da fare! Infatti, osserviamo che nel protocollo SMTP sarebbe teoricamente possibile specificare il mittente di una mail senza autenticazione. Per farlo, basterebbe collegarsi direttamente ad una mail server e inviare una mail specificando l'indirizzo mittente falsificato... (usando i comandi **MAIL FROM** e l'header **From: ...**)
- In realtà oggi è molto difficile da fare, siccome le "*mitigazioni*" che abbiamo oggi sono piuttosto efficaci

In pratica le mitigazioni ai due problemi consistono in applicare delle *euristiche* per conoscere mail *spam o "spoofed"*. Come, ad esempio:

- Indirizzi SMTP diversi da indirizzi in email
- Domini dichiarati in EHLO non esistenti
- Header mancanti
- Analisi del testo usando algoritmi ML (nota! oggi questi algoritmi hanno un'accuracy del circa 0.99)
- Uso di protocolli ancora più specifici per l'autenticazione

- eccetera...

Ma i problemi persistono perché:

- **Spam**: Le mail spam sono moltissime, e quindi nonostante l'accuratezza dei modelli di classificazione, alcuni vengono comunque "fatte passare"
- **Spoofing**: L'attacker, se disposto di risorse sufficienti, sarà comunque in grado di falsificare una mail che passi i filtri euristici descritti. Inoltre, un'alternativa più semplice del **spoofing** è quello semplicemente di **ingannare** registrando domini mail con nomi che "**assomiglino**" a certi enti (principio derivante dal caso delle DNS). Esempio: l'attacker può comprare il dominio **polizia-postale-it.it** e crearcì delle RR di tipo MX..

Collegamento Diretto con i Server Internet

X

Cuoriosità pratica: programmi per aprire connessioni a server manualmente. Programma telnet, esempio con email (SMTP e POP). Problemi pratici dell'esempio.

X

0. Voci correlate

- Comunicazione tra Processi
- Proprietà dei Servizi di Comunicazione
- Protocolli Email

1. Connessione Diretta con i Server Internet

RICHIAMO. Ricordiamo che ogni *server* implementa un protocollo, che descrive come devono essere formattate le request o response. Molti protocolli hanno request/response *a linee di caratteri*.

Allora si potrebbe comunicare "*direttamente*" con i server che implementano tali protocolli dal proprio PC, usando il programma *telnet*. Tramite il programma posso *aprire connessioni a server* e *ciò che scrivo è il messaggio*, e il *messaggio che vedo stampato a schermo è la risposta*.

I comandi per usare telnet sono semplicemente le seguenti:

- **telnet**: Aprire il programma
- **open <server_name> <portnumber>**: Aprire una connessione
- **set LOCAL_ECHO**: Per vedere ciò che viene stampato a schermo

Osservazione. Il programma funziona in un modo tale che i backspace non vengano riconosciuti come caratteri, ovvero è impossibile "*correggere*" il testo. L'unica cosa che si può fare è quello di interire enter e sperare che ci sia uno "errore di sintassi"

X

2. Esempio di Utilizzo: Interazione coi Mail Server

Esempio. Ad esempio, si può usare telnet per interagire coi *mail server*. Tuttavia, ci sono moltissimi problemi che renderebbe questo caso d'utilizzo difficile, tra cui:

- Il router di frontiera potrebbe proibirci il traffico SMTP
- Il Mail Server potrebbe chiudere la connessione in quanto non ci riconosce come un altro mail serve
- Il Mail Server potrebbe chiudere la connessione in quanto non siamo autenticati

Quindi ci sono altri programmi che ci permettono di fare tutto ciò, tra cui:

- **swaks** per specificare tutte le opzioni nelle SMTP request
- **openssl s_client** per criptare
- PuTTy per gestire protocolli criptati

Aspetti Vari delle Mailbox

X

Aspetti vari sulle mailbox: *forwarding*, *mailing list* e *send as*.

X

0. Voci correlate

- Architettura e-Mail

1. Aspetti Misti

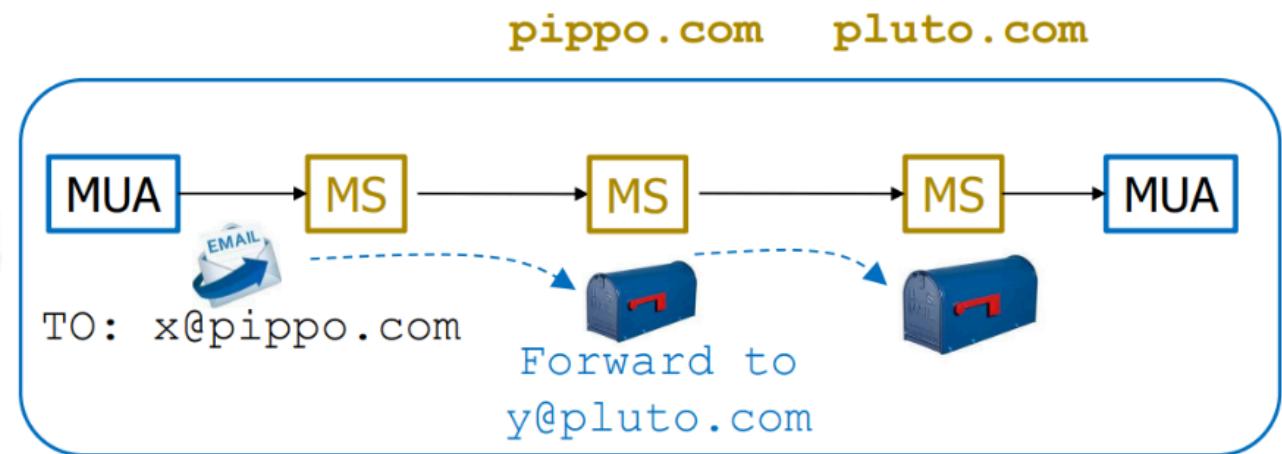
Supponiamo di avere una *mail* che "passa" tra due mail server, MS e MS'. Se $MS \neq MS'$, allora si dovrebbe aspettare che la mail *passi* tra solo *due server*.

1.1. Mail Forwarding

In realtà vedremo che non è sempre vero, per gli aspetti sulle mailbox che vedremo.

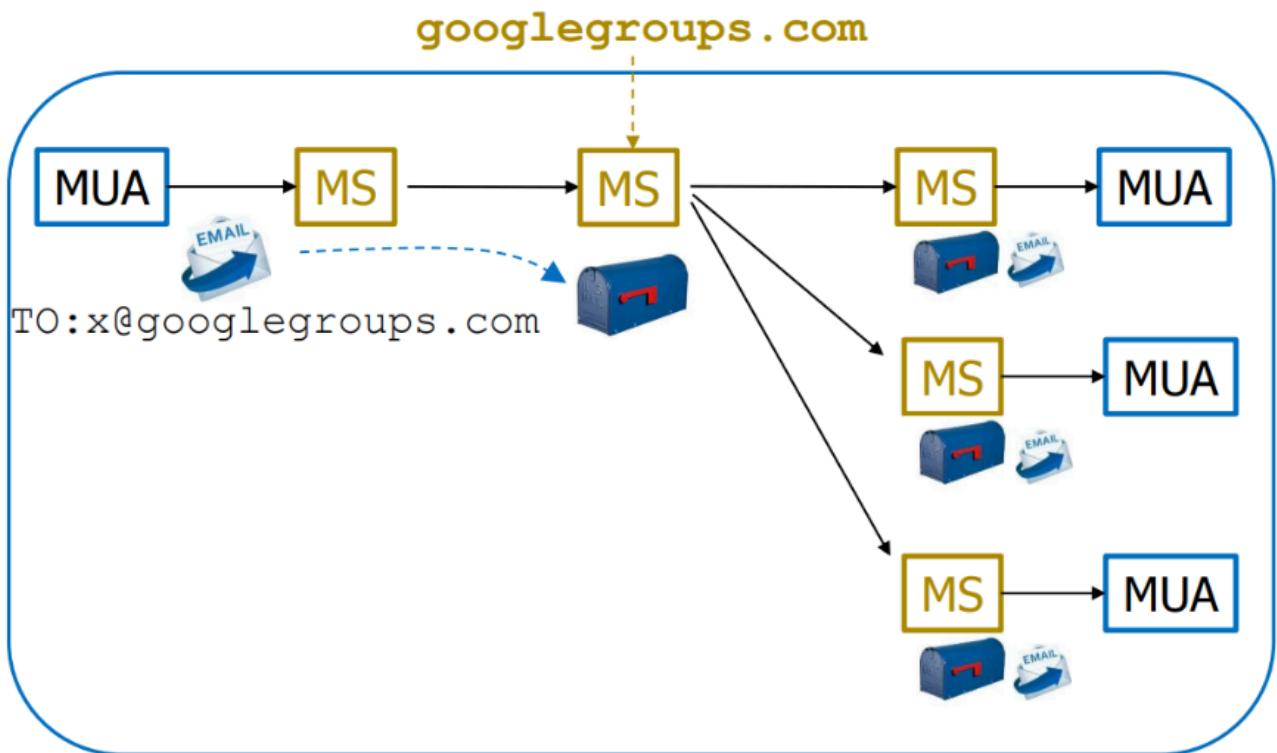
Cominciamo dapprima con la *forwarding*: in pratica, ogni *mailbox-A* può essere configurata per *inoltrare* una mail ricevuta verso ad un'altra *mailbox-B*, e optionalmente cancellare la mail.

Quindi, in questo caso si ha che la mail passa per *tre server*.



1.2. Mailing List

In altri *mail server*, è possibile configurare una *mailbox* come una *mailing list*; ovvero di associare un *insieme di mailbox* ad una *mailing list*, e ogni email ricevuto dalla mailing list viene trasmessa a *tutte le mailbox* nella mailing list.



Osservazione. La mailing list può comprendere mailboxes di *dominio diverso!* Quindi la mail passa per più mail server

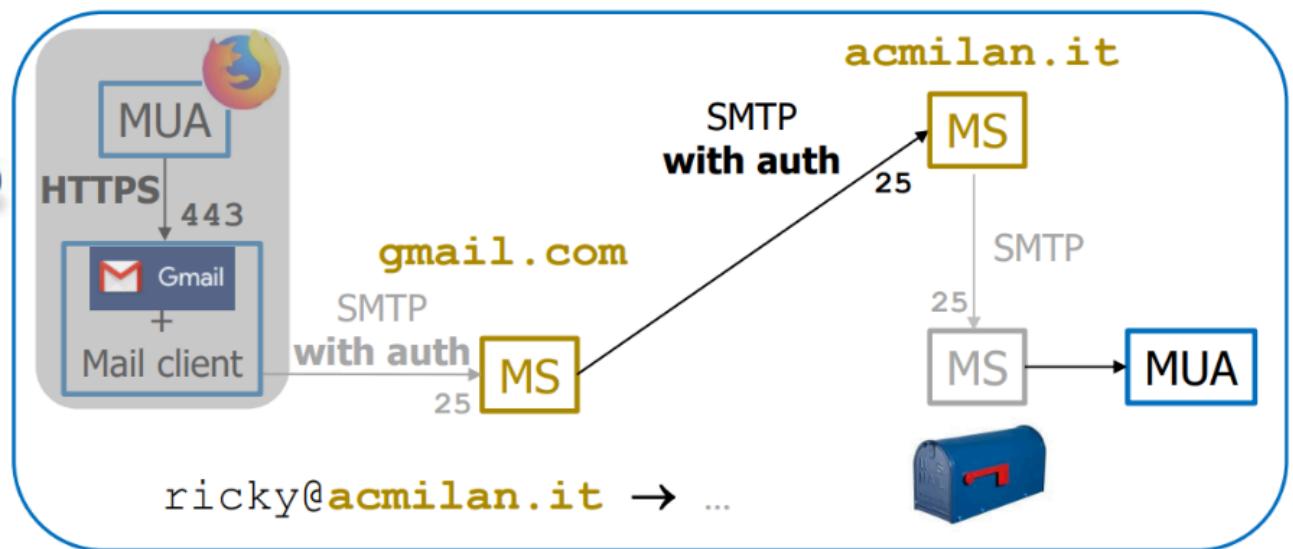
1.3. Send As

In alcuni *mail client* (come `gmail`), è possibile scegliere di mandare le mail con un'*identità diversa*. Tuttavia, ciò non vuol dire che posso usare un dominio email *mittente* arbitrario! (sennò sarebbe spoofing...)

In realtà funziona che il *mail server* invia la mail all'altra *mail server* (che si riferisce alla "*identità scelta*"), tramite protocollo SMTP *con autenticazione*; dopodiché il mail server lo invia come di consueto.

Quindi tutto ciò che serve per collegarsi all'altro mail server è definito *in configurazione*, tra cui username, password, eccetera...

Come con la forwarding, abbiamo che la mail passa per *tre* mail server.



"Protocollo IP"

SEZIONE A. NETWORK

Fondamenti di Network

X

Fondamenti sul network. Definizione di Network: punto di vista "pratico" e punto di vista implementativo. Esempio di network tech: Ethernet, Bluetooth.

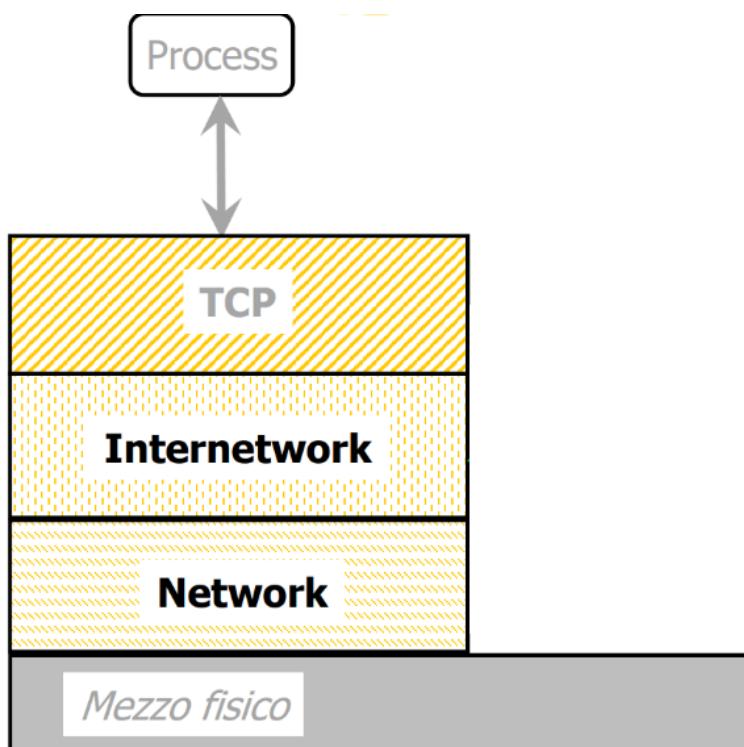
X

0. Voci correlate

- Comunicazione tra Processi
- Proprietà dei Servizi di Comunicazione

1. Fondamenti sul Network

Ricordiamo che per quanto riguarda la comunicazione di *processi*, a livello implementativo (approx.) essa avviene su *più livelli*. Fin'ora (ovvero fino a quanto avevamo trattato DNS, Web e Mail) abbiamo visto il livello più alto, ovvero livello *TCP/UDP* usato dai processi.



Adesso scendiamo di livello. Partiamo dal *network*, ovvero il "*primo*" (o l'ultimo, dipende dall'ordine) livello di comunicazione tra processi.

Network. Per parlare del *network*, distinguiamo due aspetti:

1. L'aspetto "*funzionale*", ovvero vogliamo descrivere il network dal punto di vista di "*internetwork*" (o superiore)
2. L'aspetto *implementativo*, ovvero "*come è fatto*".

"Come si usa": Nel network, abbiamo che ogni *nodo* (calcolatore) può comunicare con un *qualsiasi altro nodo* che abbia la stessa *network tech* (hardware + software).

Inoltre, ogni *collegamento* nodo possiede un *identificatore univoco*, l'indirizzo *network*. Essa identifica l'*interfaccia di rete* nel *network*.

Infine, ogni *network tech* possiede le stesse proprietà sui servizi di comunicazione: quindi il concetto di connectionless/connection oriented, reliability/unreliability e byte-oriented/message-oriented. Inoltre, vengono specificati altri aspetti tra cui *formato degli indirizzi network*, *massimo numero di nodi collegabili tra di loro*, *massima distanza fisica di collegamento* e *la massima dimensione dei messaggi da consegnare* (MTU) (se message oriented).

Gli aspetti di *max nodes* e *max distance* dipendono dalla tecnologia usata, il resto è stipulato mediante le convenzioni (lato sw).

Le network tech possono essere suddivise in due "*tipologie*", a seconda delle loro proprietà.

- *WAN* (Wide Area Network): Estensione geografica
- *LAN* (Local Area Network): Estensione limitata, ma latenza bassa

"Come è fatto": Al livello implementativo, dipende dalla network tech usata. Generalmente, i processi si scambiano "messaggi" mediante i *frame*, che sono composti da *payload* e *control information* (formato specificato da ogni network tech).

X

2. Esempi di Network Tech

ESEMPIO. *Ethernet* è una network tech con le seguenti caratteristiche:

- Connectionless
- Unreliable
- Message-oriented
- Ogni indirizzo ethernet ("MAC") è un numero naturale composto da 6 byte
- All'incirca si collegano 100 nodi massimo, con massima distanza di 100 metri
- MTU è di 1500 byte
- L'indirizzo FF:FF:FF:FF è riservato ai messaggi "*broadcast*", ovvero quelli che vengono mandati a tutti nel network (approfondiremo opportunamente)

Implementazione di ethernet. Reminder sulle proprietà di ethernet, formato dell'indirizzo ethernet (MAC). Formato di ethernet frame. Concetto di switch e access point. Funzionamento dei switch, switch table. Estensione delle switch: "fondare" network ethernet. Access point: funzionamento. Osservazioni.

0. Voci correlate

- Fondamenti su Network

1. Indirizzi Mac, Ethernet Frame e Proprietà Ethernet

Vediamo adesso l'implementazione di *ethernet*, ovvero cosa succede quando un nodo manda un messaggio all'altro.

Vediamo i primi aspetti implementativi dell'ethernet.

INDIRIZZI ETHERNET.

Partiamo dagli *indirizzi ethernet*, conosciuti anche come *indirizzi MAC*. Sono formati da *6 byte* e rappresentano univocamente una scheda ethernet ognuna, ad eccezione dell'indirizzo con tutti 1. Per quanto riguarda il formato, la rappresentiamo in *notazione esadecimale* (che è facilmente convertibile a partire da multipli di byte), e ogni "*due cifre*" vengono divise dal simbolo dei due punti .

L'unico indirizzo che non può essere usato è l'indirizzo broadcast FF:FF:FF:FF:FF:FF, ovvero l'indirizzo con tutti uno. Questo viene usato per instradare pacchetti che vengono *inviati a tutti* nel network (vedremo dopo nei dettagli perché e come).

Esempio. 11010001 00111000 11110111 11010001 00111000 11110111 diventa D1:38:F7:D1:38:F7 (basta suddividere ogni byte in due parti da 4 bit, e convertire ciascuna parte in esadecimale)

ETHERNET FRAME.

Ogni *messaggio* è il *payload di un frame Ethernet*. Ogni payload frame viene trasmessa in una maniera *seriale*, ovvero "*viaggiano un bit dopo l'altro, in sequenza*". Ci sono due mezzi di trasmissione di frame:

- Cavo (*wired*)
- Segnale radio (*wi-fi*)

Ognuno di questi tipi permettono una "velocità" diversa di trasmissione. Osserviamo che quindi in una network coesistono nodi con "velocità" diverse.

Per quanto riguarda il frame stesso, ricordiamo che è composto da *payload* e *control information*.

Preamble (8)	DST Address (6)	SRC Address (6)	Type (2)	Payload (<1500)	CRC (4)
-----------------	--------------------	--------------------	-------------	--------------------	------------

Gli aspetti da ricordare sono *DST address*, *SRC address*, *type* e *payload*. Gli altri sono dettagli da trascurare.

PROPRIETA' ETHERNET.

Come detto prima, ricordiamo le seguenti proprietà dell'ethernet e spieghiamo come si "adattino" in questo contesto.

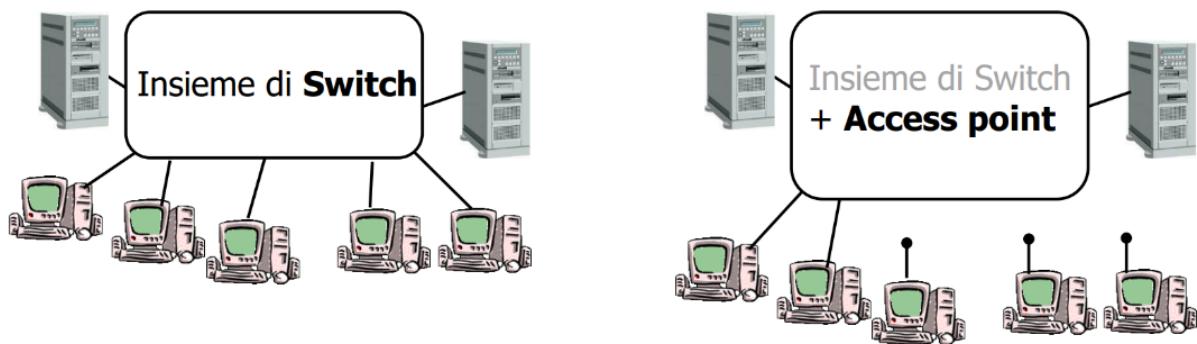
- *Message-oriented*: Ogni "messaggio" è un frame
- *Connectionless*: Non devo avere un "contesto" per mandare frame, e i frame sono indipendenti tra loro
- *Unreliable*: I nodi possono decidere di "scartare" i frame senza avvisare il mittente

X

2. Switch e Access Point

Q. Più nodi hanno velocità diverse, come si adattano? A cosa si collegano?

Essenziale i concetti di *switch* e *access point*, che andranno a definire un insieme di dispositivi collegati tra loro.



Gli *switch* servono per connettere dispositivi connessi via cavo, *access point* invece per dispositivi connessi via Wi-Fi.

Osserviamo che da un punto di vista funzionale dell'utente, non è necessario sapere *come* funziona l'architettura switch/AP per inviare messaggi. Al nodo basta solo l'indirizzo destinatario per inviare un messaggio.

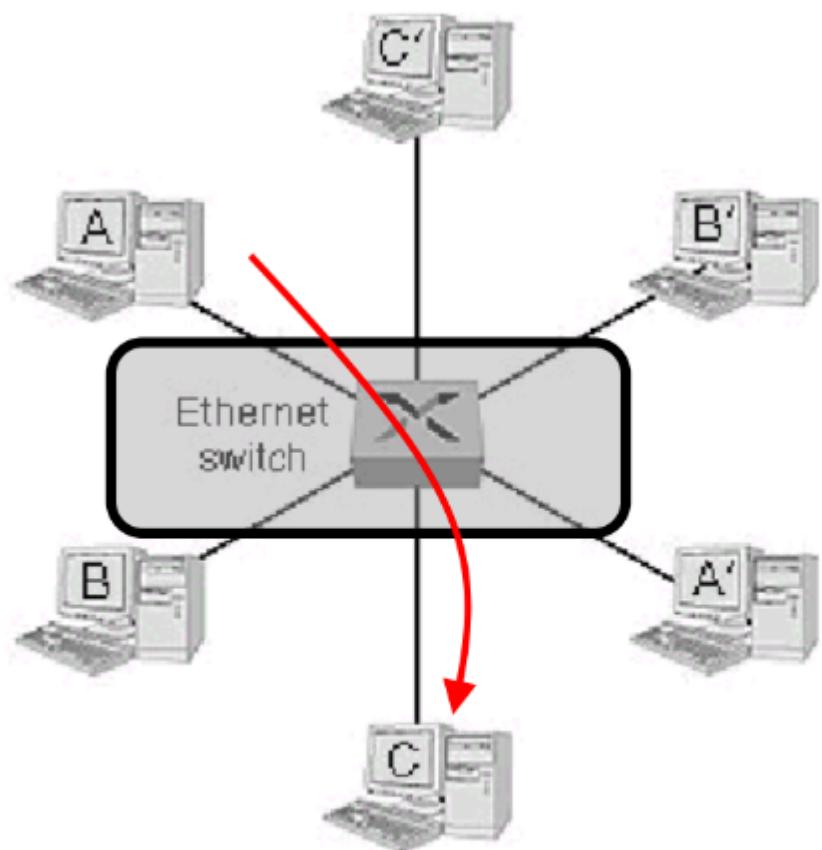
2.1. Switch

Uno *switch* è uno dispositivo composto da *porte*, a cui si possono collegare sia *nodi* che *altri switch*.



Uno switch, sostanzialmente, effettua le seguenti operazioni:

1. Ricevono un *frame* in bit (da una porta "*vedono*" i bit)
2. Analizzano l'*indirizzo destinatario*
3. Instradano il frame alla porta dell'*indirizzo destinatario* (molto velocemente!)



! Gli switch non hanno indirizzo ethernet, siccome sono completamente trasparenti dall'utente.

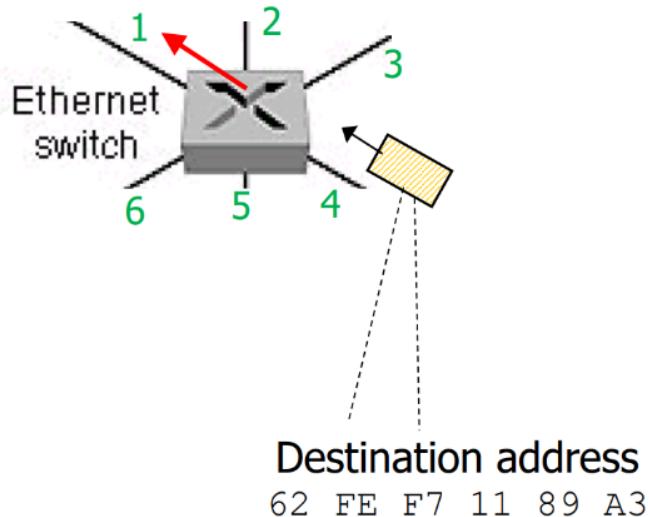
Inoltre, gli switch possono anche instradare più frame *simultaneamente* (come? dipende dalla loro qualità, diciamo...) e devono *gestire velocità di collegamento diverse* (quindi necessità di implementare funzionalità di FRAME QUEUEING).

SWITCH TABLE.

Q. Come fa il switch a conoscere la porta da instradare?

Ogni switch configura in sè automaticamente una *switch table*, ovvero una tabella di ricerca in cui ogni riga è una corrispondenza *port* ↔ *MAC address*.

La switch table viene inizializzata come una tabella vuota, e ogni volta che un nodo invia un *messaggio* lo switch ispezionerà il source address dei frame e salva (port, ETH-src) sulla tabella.



Address	port
01-12-23-34-45-56	2
62-FE-F7-11-89-A3	1
7C-BA-B2-B4-91-10	3
....

Approfondiamo lo *step 3.* del funzionamento dei switch appena visto (^54fd08).

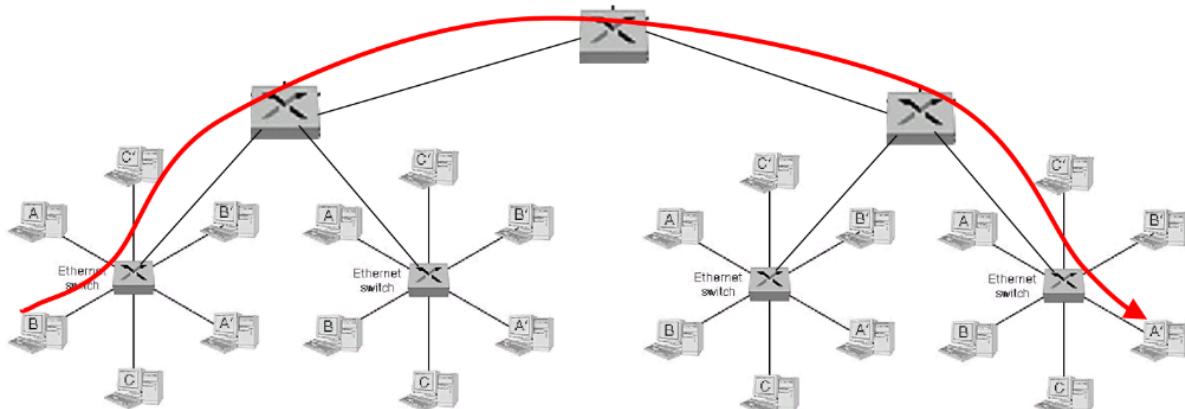
- Se lo switch non conosce la *port* di *ETH-dst*, può adoperare due strategie:
 - *Discard*: Buttare via il frame, va bene in quanto Ethernet è unreliable (e quindi fa parte del gioco)
 - *Overflow*: Inviare il frame a tutti (*ottimizzazione*)

Entrambe le strategie vanno bene. Per riempire la switch table, intuitivamente si ha che ogni volta che un nodo si collega alla network esso invia un *frame* per "farsi scoprire".

Q. Se un switch deve inoltrare un frame ad un indirizzo MAC che non è "*suo*" ma di un altro switch, come fa?

Basta estendere il concetto di switch table, ogni riga può contenere anche *più indirizzi ethernet*.

In questo modo, ogni volta che lo switch deve instradare un pacchetto ad un altro "*switch di competenza*" lontano, basta che instradi il messaggio a quel switch.

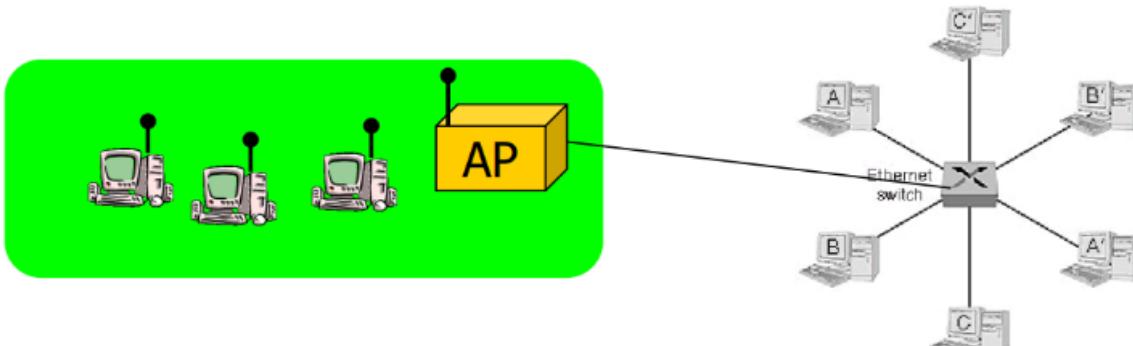


Qui notiamo una peculiarità degli switch: *è tutto automatico!*

2.2. Access Point

Gli *access point* sono dei dispositivi *a due lati*, dove uno *si collega ad uno switch* e l'altro *comunica con gli altri dispositivi "vicini"*. In ogni *cella* c'è un *access point*.

Gli *access point* sono equivalenti ad uno *switch* che non supporta comunicazioni simultanee



Curiosità. Nella realtà i pacchetti trasportati su *wifi* sono *"leggermente"* diversi da quelli trasportati via cavo. Non approfondiremo.

X

3. Ulteriori Osservazioni

Osserviamo che nel gergo tecnico, *network* può anche indicare *alcune parti di network* (che è anche *"sbagliato"*).

Nel nostro caso definiamo *network* nella seguente maniera precisa:

- A, B appartengono alla stessa *network* sse A può ricevere frame inviati da B

Osservazione. Nella realtà, quando compriamo uno *switch/AP* compriamo anche il *calcolatore in più* che serve per la configurazione del network. Per noi uno *switch/AP* è solo lo *switch/AP*

SEZIONE B. INTERNETWORK

Fondamenti di Internetwork

X

Aspetti motivanti dell'internetwork, terminologia Internet e Internetwork, proprietà e funzionalità. Implementazione dell'internetwork, pacchetti IP. Azioni del sender, receiver e del router. Domande da rispondere sull'internetwork (programma del capitolo).

X

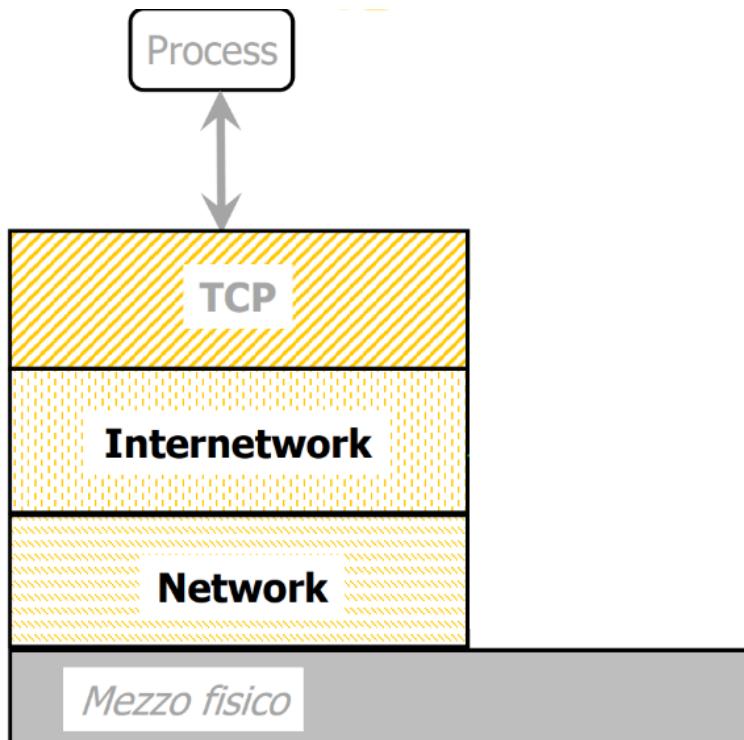
0. Voci correlate

- Fondamenti su Network

1. Introduzione all'Internetwork

PROBLEMA. Abbiamo moltissimi nodi con *tecnologie network diverse* o *lontane tra loro*, come possiamo integrarli in un'unica struttura di comunicazione? E se sono milioni di nodi? (nessuna network tech è in grado di integrarli in un'*unica* network...)

Il concetto di *internetwork* andrà a risolvere il problema appena posto, e andrà a rappresentare lo strato di astrazione "*sopra*" i network:



DEFINIZIONE. In particolare, definiamo l'*internetwork* come una *struttura logica* ("virtuale") dove ogni *nodo ha un identificatore unico* e può comunicare con *un qualsiasi altro nodo*.

OSSERVAZIONE. (*terminologia*)

Nel gergo tecnico, il termine "*internet*" può indicare due cose differenti; o intendiamo l'*internetwork*, o intendiamo *una particolare network*, che è quella usata tutti i giorni. Nel secondo caso, di solito si scrive *Internet* con la I maiuscola.

Ci sono molti "*modi*" per implementare l'internetwork, ma il nostro unico caso di interesse è l'*IP protocol*.

Proprietà. L'internetwork *IP* è un servizio di comunicazione che possiede le seguenti proprietà:

- Connectionless
- Unreliable
- Message-Oriented
- Indirizzi IP formati da 4 byte, con una certa "*struttura*" (vedremo dopo molto approfonditamente)
- MTU di 64 KB
- I nodi possono essere "*moltissimi*" e "*lontanissimi tra loro*" e possono usare network technologies diverse

X

2. Implementazione dell'Internetwork

Vediamo l'*internetwork* da un punto di vista implementativo. L'idea di base è quello di trovare un modo di collegare nodi di network diverse.

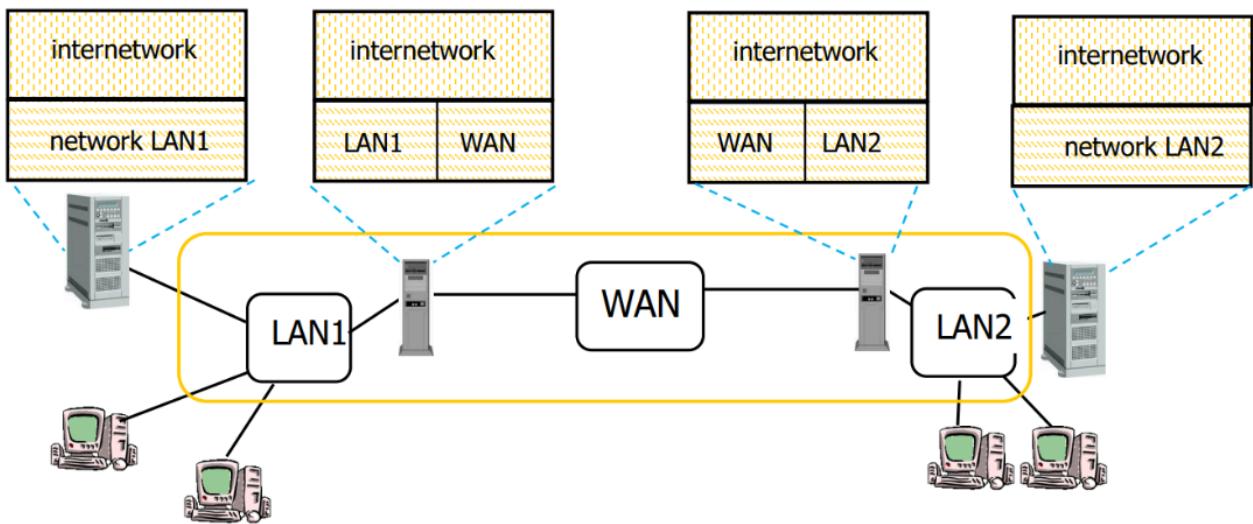
Indirizzi Nodi. Partiamo dall'*indirizzo dei nodi*. Ogni nodo in realtà possiede due indirizzi:

1. Indirizzo della *network* a cui appartiene, si dice *indirizzo fisico*
2. Indirizzo della *internetwork*, si dice *indirizzo logico*

Gli uni non hanno nulla a che fare con gli altri. Di solito, l'*indirizzo fisico* sarà solitamente l'indirizzo *ethernet*, invece l'indirizzo logico sarà l'*indirizzo IP*.

Router. Vediamo addesso come possiamo "*incollare le network*". Ci sono dei *nodi in più* per "*intradare*" e "*convertire*" pacchetti di network diverse, e si dicono *router*. Ogni router ha quindi più schede ed indirizzi network (!).

E' importante osservare che i router hanno *più schede network*, quindi più *indirizzi logici* e *fisici*!



Osservazione. Notiamo che quindi in ogni end point dobbiamo avere *software* che implementa l'internetwork.

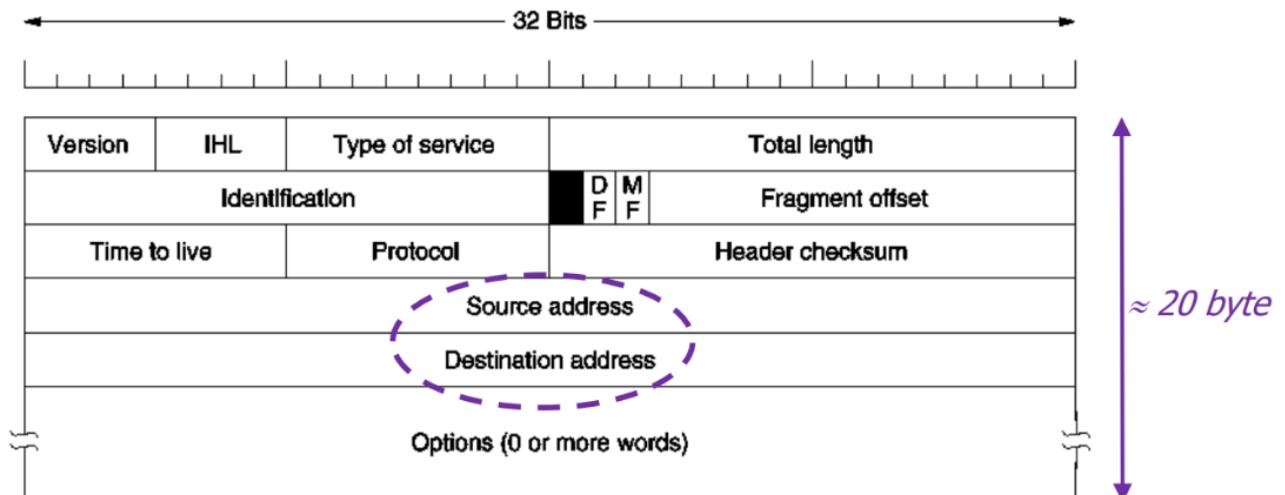
Osservazione. Solo lo *strato TCP/UDP* usa il protocollo IP, quindi ci troviamo in un livello basso di astrazione. Infatti, teniamo nota che le applicazioni *non* usano direttamente il protocollo IP.

Pacchetti. I "frame" (pacchetti) IP sono composti da due parti: *payload* e *header*.

Sinteticamente, possiamo descrivere ogni pacchetto come "*send m to IP-dst*" o "*received m from IP-src*". Gli IP header che ci interessano sono solo due, IP-dst e IP-src (gli altri sono header di controllo).

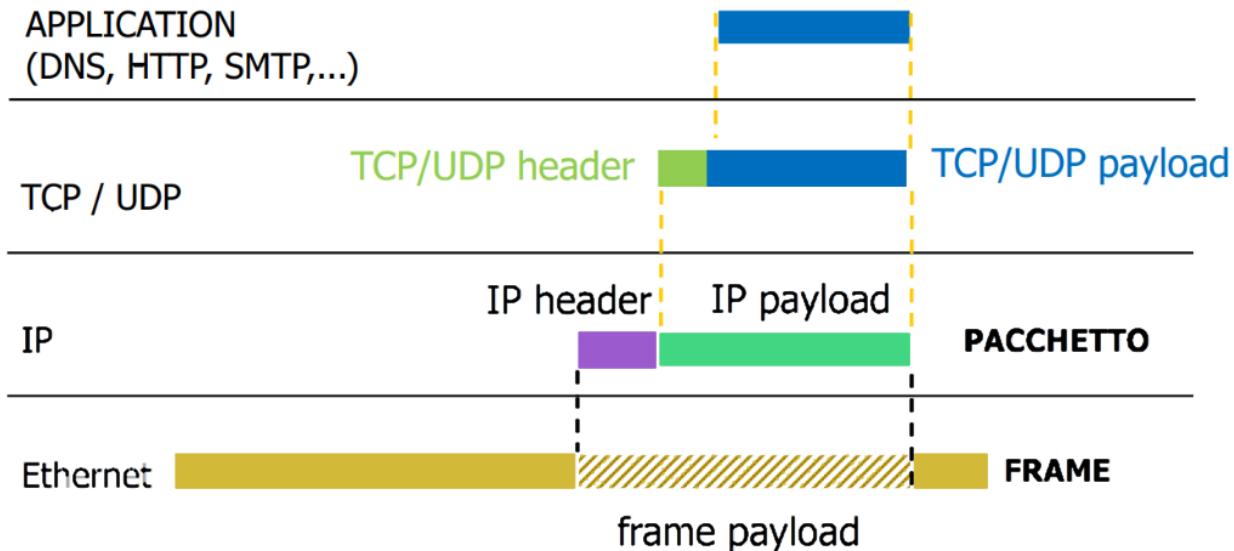
La parte "byte" è interessante, e indica come va interpretato il payload. In particolare è formato da due byte, e ogni numero indica un protocollo specifico. Nei protocolli che vedremo:

- 800 è IP
- 806 è ARP



Encapsulation. Per fare mente locale, vediamo lo scenario dell'*incapsulazione* dei pacchetti, ovvero quando dal lato applicativo vogliamo inviare un messaggio, per ogni strato che *"mandiamo giù"*, il messaggio diventa il payload di un altro frame, con ulteriori informazioni di contesto.

Esempio: HTTP request -> TCP/UDP frame -> IP packet -> Ethernet Frame



Esercizio. Scrivere il pseudocodice di un programma che invia una richiesta HTTP per prelevare un certo documento in URL-x. Riferendosi a questo programma, supponiamo di eseguirlo e tracciare tutto il traffico in un unico file. Cosa vediamo nel file?

Traccia: Contare prima le DNS request per risolvere URL-x (o proxy se facciamo ulteriori ipotesi), poi l'apertura della connessione TCP e infine le richieste/risposte HTTP opportune.

X

3. Azioni Sender, Receiver e Router

Vediamo addesso *come* vengono mandati i pacchetti, dal punto di vista dei nodi. Partiamo innanzitutto dalla *configurazione* dei calcolatori.

Ogni nodo deve avere in configurazione le seguenti informazioni:

1. *IP Address* del nodo mittente
2. ... (vedremo dopo)
3. *IP-Gateway*: Default Gateway (*router collegato alla stessa network del nodo*)
4. *IP-Name Server* (questa non è indispensabile per la sola connettività IP, ma a fine pratici è necessario avere in configurazione)

Adesso siamo pronti per descrivere le azioni in dettaglio

Sender. Il sender effettua le seguenti operazioni, descritte dal seguente pseudocodice:

- IF IP-dst "si trova sulla mia network" (!):
 - Pacchetto è payload di un frame ethernet indirizzato all'indirizzo fisico di IP-dst (!!)
- ELSE:
 - Pacchetto è payload di un frame ethernet indirizzato all'indirizzo fisico di IP-Gateway (!!)

Receiver. Deve decifrare il *pacchetto IP*; deve quindi conoscere il *protocollo IP* (implementato con del software) per "*decifrare*" il payload

Router. Anche qui pseudocodice:

- IF IP-dst != my_IP:
 - "Devo instradare pacchetto", ossia
 - IF IP-dst "si trova su una delle mie network" (!):
 - Pacchetto è il payload di un frame indirizzato all'indirizzo fisico di IP-dst (!!)
 - ELSE:
 - Sia RX il *router per arrivare a IP-dst* (!!)
 - Pacchetto è il payload di un frame indirizzato all'indirizzo fisico di RX (!!)

Notiamo che nei procedimenti descritti, ci sono un sacco di operazioni descritte "*idealmente*", senza aver idea di come vengono effettuate. Sono segnati con i punti esclamativi, e le operazioni da approfondire meglio sono le seguenti:

- *Ottenere le informazioni di configurazione*
- *Avere un test per dire se due indirizzi IP si trovano sulla stessa network o meno* (!)
- *Risalire a indirizzi fisici dagli indirizzi IP* (!!)
- *Trovare un percorso "corretto" che instradi un pacchetto* (!!)

Per risolvere i problemi, vedremo gli seguenti argomenti:

- Configurazione statica/dinamica (DHCP)
- Network number, subnet mask
- Protocollo di risoluzione ARP
- Routing

In realtà, sarebbe da farsi una domanda aggiuntiva: "*Cosa fare se un pacchetto è troppo grande per un frame?*"; la risposta viene data dal concetto di *frammentazione*, che non vedremo.

Osservazione. In ogni nodo di una internetwork, esso comunque conosce *solo* la network a cui si collega. Infatti, per mandare pacchetti non sa niente dell'internetwork! In un certo senso, ha l'illusione di comunicare con tutti mediante *l'ethernet*.

Modello Fisico dell'Internet

X

Modello fisico (approssimato) dell'Internet. Modi di collegare endpoint. Router ISP, collegamenti locali (Internet Exchange Point). Osservazioni pratici: dispositivi con più schede network, throughput nominale. Cenni storici alla topologia dell'Internet. Notazioni per disegnare reti. Osservazione: l'inaffidabilità dell'internet.

X

0. Voci correlate

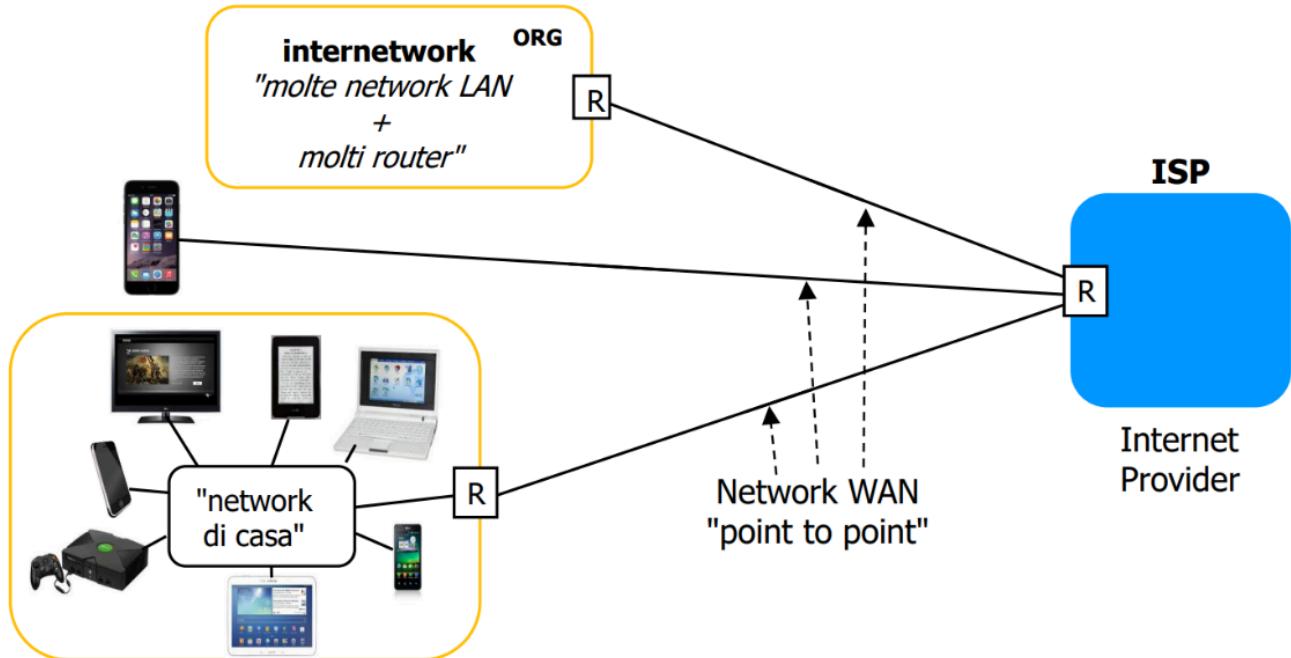
- Fondamenti su Internetwork

1. Modello Approssimato dell'Internet e Cenni Storici

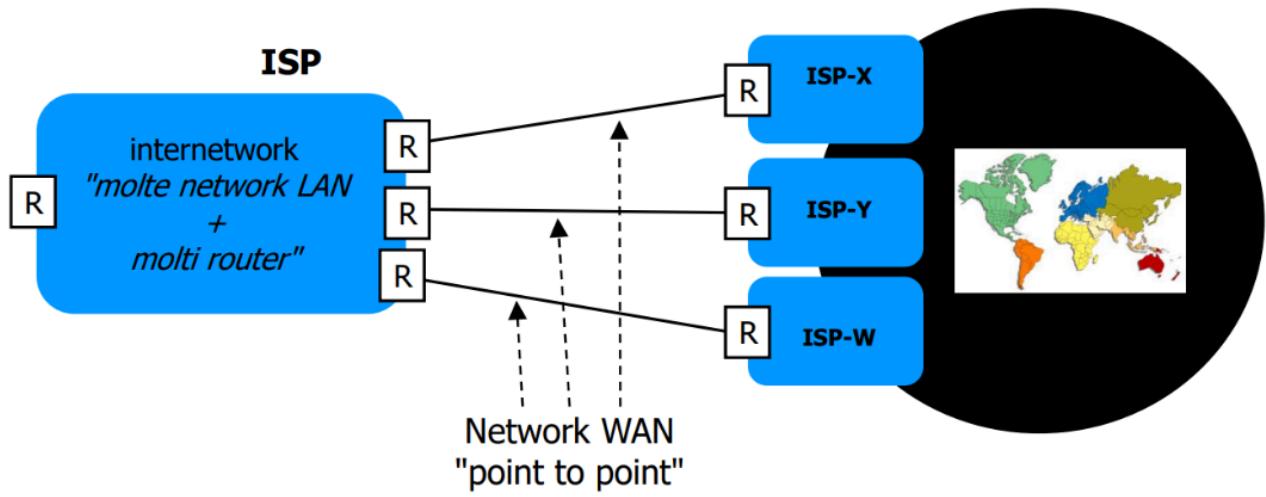
Descriviamo un *modello fisico approssimativo dell'Internet*.

Gli endpoint si collegano ad un "*router ISP*" (vedremo dopo cos'è) in tre modi:

1. In maniera *diretta*, mediante una network WAN "point-to-point"
2. *Network Ethernet* in cui collegiamo più dispositivi ad un *router di frontiera*, che poi si collega al router ISP
3. *Internetwork*, quindi più router e poi *uno di frontiera*



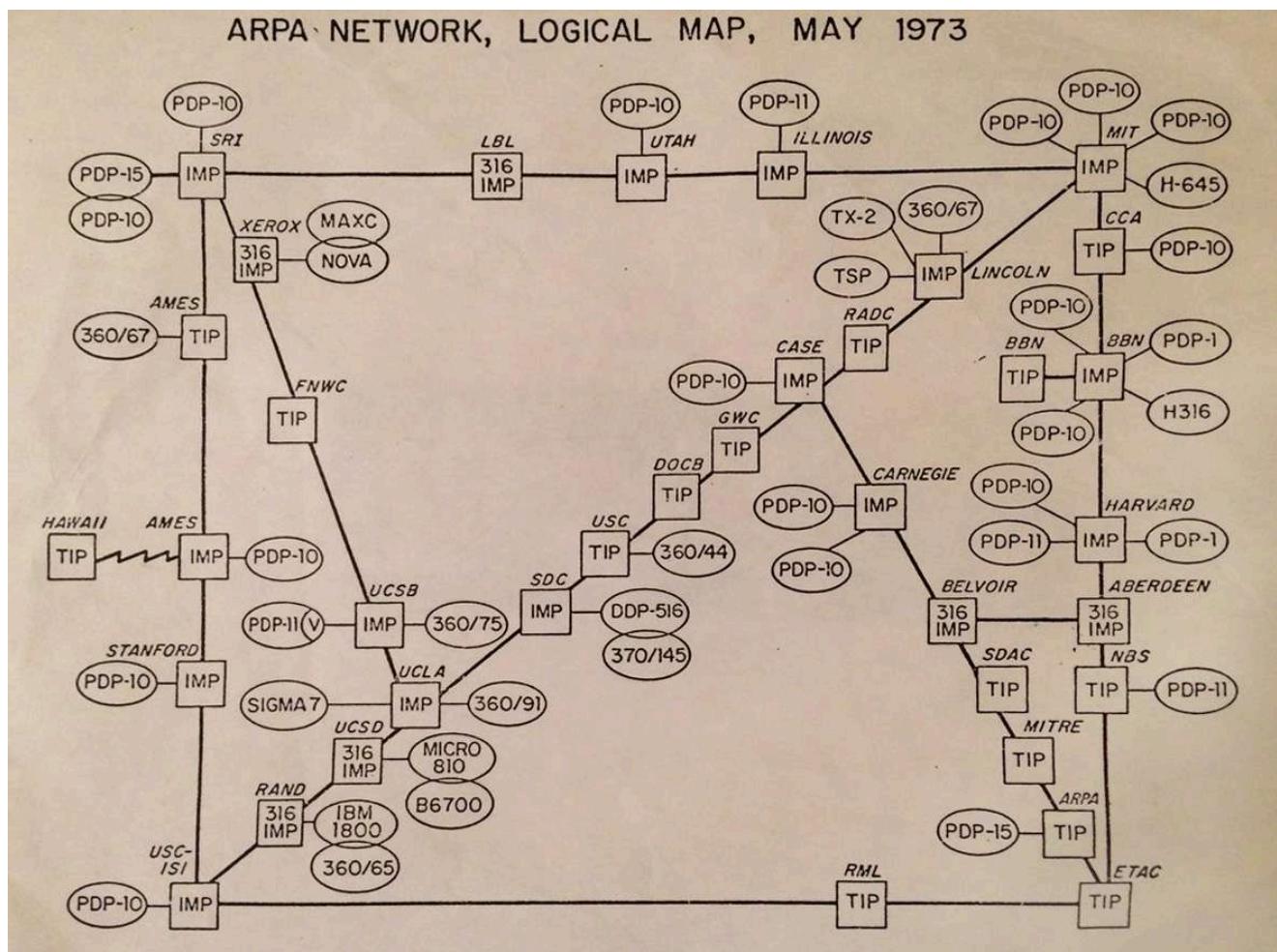
Dopo, nello "*strato successivo*" abbiamo che ogni ISP si collega ad altri ISP mediante *collegamenti diretti in data center dedicati*, chiamati *Internet Exchange Point*.



Osservazione. Molti dispositivi hanno *più interfacce network*. Non possono essere simultaneamente attive. (esempio: cellulare -> WiFi e dati mobile)

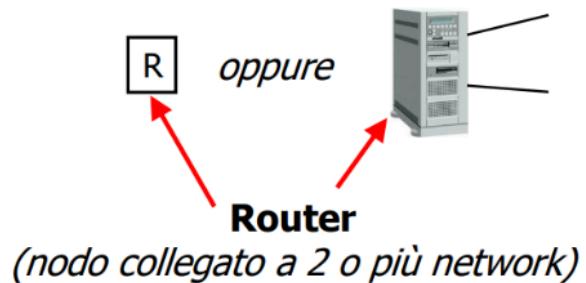
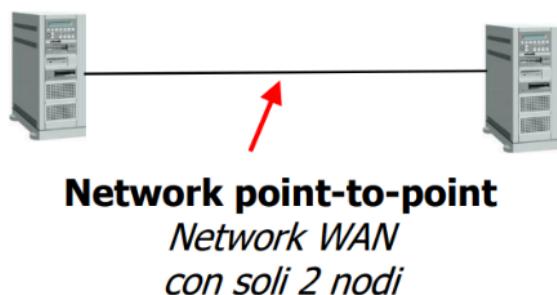
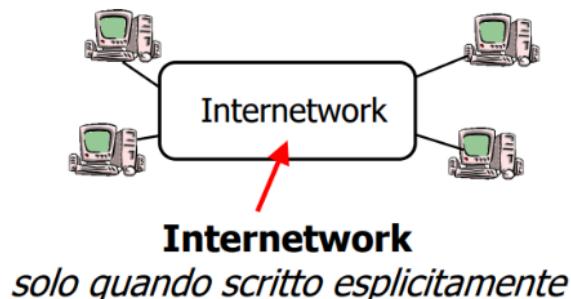
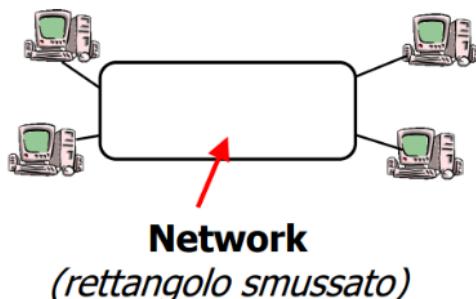
Osservazione. I contratti con gli *ISP* indicano un *throughput nominale* delle network a cui si collegano, che è (circa) la massima velocità di trasferimento dei collegamenti. Solitamente è *assimmetrico*, dove il t.n. download \gg t.n. upload

Osservazione. Storicamente, era possibile osservare la *topologia dell'Internet*, quando era davvero un modo per interconnettere *poche decine di reti* (Università, centri militari)

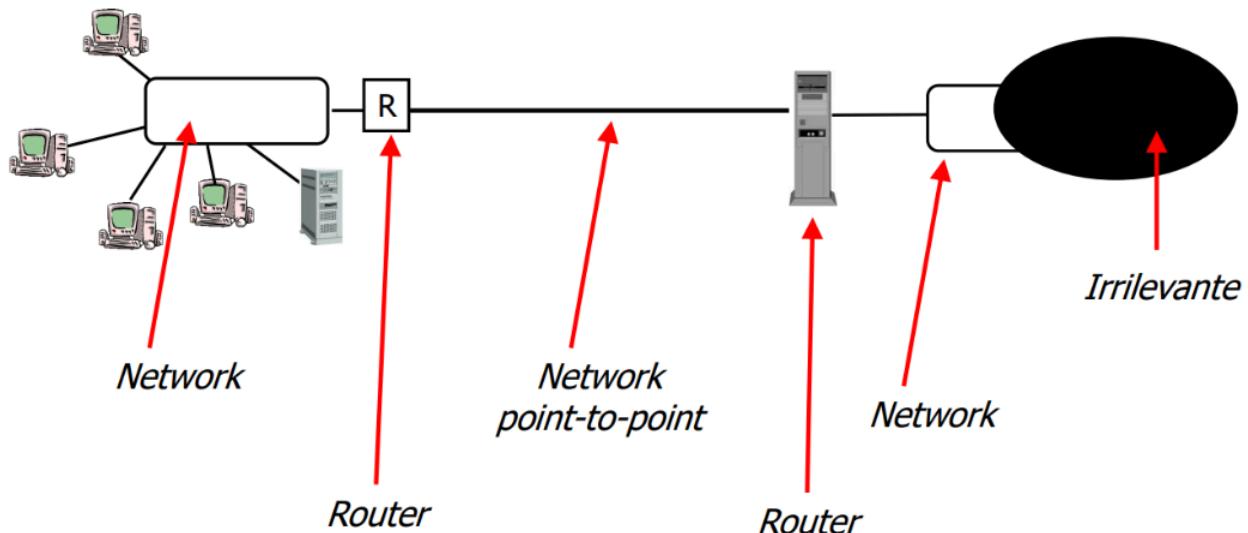


2. Notazioni Internet

Nel corso, useremo le seguenti notazioni per descrivere "*un pezzo di Internet*" (o uno schema ipotetico)



Esempio.



X

3. Inaffidabilità dell'Internet

Osservazione. L'Internet è *intrinsecamente* inaffidabile, ovvero non presenta dei meccanismi per rilevare e risolvere certi problemi: ne dovranno occuparsi i lati *endpoint*! Questo principio si chiama "*end-to-end*".

Q. Quando l'internet può perdere pacchetti?

1. Una delle network coinvolte è intrinsecamente inaffidabile
2. Un router "*decide*" di scartare dei pacchetti (ci saranno molti motivi per farlo...)

Q. Quando abbiamo dei duplicati?

1. Le network sono intrinsecamente inaffidabili e quindi le network possono duplicare frame

Q. Perché i pacchetti possono arrivare in ordine diverso?

1. Servizio *connectionless*: Ogni pacchetto viene instradato indipendentemente dagli altri pacchetti, quindi possono seguire *cammini di routing diversi* e dunque arrivare alla loro (stessa) destinazione in ordine diverso.

Network Numbers

X

*Definizione e caratterizzazione dei network number. Assunzioni (del corso) sulle network number.
Notazioni per indicare le network numbers.*

X

0. Voci correlate

- Fondamenti su Internetwork

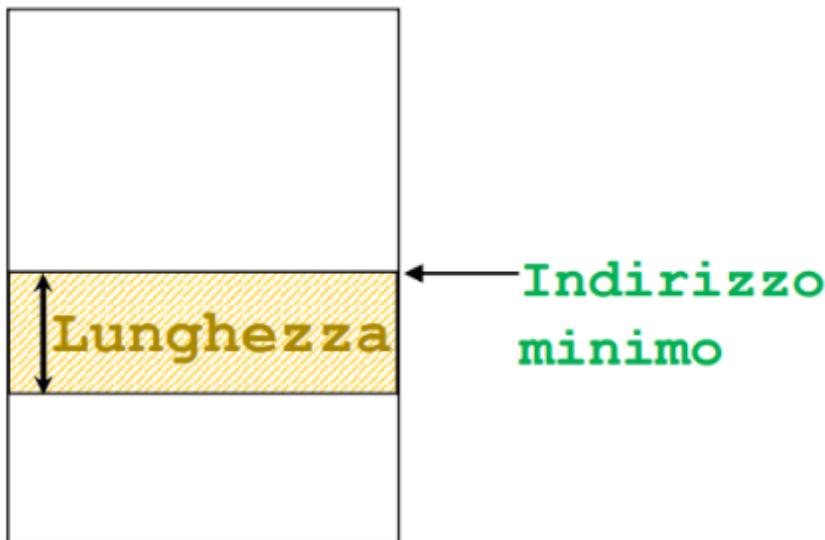
1. Definizione di Network Number

Q. Come implemento il test di appartenenza alla stessa network tra ≥ 2 indirizzi IP?

Una nozione preliminare per capirne la risposta è come segue.

Definizione. Un *Network Number* è un intervallo contiguo di indirizzi IP tali che:

- La lunghezza è un multiplo di 2, in tal caso denotiamo il multiplo con k (quindi la lunghezza è 2^k)
- Il *minimo* del network number ha k bit meno significativi tutti fissati a zero



Qual è la struttura degli indirizzi IP nello stesso network number? Si *"assomigliano"* nelle prime parti ma poi si differenziano nelle *parti meno significative*. Infatti, dato un network number (ad esempio) di lunghezza 2^4 abbiamo un generico indirizzo IP che va da xxxx.xxxx.xxxx.0000 a xxxx.xxxx.xxxx.1111.

Enunciamo quindi il seguente principio:

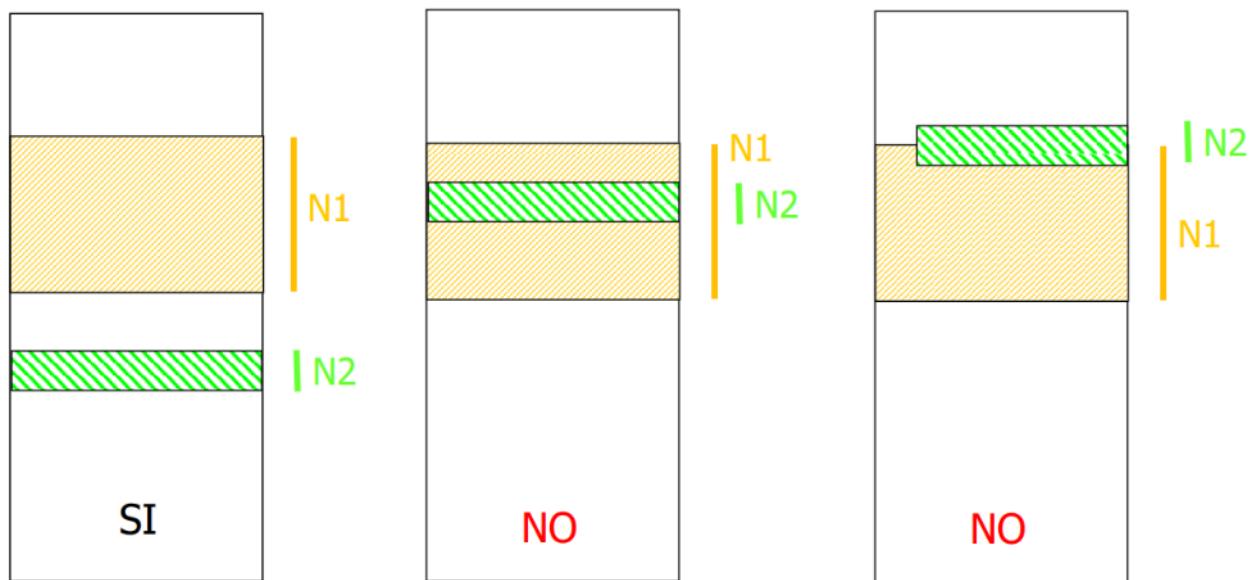
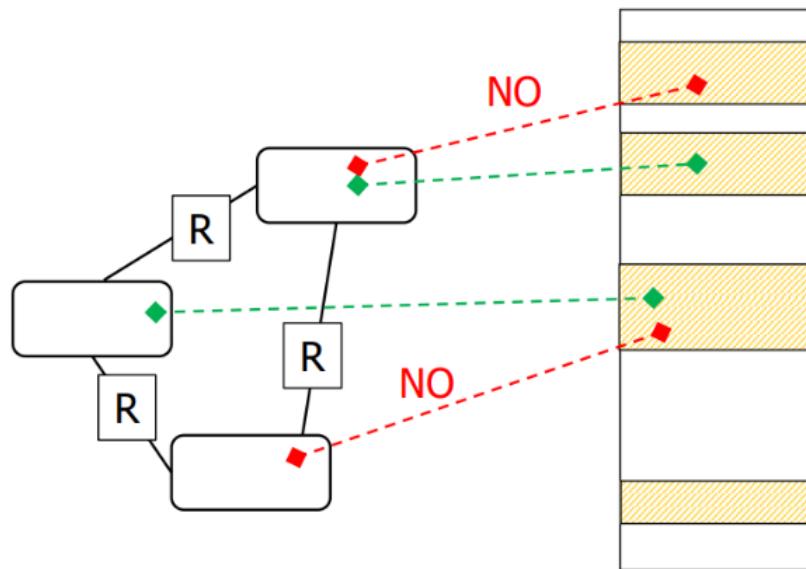
"Ogni nodo di una network si deve avere un idirizzo IP del network number del network appartenente"

X

2. Assunzioni del corso

Inoltre, poniamo un paio di assunzioni sulle *network number*.

1. Ogni *network* ha *un solo network number* ed è univoca. Quindi si ha una corrispondenza uno-ad-uno tra network number e network
2. Non ci sono sovrapposizioni tra network number



La prima assunzione ha le seguenti conseguenze:

- La size della network dipende anche dal network number *"assegnato"*

- Gli indirizzi IP non utilizzati in una network *non* verranno mai utilizzati da nessun'altro (uno "*spreco*")

La realtà delle assunzioni 1), 2) sono diverse, ma non lo vediamo nel corso.

X

3. Notazioni per le Network Numbers

Vediamo delle *notazioni* per indicare le *network numbers*.

Notazione. Una *network number* con lunghezza 2^k e con indirizzo minimo `min-addr` si indica con `min-addr/(32-k)`, ovvero la prima parte indica l'indirizzo *minimo*, la seconda parte indica *il numero dei bit "fissati"*.

Esempio. 131.0.0.0/8 indica tutti i network number compresi tra 131.0.0.0 e 131.1111.1111.1111

Osservazione. Notiamo che se la lunghezza è un *multiplo di 8* allora la network number è visualizzabile a "*occhio*", siccome la suddivisione tra "*parte fissa*" e "*parte libera*" si trova proprio su uno dei "*punti*". Vedremo successivamente che significato hanno la "*parte fissata*" e la "*parte libera*", quando vedremo il partizionamento delle network numbers.

Osservazione. Incrementare la "*lunghezza*" del network number equivale a dimezzare l'intervallo indicato dal network number. Ossia se .../24 indica un network number con 2^8 indirizzi allora .../25 indica un network number con 2^7 indirizzi.

Esercizio. 10.93.22.12/23. Qual è il range degli indirizzi IP?

Allocazione delle Network Number

X

Allocazione delle network numbers: chi e come, operazioni di routing e forensics. Concetto di network number e host number di ogni indirizzo IP. Host number non ammissibili (convenzione), numero di indirizzi IP utilizzabili per ogni network number.

X

0. Voci correlate

- Network Number

1. Chi Alloca le Network Numbers?

Q. Chi decide la suddivisione/distribuzione delle network numbers? Come viene fatta?

L'operazione aritmetica della *suddivisione* delle network numbers si chiama *subnetting* e viene fatto con un procedimento aritmetico simile alla progressione geometrica, ovvero usare il fatto che la seguente serie converge:

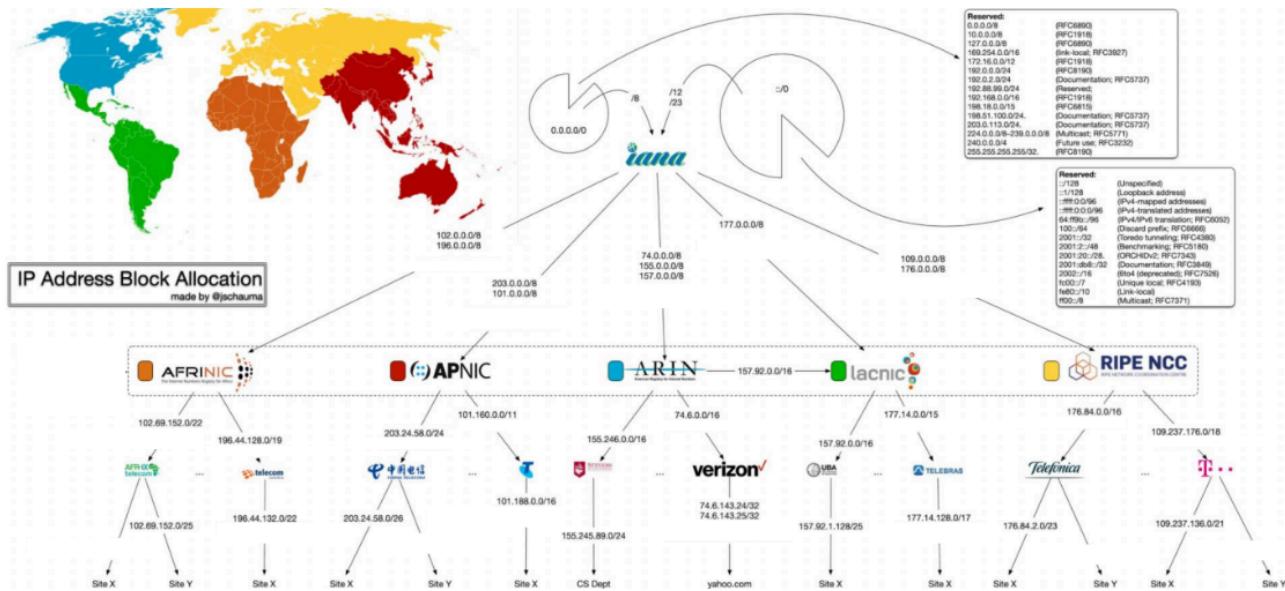
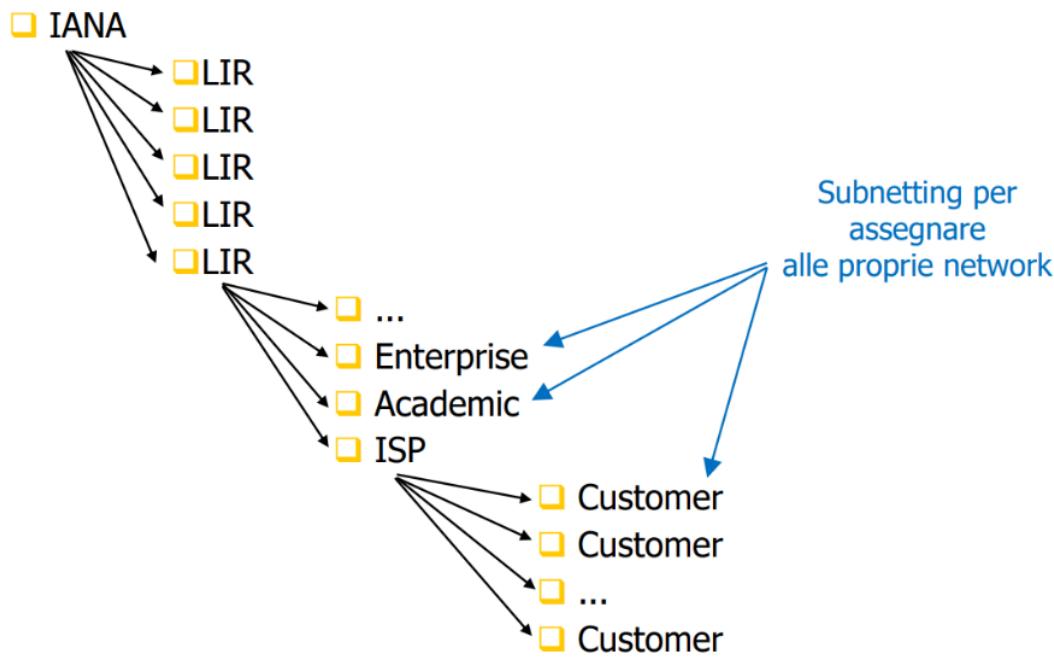
$$1 = 1 + 1/2 + 1/2 = 1 + 1/2 + 1/4 + 1/4 = \dots = \sum_i \frac{1}{2^i}$$

Vedremo dopo *bene* come viene fatto, per ora vediamo *chi* lo fa e quali sono le conseguenze sul significato delle network numbers.

IANA. Un'organizzazione "*possiede*" tutto lo spazio delle network numbers (IANA) e definisce più network numbers e assegna ognuno ad ogni "*continente*", ovvero un'*autorità continentale* che si chiama *LIR* (Local Internet Registry). Le LIR sono 5.

LIR. Dopodiché, ogni LIR-x suddivide il proprio spazio in più network numbers e li assegna ad *ORG-n*, che sono

- O enterprises
- O *istituzioni accademiche*, che effettueranno ulteriore subnetting
- O *Internet Service Providers*, che assegneranno degli indirizzi IP a dei "*customers*" (o un solo indirizzo o un "*piccolo*" network number)



Q. Come associo il network number al proprietario?

Questa è l'operazione di "*forensics*", e dipende dal livello di "*profondità*" della network number in cui si trova.

- Se è della IANA, o uno dei LIR, o una delle istituzioni/ISP/imprese assegnate dalla LIR, è possibile consultare un *servizio di database pubblico* mediante il *protocollo whois*
- Altrimenti i dati non sono pubblicamente accessibili e le suddivisioni non sono pubblicamente descritte, solo le autorità giudiziarie hanno possibilità di effettuare ulteriori indagini.

Osservazione. Le allocazioni dei network number sono gestiti in una maniera *autonoma!*

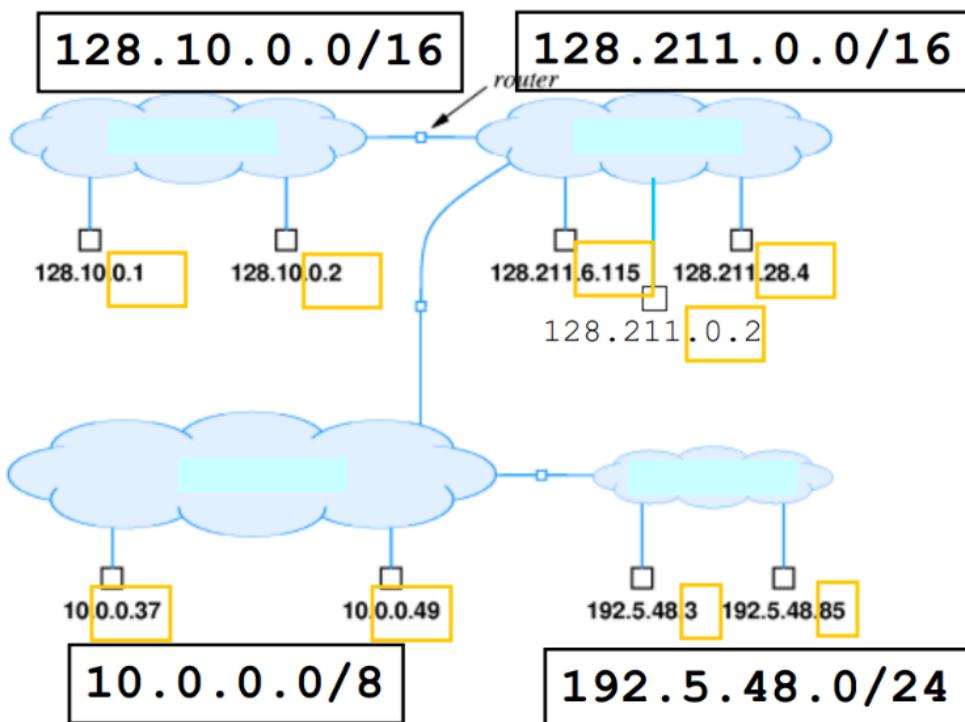
Osservazione. Nonostante ciò, *intradare* i pacchetti è comunque possibile. Infatti, forensics != routing.

2. Network Number e Host Number

Quindi, dato una *network number* $x.x.x.x/k$, abbiamo che possiamo suddividerla in due parti: una "*fissa*", ovvero i primi k bit, e un'altra "*libera*", gli ultimi k bit.

In virtù delle osservazioni fatte, possiamo dire che la prima parte è la *network number* e identifica la network, invece la seconda parte indica il *nodo* e la si chiama *host number*.

La network number è assegnata da un'autorità centrale e identificano nodi con la *stessa network*, l'*host number* viene invece assegnato da un'autorità "*locale*" e identificano i singoli nodi all'interno di una network.



Alcuni *host number* non sono assegnabili per convenzione. In particolare sono:

- Host number con tutti *zero*, siccome dovrebbe rappresentare la *network number* (?)
- Host number con tutti *uno*, siccome rappresenta il "*broadcast*" nel network

Quindi dato una network number di lunghezza x , in realtà posso assegnare 2^{32-x} indirizzi IP a nodi.

Osservazione. Quindi .../31 non esiste come host number, e .../30 è l'host number più piccolo possibile

Subnet Mask

X

IP to network number, notazione aumentata. Definizione di subnet mask, intuizione delle subnet mask. Test di appartenenza alla stessa network tra due nodi.

X

0. Voci correlate

- Network Number
- Allocazione delle Network Numbers

1. IP to Network Number

Q. Supponiamo di avere un indirizzo IP, come ricaviamo la sua network number?

Con un esempio, osserviamo immediatamente che non possiamo sapere a priori la sua network number di appartenenza. Infatti, ad esempi 131.114.9.252 potrebbe appartenere a più network number:

- 131.0.0.0/8
- 131.114.0.0/16
- Eccetera

Quindi dobbiamo "*incrementare*" la nostra notazione e trovare un modo per indicare la network number di appartenenza. Supponendo di avere la coppia (*ip-addr, netnum/k*), indichiamo *l'indirizzo IP* con *ip-addr/k*.

Possiamo comunque *differirlo* da un network number, siccome non è il *numero minimo* (che non può essere un host number!)

In realtà ci sono altre notazioni usate nel gergo tecnico che sono più ambigue, ma nel corso non le useremo.

X

2. Subnet Mask

Vogliamo realizzare finalmente il *test* per indicare se due indirizzi IP appartengono alla stessa network o meno.

Specifichiamo il *network number* al suo *network di appartenenza* indirettamente mediante una *subnet mask*. In particolare, una *subnet mask* è una sequenza particolare a 32 bit che "*inizia con tutti uno*" e a un certo punto "*conclude con tutti 0*".

Esempio. 255.255.0.0 è una subnet mask

Notiamo che quindi, dato un *indirizzo IP* e una *subnet mask*, possiamo ricavare:

- *Lunghezza*: codificata direttamente dalla subnet mask, basta contare gli uno
- *Network Number*: effettuare l'operazione bitwise IP and SUBNET_MASK
- *Host Number*: effettuare l'operazione bitwise IP and not(SUBNET_MASK)

Osservazione. Ci sono molti modi capire se una sequenza *non* è una subnet mask. Esempi:

- Avere l'ultimo byte come un numero dispari
- Altri casi evidenti, come 255.128.255.0

X

3. Test Two IPs in Netnum

Finalmente possiamo relizzare il test per determinare se un certo indirizzo appartiene alla mia stessa network o meno.

Supponendo di avere la *mia subnet mask e il mio indirizzo IP* in configurazione, ho che il predicato $P(x)$ "x sta sulla mia network number" è equivalente a

$$P(x) \iff (\text{my-IP AND my-SM}) = (\text{other-IP AND my-SM})$$

La dimostrazione è *omessa*.

Configurazione Statica e Dinamica dei Nodi su Internet

X

Configurazione dei nodi collegati ad una network. Configurazione statica, configurazione dinamica. Schema generale.

X

0. Voci correlate

- Fondamenti su Internetwork

1. Configurazione Statica e Dinamica

Q. Un nodo collegato ad una network deve avere le seguenti informazioni, per mandare un pacchetto ethernet, deve possedere le seguenti informazioni:

- Indirizzo IP sorgente (proprio)
- Indirizzo IP destinatario
- Subnet Mask
- IP Gateway
- IP del Name Server (non indispensabile nel contesto delle network/internetwork)

Come si ottengono queste informazioni? In *configurazione*. Vediamo che ci sono due metodi per acquisire le informazioni menzionate sopra.

Statico. Sono *inserite manualmente* e non cambiano mai, inoltre vengono memorizzate in memoria. I dettagli per questo tipo di configurazione dipende dal sistema operativo usato.

Dinamico. Ottiene le informazioni automaticamente quando "*si accende*" il calcolatore. In particolare, funziona che un *server di configurazione*, raggiungibile *da broadcast*, assegna tutte le informazioni necessarie. Tuttavia le informazioni fornite possono essere diverse ad ogni "*momento*" di collegamento, ossia ogni volta che "*riaccendiamo il calcolatore*" possiamo ottenere informazioni diverse.

Tipicamente, si usa la configurazione *statica* per i router, "*server*" node e i proxy; invece i "*nodi non server*" usano la configurazione dinamica.

Osservazione. Assumiamo sempre che le informazioni fornite siano sempre corrette.

Protocollo DHCP e ARP

X

Protocollo DHCP e ARP. Protocollo DHCP (semplificazione): funzionamento del protocollo, convenzioni. Protocollo ARP: come trovare indirizzi Ethernet (fisici) da indirizzi IP (logici). Idea del protocollo ARP, notazione dei frame ARP. ARP caching. Esercizio di riepilogo.

X

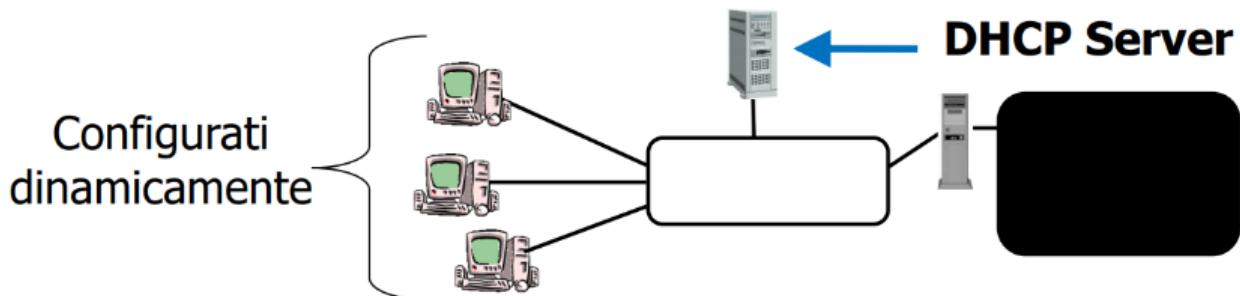
0. Voci correlate

- Fondamenti su Internetwork
- Configurazione Statica e Dinamica dei Nodi su Internet

1. Protocollo DHCP

N.B. Questo argomento è stato presentato in una maniera semplificata, con delle discrepanze col funzionamento reale del protocollo DHCP.

Nella network ho dei nodi da *configurare dinamicamente*. Col protocollo DHCP, questi nodi comunicano col *DHCP server* (ovviamente sulla stessa network) mediante dei *frame broadcast*.



Si segue quindi questo procedimento:

1. Il *client* invia un frame *DHCPDISCOVER* in broadcast ("chi è il server?")
2. Solamente il *server DHCP* ne prende nota e sceglie un *indirizzo IP libero* e crea l'associazione tra indirizzo logico e fisico
3. Il *server DHCP* invia un frame *DHCPOFFER* al client
4. Per "sempre":
 1. Il client invia un frame al DHCP per "*confermare*" che le sue informazioni siano ancora attive
 2. DHCP conferma o revoca l'allocazione

Il contenuto del frame (*payload*) viene descritto dal protocollo DHCP che non vedremo e ci limiteremo a dire che i contenuti siano *DHCPDISCOVER* e *DHCPOFFER + config. info..*

Come sempre, osserviamo che il protocollo si basa sul principio dell'onestà, ossia solo il server DHCP risponde al frame DHCPDISCOVER.

X

2. Protocollo ARP (Address Resolution Protocol)

Q. Come traduco un *indirizzo IP* in un *indirizzo Ethernet*? Ossia, come associo l'indirizzo logico all'indirizzo fisico? Questo serve per inviare *frame* nel network.

Intuitivamente devo inviare frame *al mio network*... ma come?!

IDEA. Il nodo sorgente invia un *frame broadcast* con "payload ARP", che a grossomodo vuol dire "*Chi è IP-x? Dire a ETH-src*", e risponde solo chi ha in effetti l'indirizzo IP.

Notazione. Useremo la seguente notazione per scrivere il payload ARP.

- *Request*: ETH-src, IP-src, ?, IP-dst
- *Response*: ETH-src, IP-src, ETH-dst, IP-dst

Osserviamo che la prima coppia è "*inutile*", in quanto viene ripetuta e non è indispensabile per inviare il frame di richiesta/risposta. I motivi veri ci sono, ma sono complicatissimi e non li vedremo...

Quindi in totale ho i frame composti nei seguenti modi:

ETH-dst	ETH-src	TYPE	PAYLOAD	
FF:...:FF	ETH-A	ARP	ETH-A, IP-A, ?, IP-B	# request
ETH-A	ETH-B	ARP	ETH-A, IP-A, ETH-B, IP-B	# response

Q. Se l'indirizzo IP non sta nel mio network?

Semplicemente mando il pacchetto IP al router gateway di default... ovviamente risolvo il suo indirizzo IP con ARP.

Q. Per risolvere nomi già visti prima, devo ripetere le ARP request?

Nella realtà non è necessario siccome nel sistema operativo è presente il *modulo ARP* che "*la comunicazione ARO*", ossia gestisce anche il caso in cui l'indirizzo IP non sta nel mio network o quando riceve una richiesta ARP, e tiene in memoria una *ARP cache*.

Facciamo degli approfondimenti importanti sul protocollo ARP.

GRATUITIOUS ARP. Osserviamo che nella realtà ci sono *richieste ARP* per "*risolvere il proprio indirizzo*", ossia un frame ARP con payload ETH-A, IP-A, ?, IP-A.

Quando e perché: Ogni volta che un nodo si collega/riconnella ad una network, così:

- Se capisce che c'è un altro nodo avente lo *stesso indirizzo IP*, che è un problema di configurazione.
- Rende obsolete le vecchie entry ARP; infatti, quando si invia una ARP request, tutti i nodi potrebbero prenderne nota e aggiornare la loro ARP cache salvando l'indirizzo IP del richiedente.
- Modifica la switch table

X

3. Riepilogo DHCP e ARP

Per consolidare gli argomenti visti, vediamo uno *scenario comune*.

Esercizio. Supponiamo allocazione dinamica delle informazioni di configurazione. Sia A un nodo "*appena acceso*" e manda un frame al nodo B, che sta in un network diverso. Quali sono i frame inviati da A?

Traccia: Quali informazioni ci servono? IP-dst: OK; IP-src: Non lo conosco; ETH-src: OK; ETH-dst: Non lo conosco

Quindi il procedimento dei frame che invierà A serviranno per:

- Scoprire il suo indirizzo IP, quindi inviare frame DHCPDISCOVER
- Inviare una gratuitous ARP
- Scoprire l'indirizzo IP di IP-dst, che è il gateway siccome conoscendo la subnet mask può determinare che IP-dst non appartenga alla sua network, mandando frame ARP
- Inviare il frame IP

Avendo il seguente elenco, in formato tabella.

dst	src	type	payload
FF:...:FF	ETH-A	DHCP	(DHCPDISCOVER)
ETH-A	ETH-DHCP	DHCP	(DHCPOFFER + INFO)
FF:...:FF	ETH-A	ARP	(ETH-A IP-A ? IP-A)
FF:...:FF	ETH-A	ARP	(ETH-A IP-A ? IP-GW)
ETH-A	ETH-GW	ARP	(ETH-A IP-A ETH-GW IP-GW)
ETH-GW	ETH-A	IP	(... IP-A IP-B ... PAYLOAD-IP)

N.B. Da non imparare a memoria, capire perché abbiamo svolto l'esercizio così

Network Routing

X

Network routing. Metodi di routing: routing statico e routing dinamico, idea. Notazione dei router. Idea del routing statico: routing table, algoritmo per routing statico. Generalizzazione su internetwork: azione di default della routing table. Errori comuni sul routing. Routing delle organizzazioni: come funziona.

X

0. Voci correlate

- Fondamenti su Internetwork

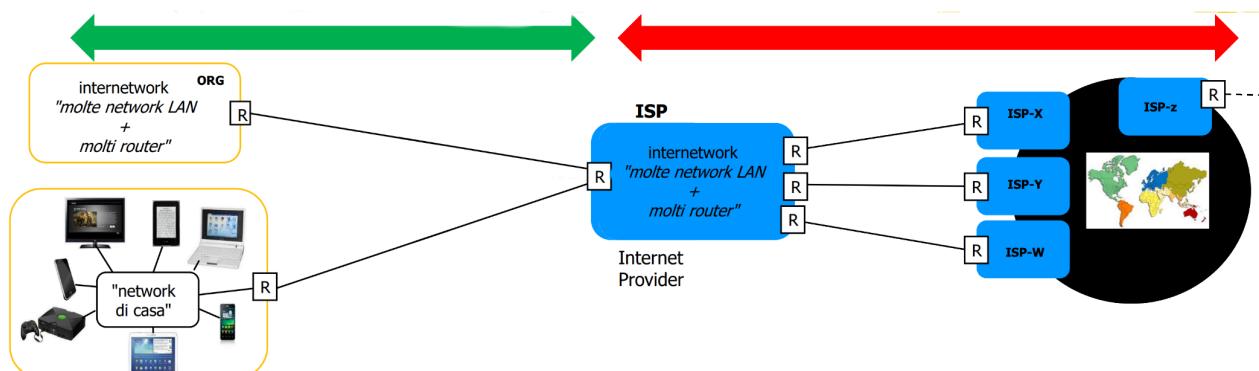
1. Routing Statico e Dinamico

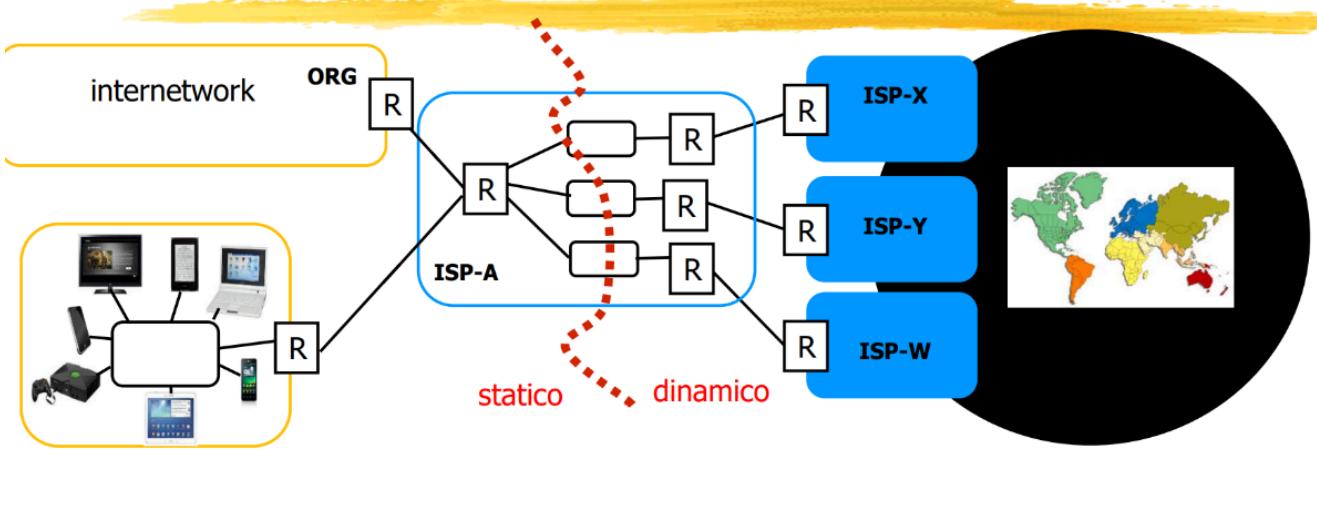
Q. Come fa un router a instradare *"correttamente"* i pacchetti?

Questo è un tema molto ampio, e comprende anche degli argomenti complicati. Per iniziare, ci sono *due* metodi per effettuare il routing:

- *Statico*: il router è configurato da qualcuno e *non cambia mai*; lo si usa quando siamo *"vicini"* agli end point
- *Dinamico*: una tabella di routing si aggiorna automaticamente (*"circa"* una switch table) comunicando con i router vicini (differenza dalle switch table) mediante protocollo BGP; lo si usa quando siamo invece *"lontani"* dagli end point; non vedremo questo tipo di routing nei dettagli.

I vantaggi del routing statico sono la *semplicità e velocità*, tuttavia non reagiscono a guasti o variazioni di traffico. D'altra parte, il routing dinamico è più *complicato e lento* ma è in grado di reagire a guasti, picchi di traffico o variazioni.





2. Routing Statico

Vediamo come un router che effettua il routing statico può scegliere il "*next hop*" (ossia il nodo successore a cui inviare il frame).

Notazione. Osserviamo che ogni router ha un'identificazione per *ogni sua interfaccia* ("lato"), ed essendo a più interfacce (per collegarsi a più network), ha più informazioni di configurazione su network; e ciò comprende gli indirizzi IP. Ogni router associa un identificatore alle proprie interfacce, e li indichiamo in forma simbolica senza ambiguità. Alcuni esempi:



Vediamo l'idea di base per scegliere il "*next hop*" nel routing statico.

Idea. Ogni router ha una *routing table statica*, composta da due colonne:

- *Network number*: Il network number di IP-dst
- *Interfaccia e Azione*: Sull'interfaccia specifica, compiere un'azione specifica. Le azioni possono essere due:
 - "*Consegno direttamente*" se il destinatario appartiene ad una delle network number del router
 - "*Instrado*" altrimenti, e lo invio al router successivo

In un certo senso, è una mappa "*network number* \mapsto *interfaccia, azione*"-

Notazione. "*consegnare*" avvolte lo si scrive come "*direct*", e "*instradare verso IP-x*" lo si scrive solo come "*IP-x*".

Quindi si ha il seguente algoritmo:

ALGORITMO. (*Routing Statico*)

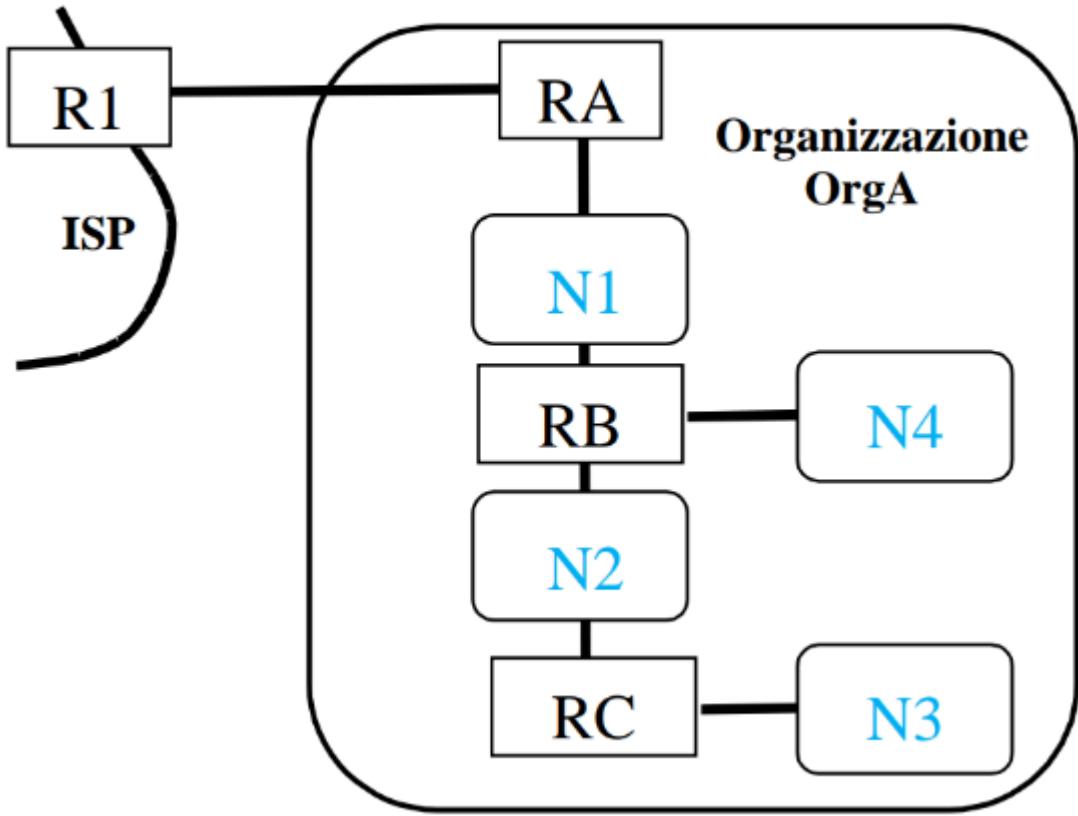
1. Per ogni pacchetto (IP-src, IP-dst, ...) da instradare:
 1. Se esiste una riga del routing table x tale che $\text{IP-dst} \in x.\text{range}$ (per fare il test basta usare le subnet mask):
 1. Eseguire $x.\text{action}$
 2. Altrimenti:
 1. *Scartare* (!)

Notiamo che in un passo successivo il passo "*scartare*" rende inoperabile la comunicazione tra la "*organizzazione interna*" e il "*mondo esterno*" (l'Internet), siccome ogni network number dev'essere descritto almeno in una riga.

L'*azione di default* è in grado di generalizzare il routing statico in un modo tale da rendere possibile comunicazione con l'internet.

IDEA. Invece di scartare pacchetti destinati a network "*esterni*" (ovvero che non appartengono alle network number che conosco), lo mando di default al *router di frontiera*. Quindi nel routing table si ha una riga composta da (default, IP-R') dove R' è il router successivo per poter raggiungere al router di frontiera (per poter "*uscire*" dall'organizzazione).

Esercizio. Creare delle tabelle di routing per la seguente organizzazione:



Esercizio. Supponiamo che il router RC riceve un pacchetto destinato a IP-A-N4. Cosa fa? Scrivere il frame che andrà a inviare.

Traccia: lo instrada verso IP-RB-DOWN, il frame sarà ARP(IP-RB-DOWN), ETH-RC-UP, IP, (pacchetto). Nello svolgimento specificare anche come si effettua la risoluzione ARP.

Consiglio. Partire dal caso simbolico ed eventualmente (se necessario) ricondursi al caso numerico.

Osservazione. Nelle tabelle di routing si ha solo una visione parziale del *network*, conosce solo la "*direzione*" della destinazione. Non conosce MAI il cammino completo (nel caso di "*network lontane*")!

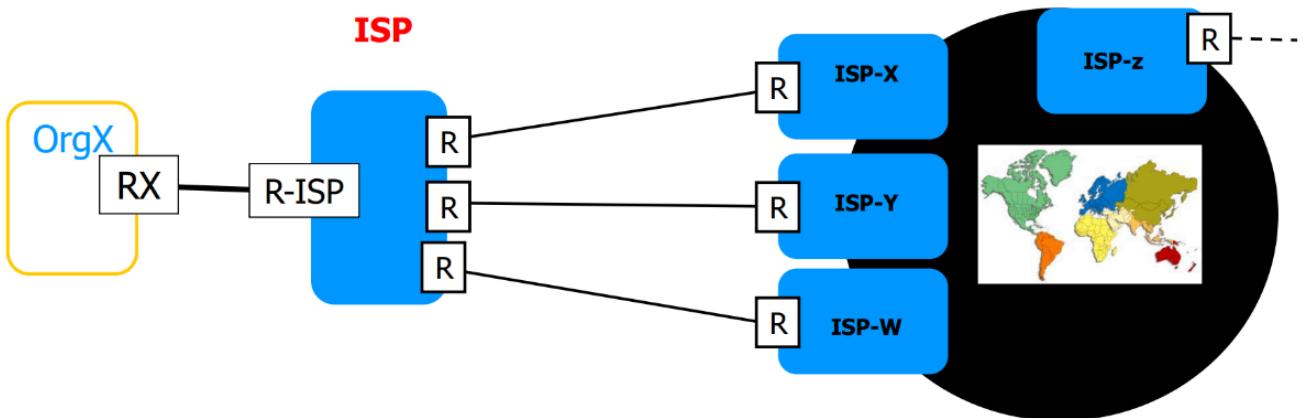
ERRORI COMUNI. (*Ocio!*)

1. Omettere la riga di default; come fa a comunicare con l'Internet?
2. Indicare indirizzi IP non direttamente connessi col router mediante network; come fa ad inviare le richieste ARP? (l'osservazione di prima)
3. Indicare "*direct*" per l'invio del pacchetto al router di frontiera; in questo modo manderei il pacchetto *a tutte le calcolatrici collegate su Internet*.

3. Routing delle Organizzazioni

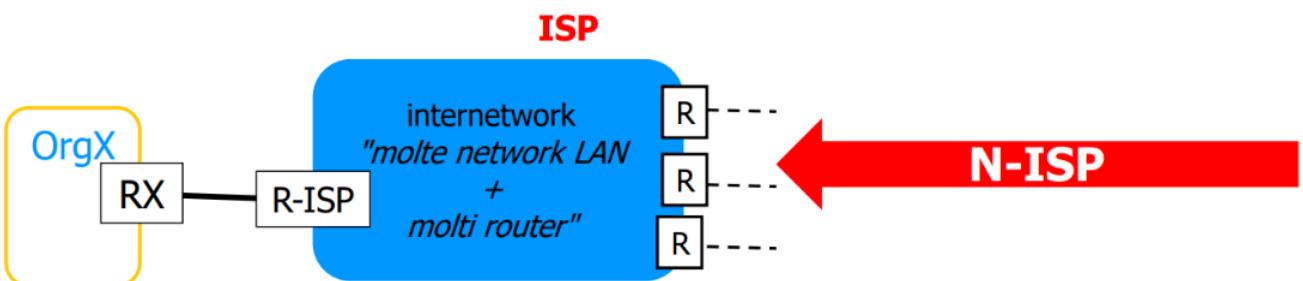
Vedremo il *procedimento operativo* per suddividere le network number. Tuttavia, ci sofferiamo *come* funziona il routing sulle organizzazioni.

Abbiamo il seguente scenario: un'organizzazione ORG desidera collegarsi ad internet, e quindi acquista un network number N-ORGx da un'ISP (che è contenuto in N-ISPs). Dopodiché, l'ISP ha ulteriori router che si "collegano all'Internet"



1. *Network di collegamento tra RX e R-ISP*: è un collegamento "*point to point*", ed è una network a tutti gli effetti. Quindi deve avere una network number. In linea di principio, può essere o a carico di ORG o a carico di ISP; ovvero, o è *interno* a N-ORGX o è aggiuntivo. Nel nostro caso, considereremo sempre questa network a *carico di ISP*.
2. *Resto dell'internet*: Il collegamento di ORGx è irrilevante all'Internet, non ha necessità di informare nessuno!
3. *ISP*: Deve modificare le tabelle di routing dei propri router, in un modo tale da instradare N-ORGX verso IP-RX-Inbound, invece di scartarla.
4. *ORGx*: Può suddividere il proprio network in più network col procedimento di *subnetting*. Si usa il *routing statico*, e le regole "*otherwise*" instradano verso ISP-R-ISP-Inbound

Notiamo che nessuno è informato del subnetting di ORGx, è irrilevante dal punto di vista esterno. Per l'ISP c'è solo un network number (N-ORGx) e per l'Internet c'è un solo network number (N-ISPs).



3.

Subnetting

X

Subnetting in termini di operazione aritmetica. Osservazione sui numeri in base 2: partizionamento in metà inferiore e superiore di un numero a n bit. Problema di subnetting: calcolare la dimensione del blocco di network number da comprare aver avere m network con n_1, \dots, n_m nodi. Procedimento errato e corretto.

X

0. Voci correlate

- Allocazione delle Network Numbers
- Network Routing

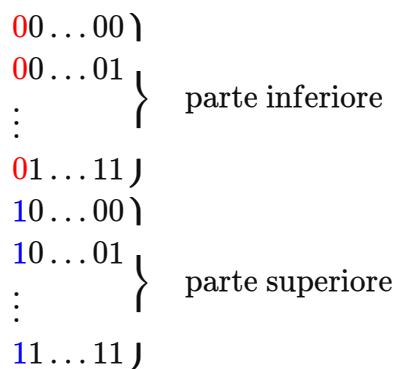
1. Subnetting

Q. Supponiamo di comprare un network number e li vogliamo "*distribuire*" tra tre network; come li suddivido in una "*maniera equa*"?

Il *subnetting* ha come obiettivo rispondere alla domanda di prima, in particolare andrà a creare *quattro suddivisioni del network number*; tre di queste parti andranno alle network, una sarà inutilizzata. Vediamo come si fa facendo la seguente *premessa matematica*.

Osservazione. Data una sequenza di N bit, si ha che n appartiene alla metà superiore di tutti i numeri possibili rappresentabili *se e solo se* il bit più significativo è impostato a 1.

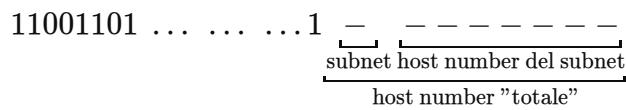
Abbiamo quindi lo seguente schema:



Analogamente possiamo creare un sottopartizionamento della parte inferiore fissando il *secondo bit significativo*, creando quindi altre due partizioni (se lo faccio anche dalla parte superiore), andando così via...

In un certo senso, i primi k -esimi bit *fissati* e *immutabili* definiscono il *subnet*, invece gli ultimi bit liberi formano degli *host number* nel network number partizionato.

Esempio. Supponiamo di avere una network number del tipo xxx/23, ossia gli ultimi 9 bit sono liberi. Per fare il subnetting, fisso ulteriori k bit degli ultimi 9 bit e il resto sono gli host number effettivi. Uno schema è del tipo



Il proprietario del network può quindi suddividere come vuole il suo network number, fissando i primi k bit più significativi a disposizione. Notiamo che è pressapoco lo stesso procedimento che fanno le organizzazioni a livello mondiale.

Esercizio. Dato il network number 200.23.16.0/20, dire:

- Il numero di indirizzi disponibili e usufruibili
- Suddividere la network number in:
 - 4 blocchi della stessa dimensione
 - 8 blocchi della stessa dimensione
 - 2 blocchi di 1/4 e 4 blocchi di 1/8
- Al riferimento di esercizio di prima, calcolare l'indirizzo IP minimo di ogni suddivisione

Traccia: Vediamo solo l'ultima richiesta. Prima dividiamo in 4 blocchi da 1/4 fissando i primi 2 bit, avendo così i 2 blocchi da 1/4. Dopodiche per i restanti blocchi, li dimezziamo di nuovo fissando il terzo bit ottenendo dunque 4 blocchi di 1/8. Rimane al lettore da svolgere effettivamente l'esercizio, creando una tabella di suddivisioni.

Vediamo un altro aspetto pratico del subnetting.

Q. Supponiamo di volere m network, in cui ognuna ha n_1, \dots, n_m nodi. Di quale dimensione devo comprare il network number?

Procedimento errato: Sommare n_1, \dots, n_m e arrotondarlo alla potenza di 2 più vicina (in alto), e comprarsi una network number di quel esempio.

E' un procedimento errato, in quanto non garantisce che la dimensione fornita sia sufficiente. Per esercizio dimostrare che non funziona per $m = 2$ e $n_1 = 520$ e $n_2 = 270$ (intuitivamente, prendo numeri "*vicini*" a potenze di due dal basso).

Procedimento corretto: Arrotondare n_1, \dots, n_m alle potenze di due più vicine e denotarle con $\hat{n}_1, \dots, \hat{n}_m$ e poi sommare le potenze di due e arrotondare la somma alla potenza più vicina di due.

Usando lo stesso controesempio di prima, vedremo che funziona.

Osservazione. Il problema dello *spreco* degli indirizzi IP è purtroppo inevitabile... in particolare è frequente quando abbiamo network point-to-point (4 indirizzi /30).

SEZIONE C. ULTERIORI ASPETTI DELL'INTERNET

Indirizzi IP Pubblici e Privati

X

Fondamenta sui collegamenti residenziali. Nuovo principio: differenza tra indirizzi IP privati e pubblici, i 4 intervalli di indirizzi privati. Definizione intuitiva di indirizzo IP privato, differenza dagli indirizzi IP pubblici. Struttura di assegnazione di indirizzi IP privati e pubblici. Conseguenze del principio. Motivazioni. Modulo NAT per collegamento diretto dai nodi con indirizzo IP privato (cenni). Osservazioni varie e curiosità storiche.

X

0. Voci correlate

- Fondamenti su Internetwork

1. Indirizzi IP Privati

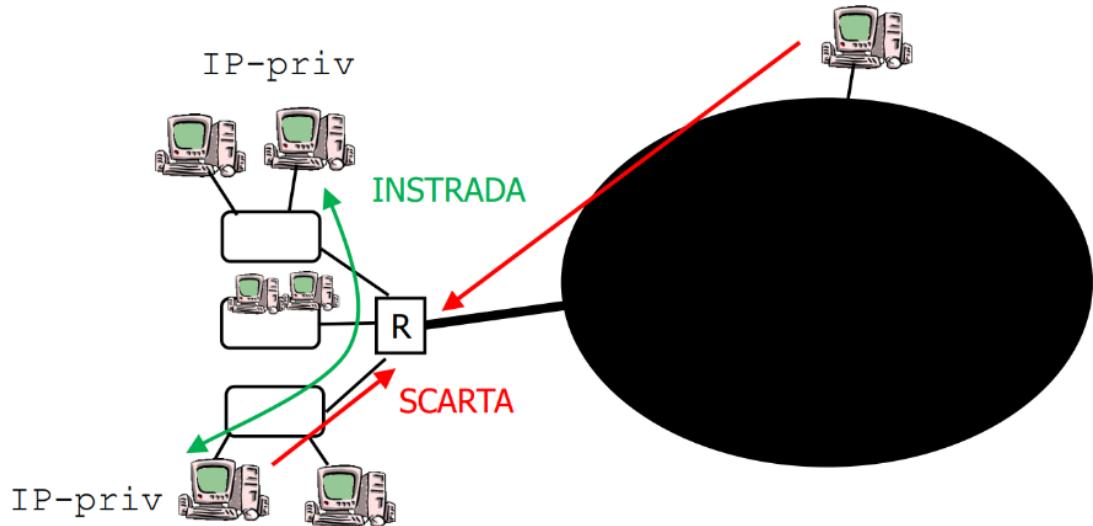
Fin'ora avevamo dato per scontato che tutti gli indirizzi IP fossero "*pubblici*", ossia tutti avessero un indirizzo IP univoco in *tutto Internet*. In un certo senso, davamo per scontato che fossero "*equivalenti*".

Nella realtà, questo non accade e bisogna fare la distinzione tra *indirizzi IP privati* e *indirizzi IP pubblici*.

Nel protocollo IP sono stati designati *quattro intervalli* degli indirizzi IP (network numbers, diciamo) per contenere gli "*indirizzi privati*", e sono:

- 10.0.0.0/8
- 172.16.0.0/12
- *192.168.0.0/16* (!, l'unico da conoscere per cultura)
- 224.0.0.0/4: Indirizzi IP per multicast (*non vedremo*)

Intuitivamente, un indirizzo IP è un indirizzo per cui ogni *router di frontiera deve scartare pacchetti* provenienti da indirizzi IP privati o destinati verso indirizzi IP pubblici. In un certo senso, avere un indirizzo privato vuol dire poter usarlo solo all'interno dell'*organizzazione* (o network, o internetwork).

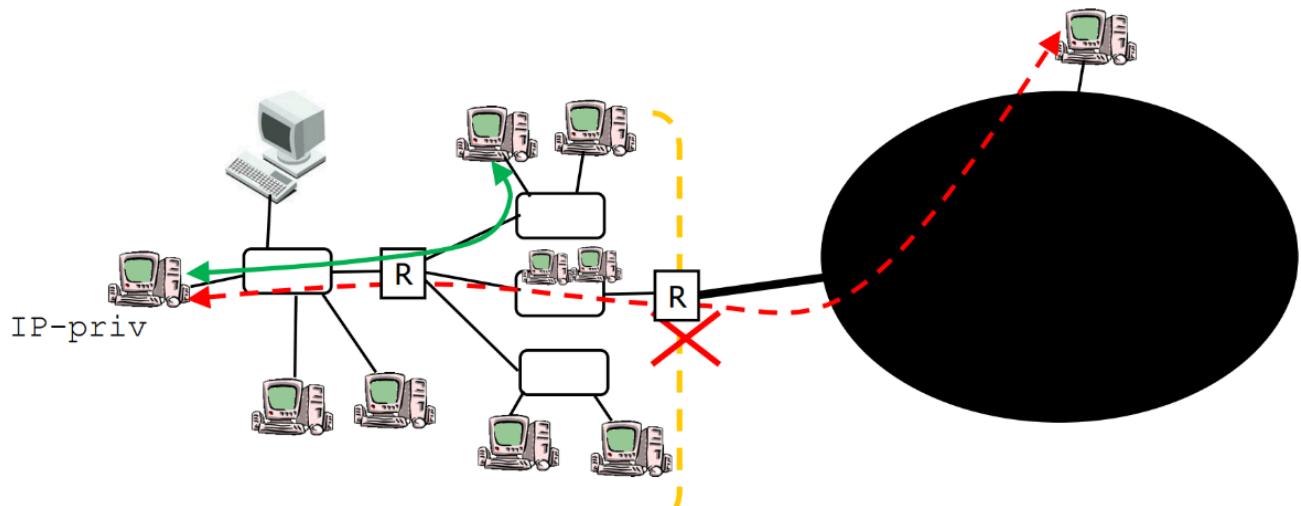


X

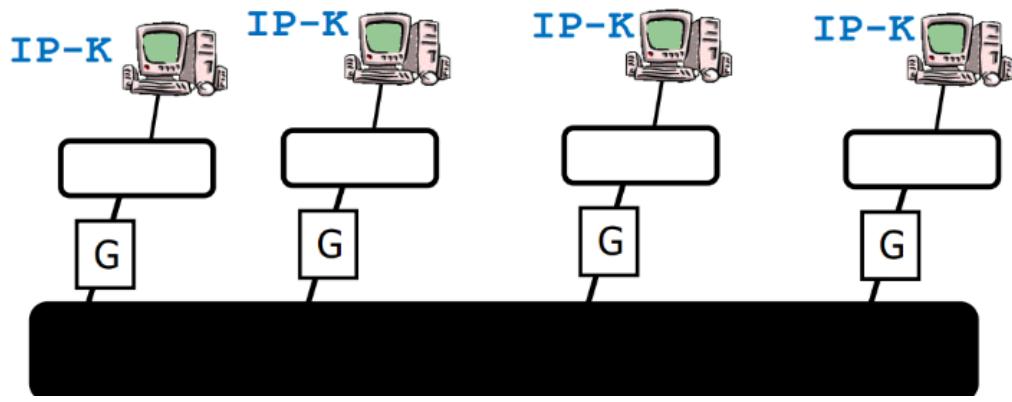
2. Conseguenze degli Indirizzi IP Privati

Il principio degli *indirizzi IP privati* fa crollare molte certezze, portandosi con sè molte conseguenze. Vediamo le conseguenze più fondamentali:

1. *Nodi con indirizzi IP privati non hanno più connettività col "resto del mondo"*: Sarà in grado solo di comunicare con i nodi all'interno dell'organizzazione ("ambiente"), ossia tutti i nodi per cui non passa il router di frontiera in mezzo

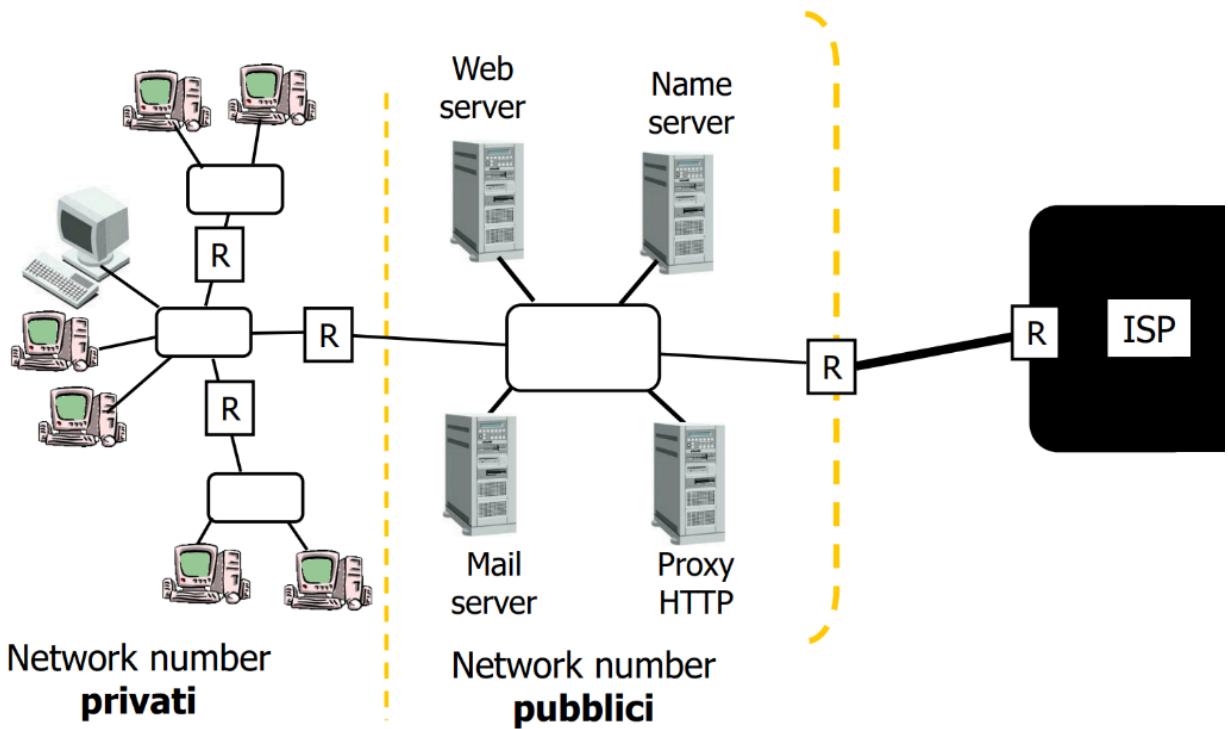


2. *Gli Indirizzi IP privati non vanno acquistati e informati*: Essendo gli indirizzi IP privati usufruibili solo all'interno delle organizzazioni e non *tra* "organizzazioni", l'assegnazione degli indirizzi IP privati *non sono da informare*, quindi neanche da acquistare. Al contrario, l'assegnazione degli indirizzi IP pubblici vanno informati e acquistati da ISP.
3. Ogni *network number* può comprendere o solo *indirizzi IP privati* o *indirizzi IP pubblici*.
4. Molte organizzazioni *diverse* possono usare gli stessi *network number* per indirizzi IP privati. Gli indirizzi IP privati sono quindi univoci *solo a livello della "organizzazione"*.



Riflettendo sulle conseguenze (in particolare 1., 2. e 3.), si evince che in un'organizzazione si effettua tipicamente l'assegnazione di indirizzi IP pubblici/privati, a seconda della *necessità* di comunicare con Internet. Ossia:

- Endpoint "*user*": privati
- Endpoint "*server*" (come mail client, DNS, eccetera...): pubblici



3. Motivazioni degli Indirizzi IP privati

Q. Perché effettuare la distinzione tra indirizzi IP privati dagli indirizzi IP pubblici?

Tanti motivi di natura pratica o tecnica:

- Si rende più *semplice* ed *economico* gestire le *network number* (il vero motivo "originale", da un punto di vista storico)
- Molto meno nodi sono visibili al *pubblico*; quindi abbiamo meno rischi di attacchi (il vero motivo "odierno")

- La stragrande maggioranza delle app funziona senza dover "parlare con Internet". Infatti:
 - Mail: contatto il *mail server* interno all'organizzazione
 - Web: contatto il *proxy*
 - Name Resolution: contatto il *DNS interno*

Quindi se nella mia organizzazione ci sono già il mail server, il proxy, il DNS interno allora ogni applicazione non dovrebbe avere necessità di comunicare con Internet.

X

4. Cenni al NAT

Q. Se un nodo con indirizzo IP privato, per qualche motivo, avesse bisogno di comunicare col resto del mondo?

Esempio. Navigazione web senza proxy, o collegamenti residenziali (vedremo bene dopo)

La *tecnologia NAT* (Network Address Translation) permette la comunicazione con Internet in queste casistiche. La vediamo solo da un *punto di vista "funzionale"* (ossia, *"come si usa?"*) e vediamo i seguenti 3 aspetti:

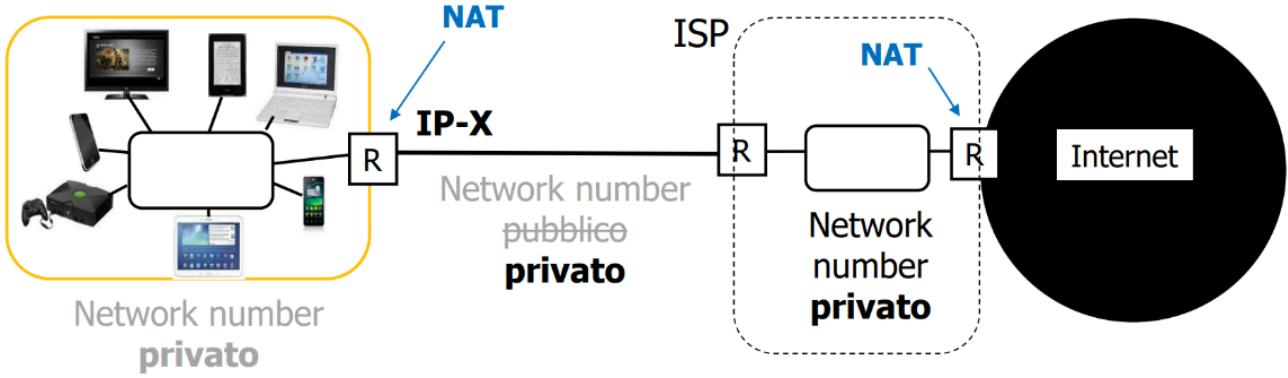
1. Ogni *organizzazione* deve avere un "*dispositivo NAT*" sul router di frontiera
2. Tutti i nodi dell'organizzazione hanno "*lo stesso indirizzo IP*" dall'esterno
3. Le comunicazioni *uscenti* non necessitano configurazione, viene gestito dal router; le comunicazioni *entranti* necessitano invece di configurazione, per poter essere "*raggiungibile*".

Gli aspetti *implementativi* ("come funziona?") del NAT sono oggetto del corso "*Reti di Calcolatori 2*" del prof. Martino Trevisan (il bro 😎).

Osservazione. Nella comunicazioni client-server (client è il nodo con indirizzo IP privato), il NAT non necessita di configurazione. Invece, se abbiamo *applicativi server* allora sarà necessario configurare il NAT

Osservazione. Un nodo con indirizzo IP privato non sa qual è il suo indirizzo IP visibile dall'esterno

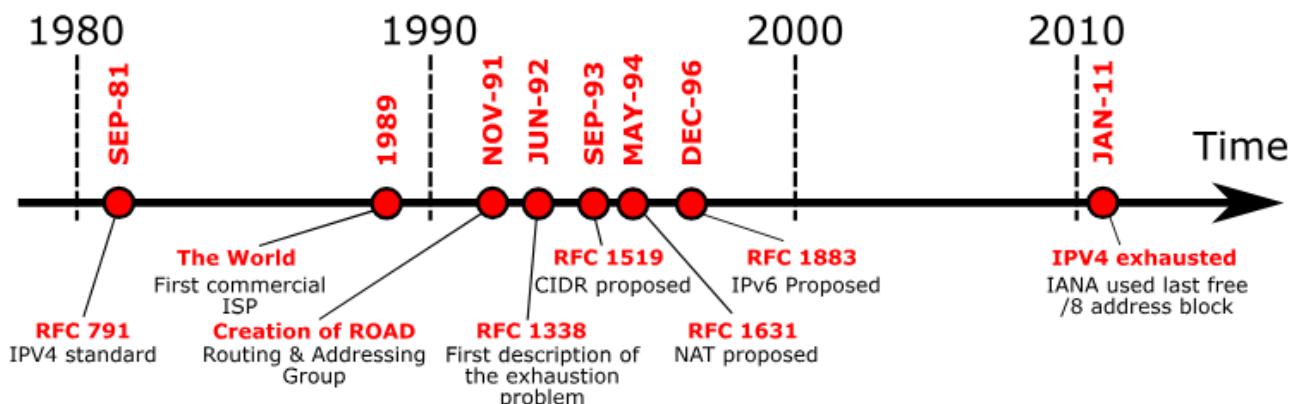
Osservazione. Spesso gli ISP assegnano indirizzi IP privati, con un ulteriore modulo NAT sui suoi router. Ciò crea ulteriori complicazioni...



Curiosità Storiche.

Nel 1991, gli indirizzi IP pubblici si stavano consumando *molto rapidamente*; quindi iniziarono le preoccupazioni e le discussioni, sviluppando il modello IPv6 nel 1993. Tutti si attesero una diffusione relativa rapida di IPv6, tuttavia sono passati circa 25 anni e sono usati "poco", siccome gli indirizzi IPv4 si erano depletati solo nel 2010.

Cosa è successo? In concorrenza si sviluppo il framework del NAT tra il 1993 e 1994, rallentando significativamente il *consumo di indirizzi IP pubblici*. Si preferì di usare quindi il NAT.



Network Point-To-Point

X

Network Point to Point (fondamenta): casi d'uso, tech stack delle network point to point. Mezzi fisici, protocollo PPP e struttura dei frame.

X

0. Voci correlate

- Fondamenti su Internetwork
- Fondamenti su Network
- Modello Fisico dell'Internet

1. Network Point to Point

I collegamenti "*point to point*" sono delle *network WAN* che collega solo *due nodi*.

Esempio. Collegamenti con ISP

Il protocollo IP richiede che ogni *network abbia una network number*, quindi il *network point-to-point* deve avere *network number* e *indirizzo IP alle estremità*. Quali tipi di network number vanno bene?

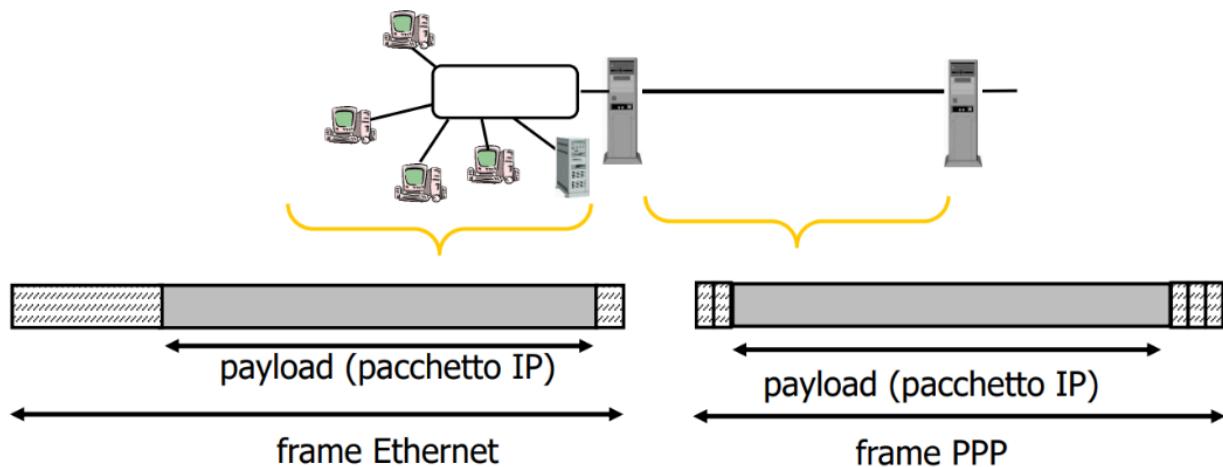
- Network number/31: *NO!* Ricordiamo che gli indirizzi host con tutti uno o zero non sono usufruibili da parte di nodi
- Network number/30: Due host number usufruibili, 01 e 10.
- Network number/29: Abiamo ancora più host number, ma sarebbe uno "spreco" di indirizzo IP...

Si evince che idealmente un *network number di una network point-to-point* sia di tipo /30.

Tecnologia Network PTP. Vediamo addesso la tecnologia per implementare le network point-to-point, quindi le sue proprietà, mezzi fisici, e protocolli.

- **Proprietà.** Una network point-to-point è tipicamente *connectionless, message-oriented* e *unreliable*.
- **Mezzi Fisici.** Molti mezzi fisici, tra cui linee telefoniche, linee dedicate o fibre ottiche
- **Formato Frame.** Il protocollo più comune per i *frame di network point-to-point* è il protocollo PPP. Ogni frame è formato da *due byte + payload IP+ 3 byte*, con MTU di 1500 byte. In particolare:
 - Il primo byte è **7E** e serve per indicare l'inizio del frame
 - Il secondo byte è **21**, indica che il frame è protocollo IP
 - L'ultimo byte è lo stesso del primo, indica la fine
 - Gli altri byte servono per motivazioni di natura elettrica (*checksum*)

Osservazione. Nel frame PPP non si indica (ovviamente) gli indirizzi IP del destinatario o del mittente.



X

Internet at Home: fondamenti sui collegamenti residenziali. Dispositivo "router"/"modem": ruoli. Schema logico di un "modem", scenario tipico (NAT + DHCP). Osservazioni sugli "hot spot" dei dispositivi mobile. Osservazione: ogni casa è una "organizzazione" (spazio di indirizzamento) autonoma/o. Osservazione: stratificazione di NAT. Nota importante: reti Wi-Fi pubblici, pericoli.

X

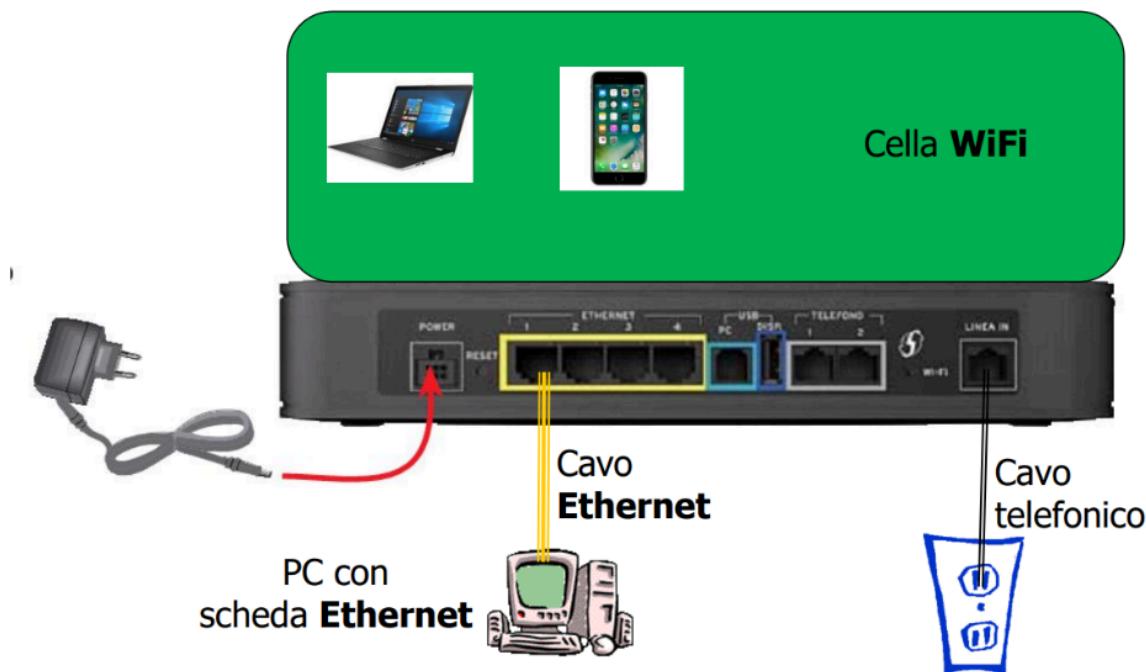
0. Voci correlate

- Indirizzi IP Pubblici e Privati
- Network Point to Point

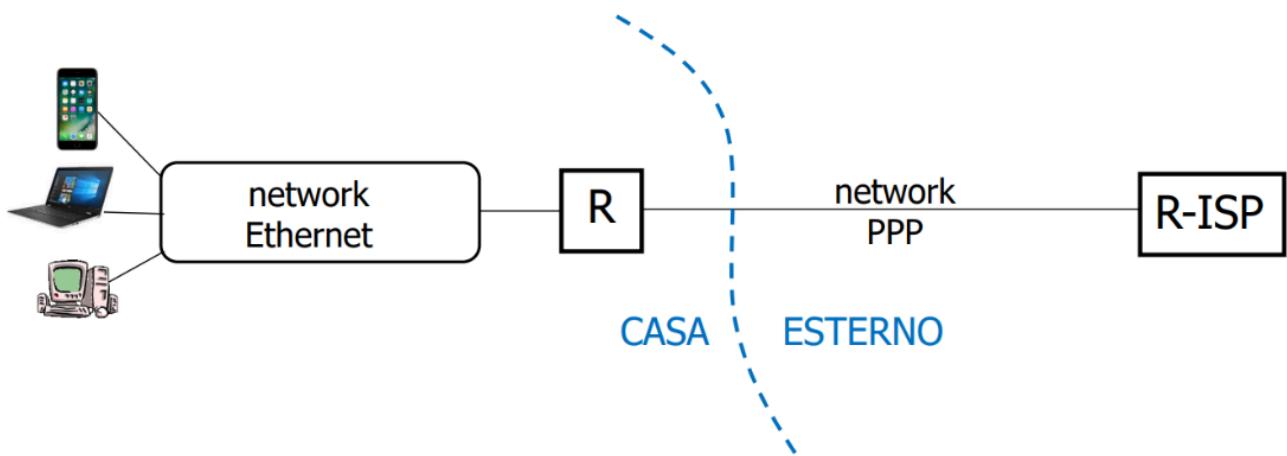
1. Internet at Home: Il "modem"

Nella maggioranza dei casi, a casa si usa un *dispositivo* (uno "scatolotto") che ha molte funzioni, diverse tra loro. Nel gergo comune la si riferisce come *"router"*, *"access point"* o *"modem"* (tutti termini tecnicamente inappropriati...). Essa compie le seguenti funzioni:

- *Router*
- *Ethernet Switch*
- *Ethernet Access Point*
- *NAT*
- *DHCP*
- *Firewall*
- *Web server, per configurazione*

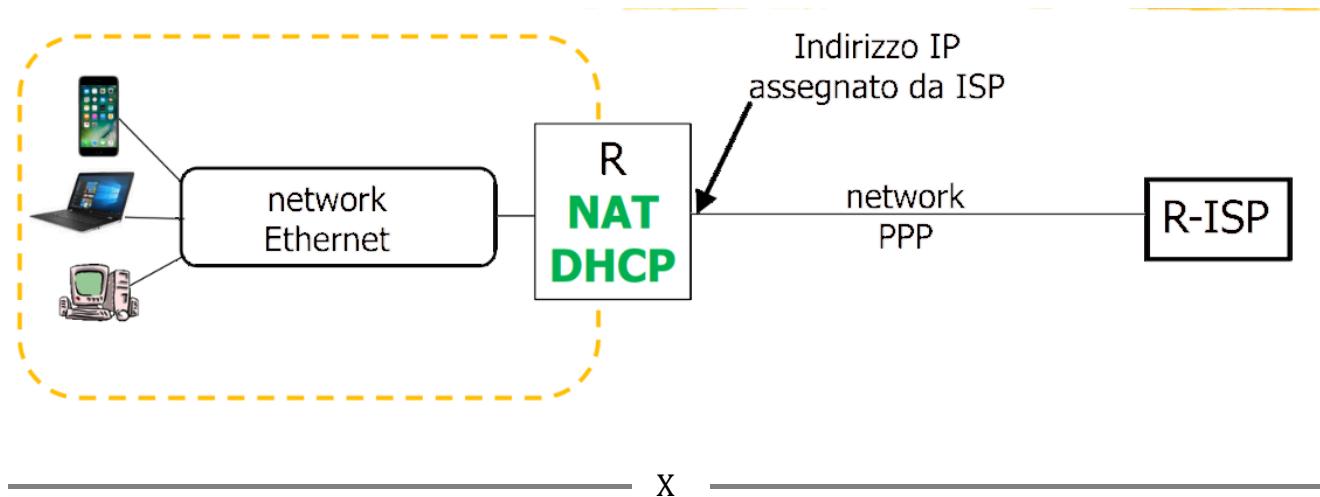


Da un punto di vista logico, abbiamo quindi un *router* che gestisce i collegamenti *all'interno della casa*, ed essa comunica con il router dell'ISP mediante una network PPP.



L'assegnazione degli indirizzi IP viene fatta *in maniera dinamica*. In particolare, dal lato ISP si ha che R-ISP è anche un server DHCP che assegna un indirizzo IP al *router*.

Di conseguenza si ha *solo un indirizzo* IP... come sarebbe possibile collegare più dispositivi? Lo si fa usando l'interfaccia *NAT + DHCP* all'interno del router! Notiamo che quindi la gestione degli indirizzi IP privati è *autonoma*, ossia l'ISP non ne è a conoscenza.

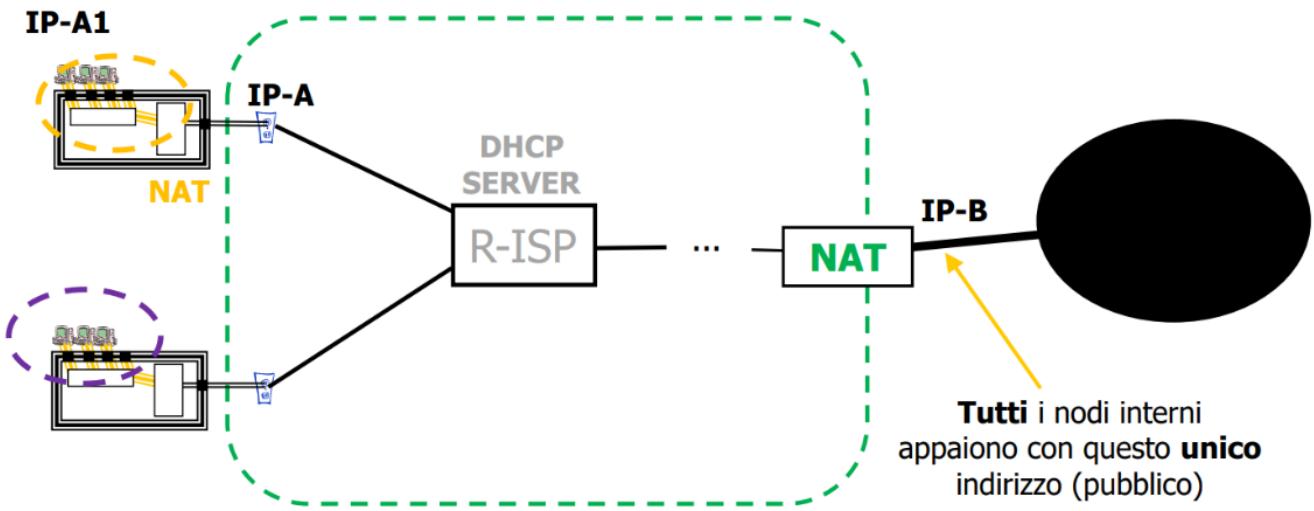


2. Osservazione sui Collegamenti Residenziali

Osservazione. (*Hot Spot*)

Alcuni dispositivi sono muniti di funzionalità "*hot spot*", ovvero possono disporre di un "*nuovo network ethernet Wi-Fi*" a cui poter collegarsi. Questa funzionalità è simile a quanto visto prima, solo che il *dispositivo stesso* ha un modulo NAT+DHCP per permettere i collegamenti. Il provider non ne è a conoscenza di questo collegamento.

Osservazione. In un certo senso, ogni "*casa*" è uno spazio di indirizzamento autonomo (NAT). Ma dall'esterno si hanno *indirizzi IP univoci* dal punto di vista dell'ISP!



Osservazione. Di conseguenza, collegarsi ad una "*rete WiFi di casa*" vuol dire ottenere un indirizzo IP dal DHCP server "*di casa*", e se inviamo pacchetti dall'esterno sembrerà che siano inviati dalla "*casa*". Quindi è importante nella pratica configurare WiFi con *autenticazione*, altrimenti una qualsiasi persona può collegarsi nella stessa cella e *osservare o modificare* il traffico, o trasmettere "*dati sospetosi*"...

Firewall: motivazione, esempi storici, principio fondamentale. Definizione di Firewall, dove si collocano i Firewall. Formato (intuitivo) delle regole Firewall, algoritmo Firewall. Osservazione: le connessioni TCP sono bidirezionali, conseguenze sulle regole Firewall. Notazione delle regole Firewall, terminologia "traffico entrante" e "traffico uscente". Esempi (esercizi). Considerazione sui "firewall veri".

0. Voci correlate

- Comunicazione tra Processi
- Fondamenti su Internetwork
- Proxy HTTP

1. Motivazioni Firewall

Nella Cybersecurity, alcune **vulnerabilità** (errori nei programmi che possono essere sfruttati per effettuare attacchi) permettono a degli attaccanti di **installare programmi** e fare ciò che vogliono. Questa vulnerabilità si chiama "**RCE**".

In particolare, l'attaccante può **costruire un messaggio** contenente un programma P (che sfrutta la vulnerabilità RCE), trovare un server con quella vulnerabilità e poi inviare il messaggio e "**infettare**" il server. Ancora peggio, può creare un "**for loop**" sugli indirizzi IP su Internet e scansionare quindi **tutti gli indirizzi IP** e infettarli tutti (tutti i server pubblici sono a rischio). Ancora molto peggio, il programma può "**autopropagarsi**", ossia il programma stesso effettua la scansione (in questo caso, il programma è un "**virus worm**"); quindi anche i nodi interni alle organizzazioni sono a rischio...

Facciamo un paio di esempi concreti:

- SQL slammer (2007) infettò 75.000 vittime nei primi 10 minuti di esecuzione
- Il ransomware WannaCry (maggio 2017) è uno dei casi più notevoli di sempre e infettò oltre 200.000 virus nelle prime 12 ore (e poi il 3/4 dell'Internet a fine giornata), causando un danno di ordini di grandezze di miliardi di dollari. Il virus sfruttò una vulnerabilità dei programmi eseguiti su porta 445 (SMB).

Q. E' necessario che i **nodi server** siano accessibili da "**qualunque parte**" dell'Internet (mondo)?

Con la teoria vista, sì. Tuttavia, mediante questi esempi diventa chiaro che abbiamo il seguente principio:

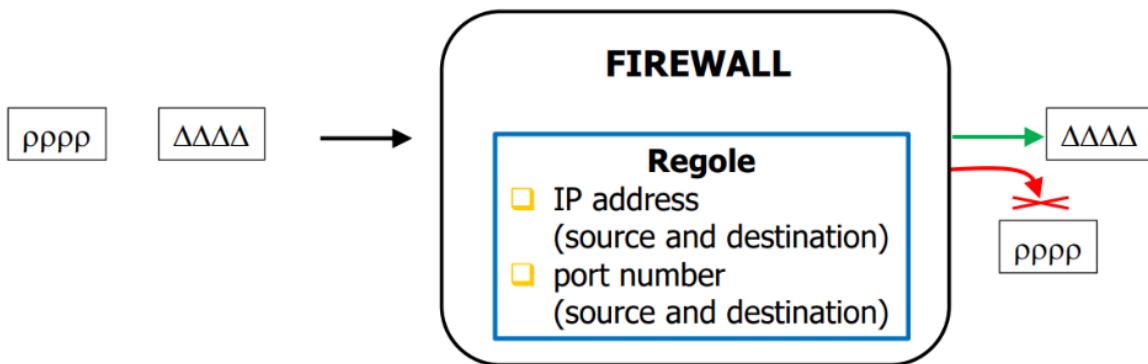
"Nessuna organizzazione del mondo permette tutto il traffico interno-esterno, e anche interno-intero in certi casi, in quanto ogni traffico permesso rappresenta un rischio. Idealmente, si permettono solo traffici permessi."

X

2. Firewall: Definizione e Notazioni

Definizione. Il *firewall* è un *dispositivo software* con due compiti:

- Analizzare il traffico IP
- Permettere o bloccare il traffico, a seconda delle regole configurate



Il firewall si può trovare sia sui *router di frontiera* che sui *dispositivi personali*. In un certo senso, il ruolo di firewall è quello di proteggere il "*ambiente interno*", ossia il dispositivo personale se si tratta di un firewall sui dispositivi personali, o l'organizzazione se si tratta di un firewall sui router di frontiera.

Oggi è *praticamente indispensabili*.

Il firewall si basa sulle "*regole*", che cosa sono precisamente?

Definizione. Le *regole firewall* sono *funzioni booleane a quattro parametri*: IP-src, PORT-src, IP-dst e PORT-dst. Per ogni parametro, possiamo usare gli operatori $=$, \wedge , \vee , $\in [\dots , \dots]$ per determinare se un certo valore sia ammissibile o meno (in realtà ci sono più operatori, ma per noi va bene).

Vediamo adesso come il firewall faccia a determinare se scartare o permettere del traffico IP.

ALGORITMO.

- Quando riceve un *pacchetto IP* con (IP-src:port-src, IP-dst:port-dst):
 - Per ogni regola nelle regole configurate:
 - Se regola(IP-src:port-src, IP-dst:port-dst) ha valore Vero: permettere il traffico
 - Altrimenti (termina il ciclo):
 - Scartare il pacchetto

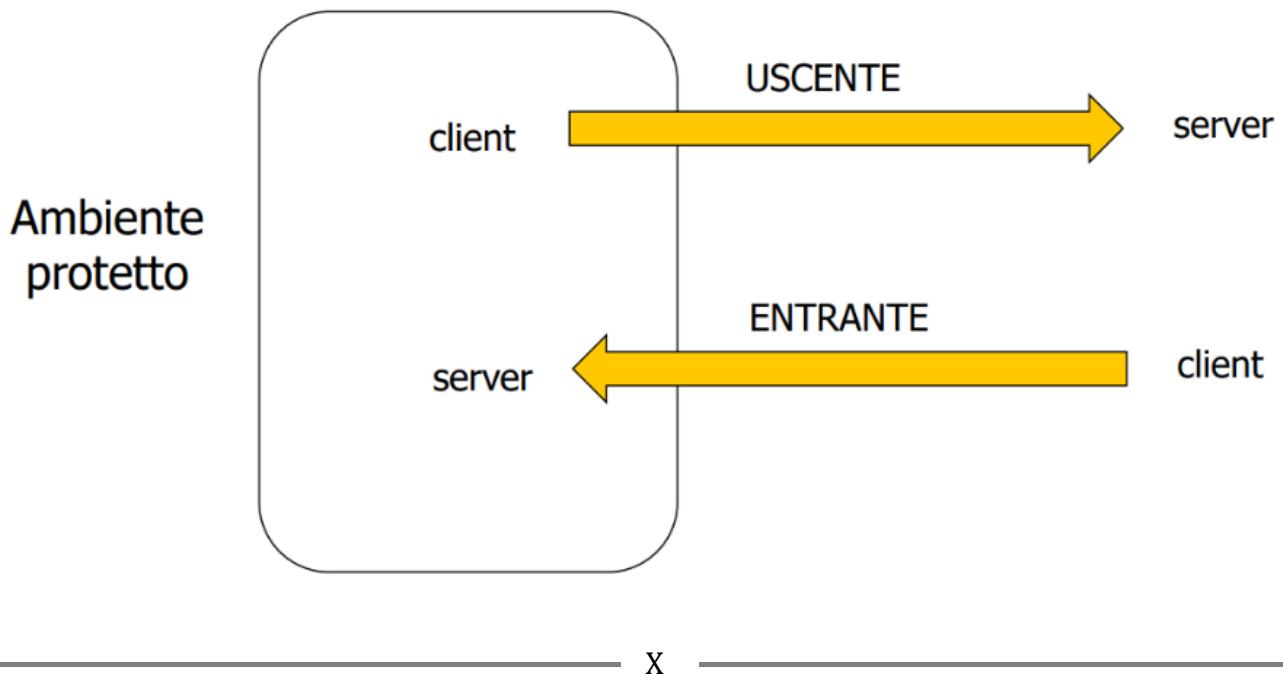
Osserviamo che quindi il firewall tende più a *proibire* che *permettere*, siccome i traffici permessi vanno specificati esplicitamente.

Osservazione. Essendo le connessioni TCP *bidirezionali*, ogni regola firewall che regola traffico TCP (poi encapsulato in IP) deve avere un suo "*coniugato*". In parole poche, devo permettere sia *richieste* che *risposte*.

Notazione. Per indicare la *configurazione* di un firewall, scriviamo una tabella dove:

- Ogni riga è una regola
- Ci sono 4 colonne, che sono i parametri delle regole firewall; quindi IP-src, PORT-src, IP-dst e PORT-dst.
- Ogni riga può avere opzionalmente un nome (vedremo dopo).
- Per indicare che un "*qualsiasi valore è accettabile*", si scrive "?". In un certo senso, indica che non possiamo conoscere quel valore a priori.

Osservazione. Il firewall di frontiera analizza solo "*traffici uscenti o entranti*". Nel gergo tecnico, per "*traffico uscente*" si intende la comunicazione client-server dove client sta nell'ambiente protetto; viceversa, per "*traffico entrante*" intendiamo la comunicazione server-client il server è collocato all'interno dell'ambiente. Ancora più semplicemente, basta chiedersi "*Chi apre la connessione TCP?*"; se è il nodo all'interno, è uscente; altrimenti è entrante.



2. Esempi ed Esercizi

Esempio. (*Proxy HTTP*)

Ci chiedevamo come un router di frontiera di un'organizzazione fosse effettivamente in grado di bloccare traffico HTTP non provenienti dal proxy. Si imposta il *firewall* ammettendo solo traffico HTTP uscente da parte del proxy. Ovvero,

name (optional)	IP-src	IP-dst	port-SRC	port-DST
HTTP Outbound	IP_PROXY	?	?	80/443
Conjugate	?	IP_PROXY	80/443	?

Osservazione. Per calcolare la regola "*coniugata*" basta scambiare i campi. Si sconsiglia fortemente di farlo, piuttosto è meglio ragionare sulla regola da scrivere; in questo modo, si può rilevare degli errori scritti nella regola precedente.

Per gli esercizi seguenti, supporre di stare all'interno di un'organizzazione.

Esercizio. Permettere invio mail da MS dell'organizzazione

Esercizio. Permettere dall'esterno il NS dell'organizzazione

Esercizio. Permettere al NS di risolvere nomi (contattare altri NS)

Esercizio. Permettere il prelievo mail

Esercizio. Permettere traffico HTTP uscente solo da una network N, sia Netnum(N) il suo network number.

Esercizio. Vedere esercizio P2 dell'elenco degli esercizi svolti.

Traccia. Ragionare prima su quali traffico DNS sono necessari, per ogni task da compiere; poi fare la cosa analoga per traffico HTTP e SMTP.

Esercizio. Supponiamo la seguente coppia di regole

name (optional)	IP-src	IP-dst	port-SRC	port-DST
rule_1	IP-MS	?	?	110
rule_2	?	IP_MS	110	?

Ha senso? Che tipo di regola è (usare la nomenclatura "outbound" o "inbound").

N.B. Negli esercizi, si considera più grave scrivere regole "*trop poco permissive*", invece di scrivere regole "*trop restrittive*".

X

3. Considerazioni su Firewall Reali

Facciamo un paio di considerazioni sui "*firewall reali*".

- In realtà la configurazione delle regole sono *molto complicate*, ma più potenti e flessibili. Esempio: Firewall su Windows

- Esistono più operandi per le regole

Inoltre, nella realtà, diventa difficile modellare le vere necessità per la configurazione delle regole, ossia quanti e quali server esistano, quanti client esistano e di cosa hanno bisogno ed eccetera...

Quindi una configurazione del firewall rappresenta una "*approssimazione*" delle necessità reali, e si può entrare in due casistiche:

- *Regole troppo restrittive*: alcuni programmi non funzionano e gli utenti si lamentano. Per motivi di carattere economico, è la più frequente (anche se dal punto di vista tecnico, è molto peggio)
- *Regole troppo permissive*: Funziona tutto, ma rimangono rischi non necessari.

Adesso facciamo un paio di pensieri sul firewall lato endpoint. *Windows Firewall* è uno dei programmi più indispensabili di sempre e permette *numerose* possibilità di configurazione senza costo aggiuntivo. Tuttavia, il profilo "*default*" permette praticamente tutto, e quindi non protegge niente.

Questa è una cosa che non dovrebbe succedere, siccome si ha uno strumento *indispensabile* per la difesa del proprio dispositivo ma non lo si usa. Il motivo a cui si riconduce principalmente è caratterizzato da *esigenze economiche e pratiche*, ovvero gli utenti non sanno configurare il firewall e non vogliono imparare o assumere qualcuno di farlo.

Poi questo rappresenta uno dei drammi odierni, siccome *molti dispositivi* presenteranno delle vulnerabilità RCE; tutti i dispositivi sono questi suscettibili ad attacchi come visti prima.

"Network Security"

Introduzione alla Network Security

X

Introduzione alla Network Security (\approx Cybersecurity): definizioni preliminari di subject e subject fisico, problema caratterizzante della Network Security. Esempi motivanti: HTTP injection, DNS spoofing. Scaletta del capitolo.

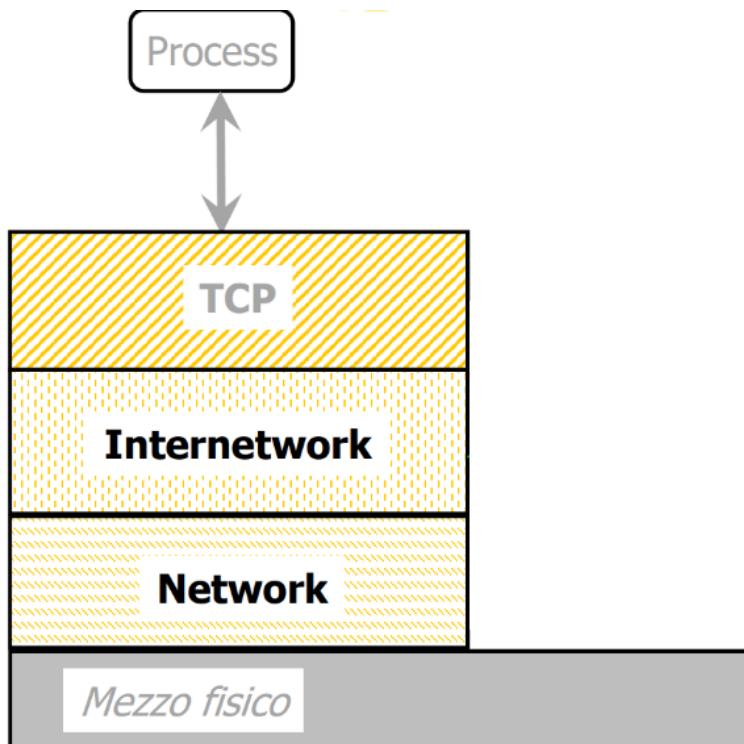
X

0. Voci correlate

- Comunicazione tra Processi
- Protocollo HTTP
- Aspetti Strategici del DNS

1. Network Security: Definizioni

Adesso torniamo al *livello applicativo* dello stack dei protocolli di Internet.



In particolare, approfondiremo su come possiamo applicare le *tecniche di crittografia* su applicazioni *quotidiane*.

Diamo prima delle definizioni preliminari:

B. Con B si denota una *sequenza di bytes* "qualsiasi", che può rappresentare:

- Un messaggio di un protocollo, come SMTP, POP, HTTP, IP/PPP, o anche dei *frame*
- Un file, una email, ...

Mittente/Creatore/Proprietario. Quando una sequenza di bytes B viene trasportata, si ha l'idea del "*mittente*" di B , o similmente l'idea del "*creatore*" / "*proprietario*" (dipende dal contesto). Si definiscono in due modi:

- *Esplicitamente*: la si specifica in un certo punto di B (ex: mail)
- *Implicitamente*: non viene specificato (ex: richiesta HTTP di una sessione autenticata)

Osservazione. Per "*mittente/creatore/proprietario*" si intendono anche i *processi* che creano B , non solo il lato utente

Subject e Subject Fisico. Un *subject* è una *stringa* nel calcolatore che indica l'utente (è circa l'*username*). Ad ogni subject si associa un *subject fisico*, che è un'utente o un'organizzazione esterna al calcolatore. Spesso ogni subject fisico è associato a più subject.

Esempio. (*Subject e Subject Fisico*)

Il DNS name è un esempio di *subject*, con subject fisico il *registrant* del DNS name.

Canale di Comunicazione. Quando un subject S genera B e la vuole trasportare ad un altro subject S' , la si fa mediante un *canale di comunicazione*. Notiamo che oltre a rappresentare uno spostamento nello spazio, si rappresenta anche uno spostamento nel tempo.

Osservazione. Il canale di comunicazione la si intende nel *senso astratto*, ovvero non è necessariamente Internet o la network. Può essere generalizzata anche in altri ambiti! (esempio: GPS)



2. Problema Fondamentale della Network Security

Vediamo il *problema unico* che caratterizza il campo della *network security*.

Problema.

Nel canale di comunicazione esiste un *network attacker* NA che vuole "*impedire*" il trasporto di B da S a S' . In particolare, supponiamo che NA sia in grado *osservare, modificare e*

fabbricare una qualsiasi sequenza di bytes che *"scorre"* nel canale di comunicazione.



Pertanto diventa impossibile comunicare a tutti gli effetti; uno degli obiettivi della *Network Security* è quello di definire delle *proprietà di sicurezza* sulla sequenza di bytes *B* e vedere come implementarla.

Osservazione. Si assume che la presenza del Network Attacker sia un *dato di fatto* a priori, siccome vogliamo pensare solo alle *conseguenze* del problema. Non pensiamo a *come sia possibile*, ed è un altro problema che è interamente diverso da quello che vogliamo risolvere.

Vediamo un paio di esempi motivanti, per sottolineare quali siano le *conseguenze* del problema posto sopra (^9d00d1)

Esempio. (*HTTP injection*)

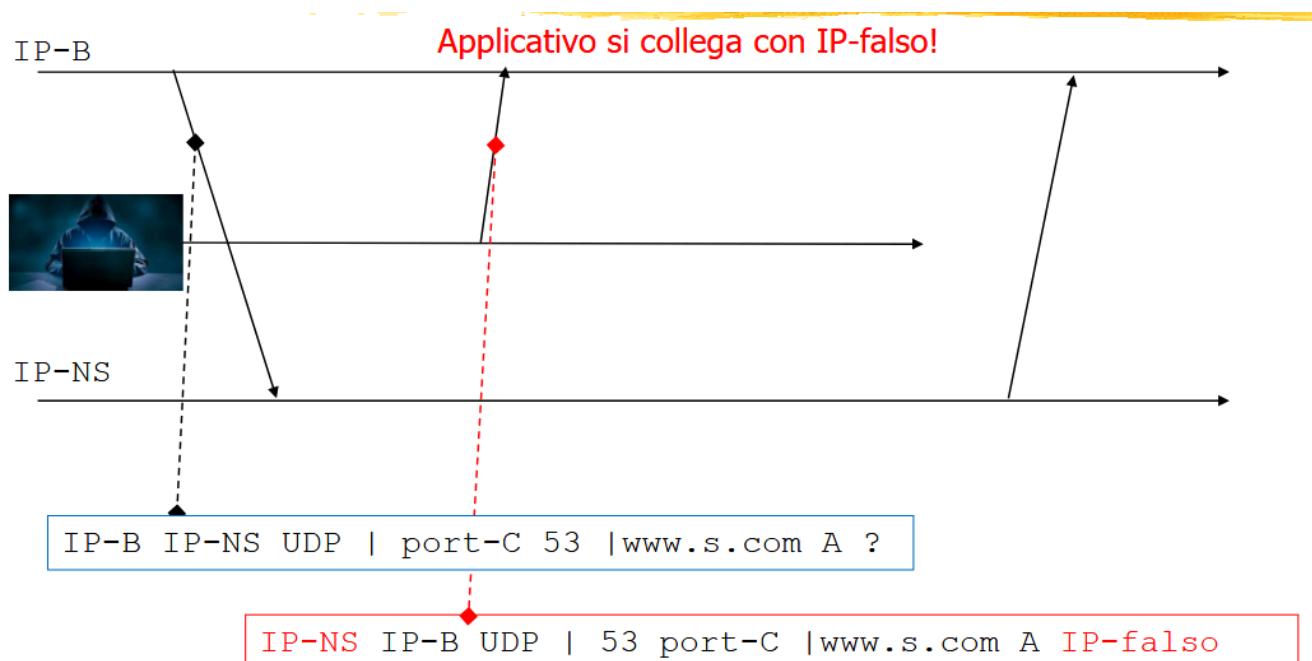
Supponiamo che il browser stia comunicando con un Web Server, e preleva la pagina dell'home page mandando una richiesta HTTP. Quello che può fare il network attacker *"in mezzo"* è quello di modificare la risposta HTTP modificando vari link (come quella per la pagina di login), così quando l'utente invia una richiesta POST per inviare le credenziali, lo manda al nodo del *network attacker*, non del *web server*.

BROWSER



Esempio. (*DNS spoofing*)

Quando un nodo invia una DNS request, è possibile che il network attacker lo intercetti e invia una risposta DNS *"fasulla"* più velocemente del *"DNS vero"*.



Esempio. (*Network Attacker*)

Ci sono molti casi in cui può esistere il network attacker, come:

- Sistemisti Access Point/Router/Name Server/ISP
- Enti statali/giudiziarie che impongono sorveglianza a ISP
- Reti WiFi aperte, "*tutti*" sono potenzialmente dei network attacker
- Reti WiFi "*personal*" in ambienti "*non personal*"
 - *Osservazione. Nelle reti WiFi, distinguiamo le reti "personal" da "enterprise" a seconda di quanto gli utenti autenticati si "fidino tra di loro".*

X

3. Scaletta

Quindi in questo capitolo vedremo tre aspetti della Network Security:

1. Definire le proprietà di sicurezza richieste
2. Definire dei strumenti matematici e studiarli dal punto di vista funzionale ("come si usano?")
3. Usare 2. per realizzare 1.

Proprietà di Sicurezza

X

Proprietà di sicurezza. Authentication, Riservatezza ed Integrità. Esempi. Osservazioni: nessun protocollo visto fin'ora garantisce queste proprietà, authentication e riservatezza da un punto di vista pratico. Altre proprietà importanti. Osservazione: le proprietà di sicurezza "esistevano" prima dell'Internet.

X

0. Voci correlate

- Introduzione alla Network Security

1. Le Tre Proprietà Fondamentali

Authentication

Supponiamo che in una sequenza di bytes B ci sia informazione *implicita/esplícita* sul suo "*mittente/proprietario*" subject S . **Authentication** consiste nel garantire che B sia "*veramente*" generata da S

Esempio. Richieste e risposte HTTP. Quando si garantisce che una richiesta HTTP sia generata *veramente* da un certo browser? Quando si garantisce che la *risposta HTTP* (con la pagina) sia veramente generata dal web server?

Esempio. Possiamo vedere i protocolli su livelli più bassi, come:

- Frame Ethernet con ETH-src = ETH-x: come garantiamo che sia veramente da ETH-x?
- Pacchetto IP con IP-src = IP-x: come garantiamo che sia veramente dal nodo di IP-x?

Esempio. Anche applicabile su *bytes intesi come dati*, vedere ad esempio il caso della *mail address spoofing* (Email Spam e Address Spoofing)

Riservatezza

Sia B una *sequenza di bytes* inviata da S a S' tramite un canale di comunicazione. La *riservatezza* è quando B è "*comprendibile*" solo ad un insieme di "*subject autorizzati*"

Vedremo di capire meglio le nozioni di "*comprendibile*" e "*subject autorizzati*" in seguito.

Esempio. Richieste HTTP con Username e Password... importante che siano comprensibili solo al WS!

Naturalmente ci sono altri numerosissimi esempi applicati a tipologie di B diverse, come files, immagini, traffico DNS con nomi dei siti, eccetera...

Integrità

Sia B costruita all'istante di tempo t_1 e "usata" all'istante di tempo t_2 . *Integrity* consiste nel garantire che B sia sempre lo stesso nelle due istanti di tempo, ossia

$$B(t_1) = B(t_2)$$

Osservazione. Luogo di costruzione può essere \neq al luogo di verifica, come il subject che costruisce B può essere \neq al subject che verifica

Esempio. Pagina web contenente *link critico* (login page):

- t_1 : Invio dal web server
- t_2 : Ricezione dal browser

Esempio. Referto medico

- t_1 : Costruzione del referto
- t_2 : Analisi in procedimento giudiziario (eventuale)

In alcuni casi l'*integrità* diventa un'aspetto cruciale.

Notiamo che nessun protocollo tra quelli visti fin'ora garantisce le proprietà appena definite, al massimo abbiamo solo *autenticazione al lato client* in situazione "*senza network attacker*".

Osservazione. Nelle applicazioni pratiche vogliamo garantire l'*autenticazione e la riservatezza* anche al livello di *subject fisico*, ovvero vogliamo garantire che un *subject* sia veramente associato al *subject fisico*. Questo richiede dei procedimenti aggiuntivi che non vedremo

Osservazione. Integrity non ha nulla a che fare con Subject

1.1. Altre proprietà

Ci sono delle proprietà altrettanto *importanti* che non vedremo in profondità:

- "*Non Repudio*" e "*Plausible Deniability*": Garantire che un subject non può negare di aver mandato B , o che può dimostrare di non aver mandato B
- "*Timestamp*": B deve esistere in un istante specificato
- *Availability*: Il sistema funziona, ovvero il network attacker non blocca la comunicazione (!)
- ...

Osservazione. Tutte le proprietà viste sono sempre state richieste sin dai tempi dei *romani*. Ciò implica naturalmente che ci sono state delle soluzioni "*pre-Internet*" per garantire le suddette proprietà, e tipicamente si basavano su due elementi:

- Contatto fisico tra subject fisici
- Comunicazioni o documenti cartacei

Crittografia a Chiave Privata e HMAC

X

Strumenti matematici per la Network Security. Premessa: assunzione della perfezione. Crittografia a chiave privata: idea, algoritmo e correttezza, esempio storico Enigma. HMAC: Idea, osservazioni e correttezza.

X

0. Voci correlate

- Proprietà di Sicurezza
- Introduzione alla Network Security

1. Premessa

Premessa. D'ora in poi assumiamo la *perfezione*:

- In implementazione degli algoritmi e protocollo
- Degli software
- Procedure operative

Questa è un'ipotesi piuttosto irrealistica, infatti spesso non vengono verificate e quindi crollano le proprietà di sicurezza per questo motivo.

Esempio. Per dare un'idea, abbiamo il seguente scenario:

- Per qualche errore sul software (vulnerabilità RCE, ad esempio) è in grado di eseguire dei programmi sul nodo
- Quindi il Network Attacker diventa Node Attacker
- Crollano le proprietà di sicurezza, siccome tengono solo del *Network Attacker*

Osservazione. Dunque gli strumenti matematici che vedremo non risolvono automaticamente i problemi! Infatti, per citare l'informatico britannico più importante nell'ambito della Cybersecurity,

"Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it".

X

2. Crittografia a Chiave Privata

Introduciamo il *primo strumento matematico*, la crittografia a chiave privata.

IDEA. Assumiamo che *due subject* condividano un *numero naturale* K di N bit, detta "*chiave privata*". Utilizzano K per garantire la *secrecy*, in particolare usando algoritmi di *codifica e decodifica* parametrizzate in K che siano "*mutualmente inversibili*" per stesso parametro.

Notazione. Il procedimento di "*criptaggio di un messaggio con chiave K* " si denota con $\text{ENCRYPT}_K(m)$, invece il procedimento di "*decrittaggio di un messaggio cifrato con chiave K* " si denota con $\text{DECRYPT}_K(c)$.

Quindi formalmente, l'idea di base è quella di usare *uno/due* algoritmi di crittaggio e decrittaggio tali che

$$m = \text{DECRYPT}_{K'} \left(\underbrace{\text{ENCRYPT}_K(m)}_c \right) \iff K = K'$$

Adesso vediamo di fare un paio di precisazioni formale sulle *chiavi K* :

- Le chiavi sono sequenze di N bit
- Pertanto ci sono 2^N chiavi possibili per N fissato. Il valore N dipende dagli algoritmi usati...

Proprietà. Affinchè l'algoritmo possa garantire la *segretezza/riservatezza*, deve valere la seguente proprietà fondamentale:

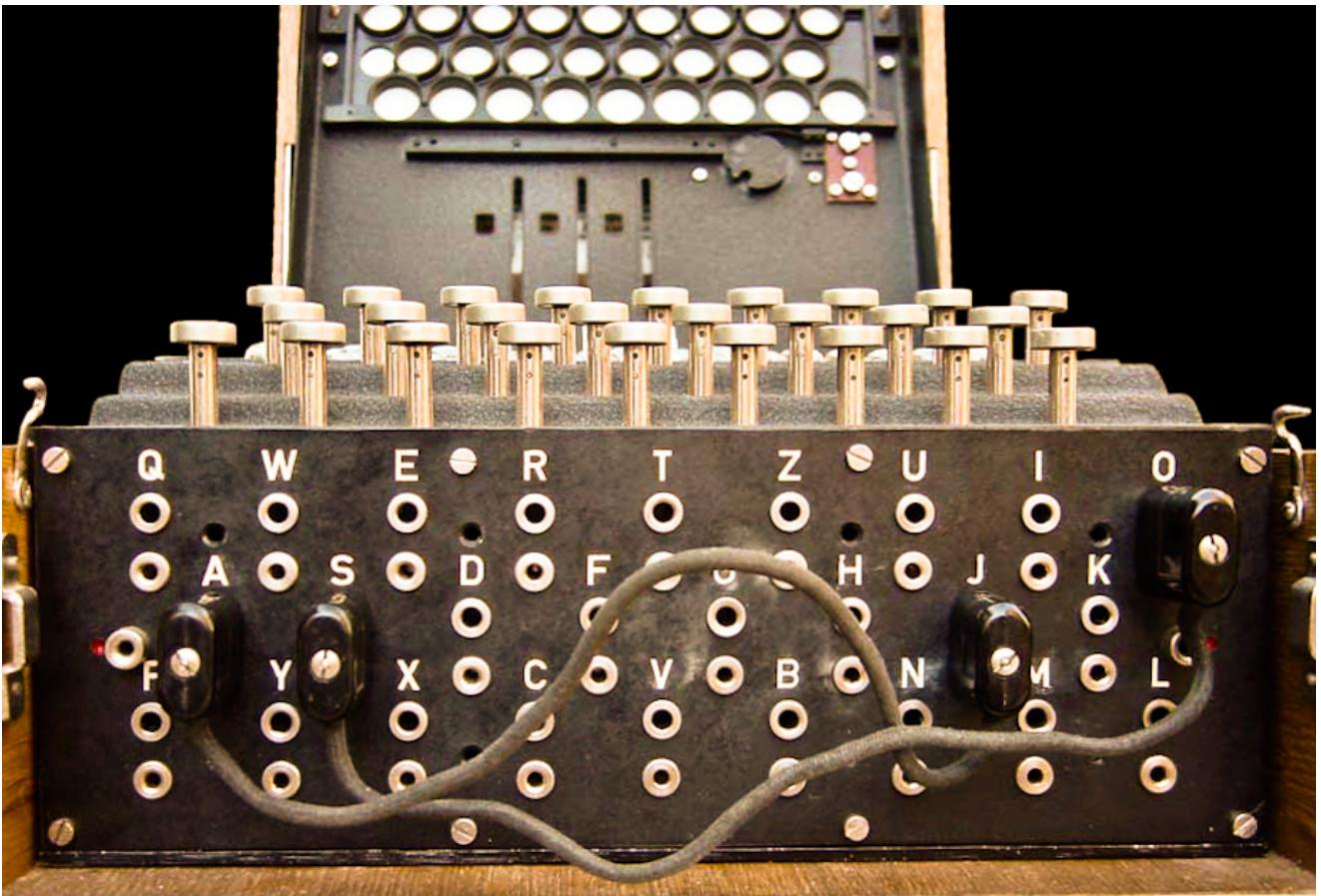
- E' "*difficile risalire*" a k da un testo cifrato
- E' "*difficile risalire*" al messaggio chiaro dal testo cifrato

In questo modo, un *network attacker* che conosce l'algoritmo di crittaggio/decrittaggio e che ha il testo cifrato, può tentare di evincere il plaintext in due modi:

1. *Ricerca esaustiva*: Indovinare la chiave K , quindi su 2^K chiavi possibili usare l'algoritmo $\text{DECRYPT}_K(\bullet)$ per evincere K e m (lo capisce quando il testo decrittato "*ha senso*")
2. *Crittoanalisi*: Osservare il testo cifrato ed evincere le informazioni sul chiave, in particolare informazioni di tipo "*di esclusione*" (nel senso che si esclude un sottoinsieme dello spazio delle chiavi), poi per applicare nuovamente la ricerca esaustiva sullo spazio ridotto

Tuttavia si dimostra che oggi entrambi gli approcci sono inutilizzabili per gli *algoritmi moderni*, siccome permettono N sufficientemente grande e non permette l'estrazione delle informazioni su K o plaintext.

Esempio. Un esempio storico della crittografia a chiave privata è la macchina *Enigma*, che però era vulnerabile alla *crittoanalisi* e infatti ad un certo punto della 2GM (seconda guerra mondiale) le chiavi private sono state scoperte, permettendo alle forze alleate di ottenere informazioni preziose.



Esempio. Alcuni algoritmi moderni:

- DES (56 bit, sono *"pochi"*)
- AES (128 o 256 bit), la più diffusa oggi
- ...

Osservazione. Le tecniche di crittografia non garantiscono comunque l'*integrità* dei messaggi... infatti, il Network Attacker può comunque modificare il *ciphertext*, cambiando quindi il testo decrittato senza che il ricevente ne accorga. Analogamente, non si garantisce neanche l'*autenticazione*, dal momento che può inviare un *"messaggio casuale"* e mandarlo al ricevente.

X

3. HMAC (Message Authentication Code)

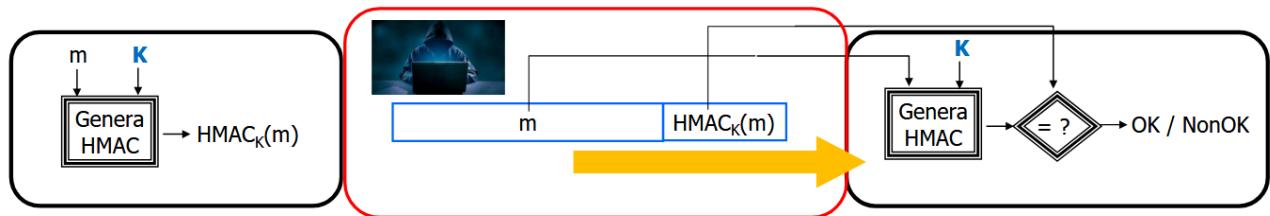
Vediamo un altro strumento per garantire un'altra proprietà di security.

Idea. Come sempre, S, S' condividono una chiave segreta K per garantire l'*autenticazione* e la *segretezza*. Si definisce una funzione parametrizzata in K , che denoteremo con $\text{HMAC}_K(\bullet)$, ed essa genera a partire da B un'altra sequenza di bytes con *lunghezza fissa*.

Per garantire le due proprietà sopra, si appende al messaggio originale m la nuova sequenza generata, ovvero $m' = m \uplus \text{HMAC}_K(m)$. In questo modo, quanto S' riceve $(m \uplus \text{HMAC}_K(m))$, può prima calcolare $\text{HMAC}_{K'}(m)$, e si ha che

$$\text{HMAC}_{K'}(m) = \text{HMAC}_K(m) \iff K = K'$$

Quindi il messaggio è *autentico e integro* siccome chi ha prodotto il messaggio conosce la chiave privata, pertanto il lato ricevente dovrà solo controllare che i valori HMAC coincidano.



Osservazione. In ogni caso, *HMAC* da sola non fornisce la *secrecy*! Quindi un network attacker può vedere il messaggio in chiaro, ma non può nè falsificare nè modificare il messaggio senza che il receiver che se ne accorga.

Si dimostra che gli algoritmo odierni di HMAC sono tali che ottenere $K, \text{HMAC}_K(m)$ solo da m sono "praticamente impossibili" (nel senso che l'attacker può solo usare la forza bruta).

Osservazione. Con l'HMAC leghiamo la nozione di *autenticazione* alla *conoscenza* della chiave K , quindi del software che conosce quel valore, invece non c'è nessun cenno di *subject* o *subject fisico*. Infatti, come ipotesi implicita si assume che *solo* S, S' conoscano K .

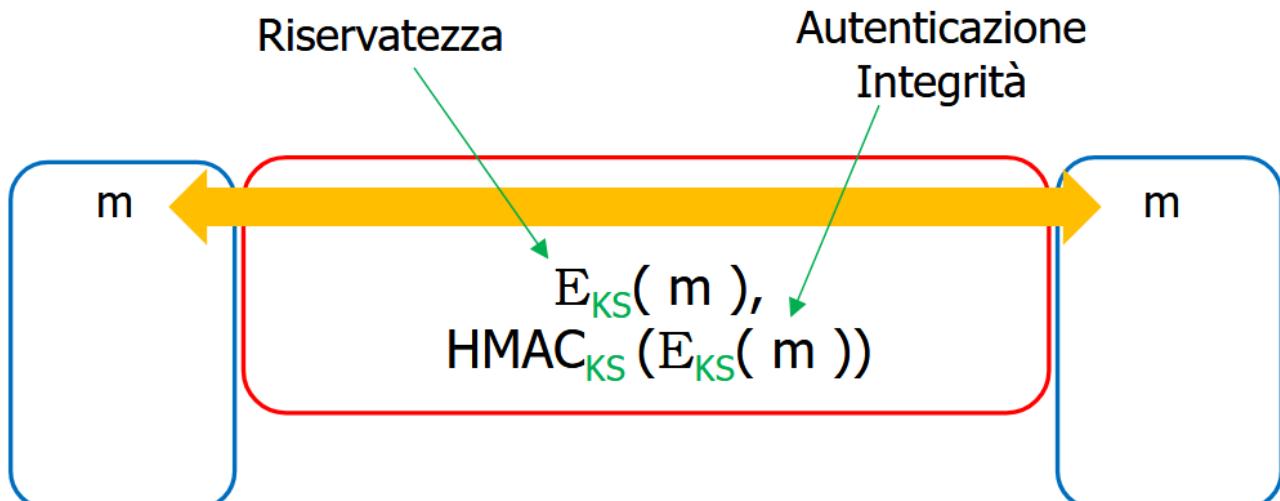
X

3. Realizzazione Parziale delle Proprietà di Sicurezza

Osservazione. Notiamo che sotto l'assunzione di aver distribuito correttamente le *chiavi segrete* tra due subject, il gioco è stato fatto. Infatti, per garantire tutte le proprietà di comunicazione (segretezza, autenticazione e integrity) si modifica il messaggio m nel seguente modo:

$$m \leftarrow \text{ENCRYPT}_K(m) + \text{HMAC}_K(\text{ENCRYPT}_K(m))$$

Nella matematica della crittografia si dimostra che è necessario *prima* crittare il messaggio e dopo calcolare l'*HMAC*, altrimenti si rischia di fare un "*leak*" sulla chiave o messaggio in chiaro.



Quindi tutto è stato fatto... o no? (*Spoiler: No, vedere l'assunzione di base degli strumenti matematici*)

Crittografia a Chiave Pubblica

X

Motivazione: distribuzione delle chiavi private. Crittografia a chiave pubblica: idea dell'algoritmo, proprietà fondamentale. Esempio: RSA. Osservazione: novità "storica" del strumento matematico, curiosità storiche. Osservazione: differenza dalla crittografia a chiave privata, non garantisce né integrità né autenticazione. Terminologia per crittografia a chiave privata e pubblica: asimmetrica e simmetrica.

X

0. Voci correlate

- Introduzione alla Network Security
- Proprietà di Sicurezza
- Crittografia a Chiave Privata e HMAC

1. Distribuzione delle Chiavi Private

Osservazione. La crittografia a chiave privata e HMAC è in grado di garantire le proprietà di sicurezza se c'è un'ipotesi fondamentale di base: ovvero, che le *chiavi private* vengono condivise tra Subject.

Q. Come fanno i subject a condividere la chiave privata?

Il problema della *distribuzione delle chiavi private* è un altro grande problema.

Esempio. (*Storico*)

Storicamente, si usava un altro *canale sicuro* di comunicazione che per definizione garantisce le proprietà di comunicazione. Ad esempio, incontrarsi di persona e scambiare le chiavi con valigette diplomatiche o oggetti del genere.



Problema. Il canale sicuro è *costoso* e "*ad alta latenza*", quindi veniva utilizzato "*poco*" in storia. Tuttavia, il problema è diversamente adesso: ci sono *tantissimi copie di Subject* che

vorrebbero scambiarsi chiavi private, basta pensare ad esempio ogni utente di GMAIL.

Infatti, se il canale sicuro fosse economico e a "bassa latenza" non ci sarebbe stata nessuna necessità di designare le *chiavi private e HMAC*... basta utilizzarli direttamente

In pratica, il canale aggiuntivo è utilizzabile solo in casistiche molto specifiche:

- Poche coppie di Subject
- Regole note a priori
- Staticità

Che è il contrario dello scenario "*Internet*".

Come si fa? Vedremo uno strumento matematico in più...

Osservazione. La crittografia quantistica NON RISOLVE QUESTO PROBLEMA! Infatti, parte sempre dal presupposto che le chiavi siano già distribuite...

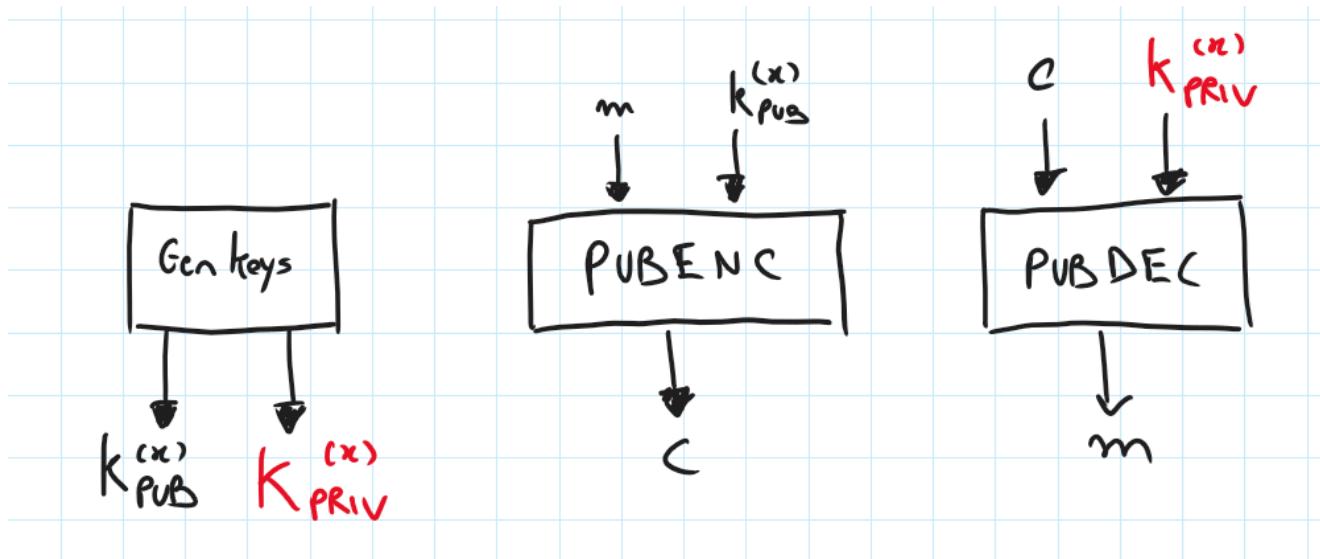
X

2. Crittografia a Chiave Pubblica

IDEA. Un algoritmo *genera* una coppia di chiavi $(K_{\text{PRIV}}, K_{\text{PUB}})$ chiamati "*chiave privata*" e "*chiave pubblica*" (vedremo perché dopo) che abbiano una "*relazione matematica*" tra loro, poi altri *due algoritmi* di crittaggio e decrittaggio pubblico, scritti come $\text{PUBENCRYPT}_{\bullet}(\bullet)$ e $\text{PUBDECRYPT}_{\bullet}(\bullet)$ soddisfano la seguente equivalenza:

$$\text{PUBDECRYPT}_{K'_{\text{PRIV}}}(\text{PUBENCRYPT}_{K_{\text{PUB}}}(m)) = m \iff K'_{\text{PRIV}} = K_{\text{PRIV}}$$

In parole povere, solo chi ha la chiave privata può decrittare il messaggio ma tutti possono crittare il messaggio.



Osservazione. Lo spazio delle chiavi *sono un sottoinsieme* delle coppie dei numeri a N bit, siccome devono soddisfare delle relazioni matematiche.

Le proprietà fondamentali che garantiscono delle proprietà di sicurezza sono come seguono:

- Dalla chiave pubblica non *"si evince"* la chiave privata
- Dal messaggio cifrato non *"si evince"* la chiave privata
- Dal messaggio cifrato non *"si evince"* il messaggio in chiaro

Questo strumento è in grado di realizzare la *segretezza* se partiamo dal presupposto che la chiave pubblica sia *"già distribuita"* al nodo sorgente. Infatti, l'attacker che conosce l'algoritmo e chiave pubblica non può fare niente senza la chiave privata!

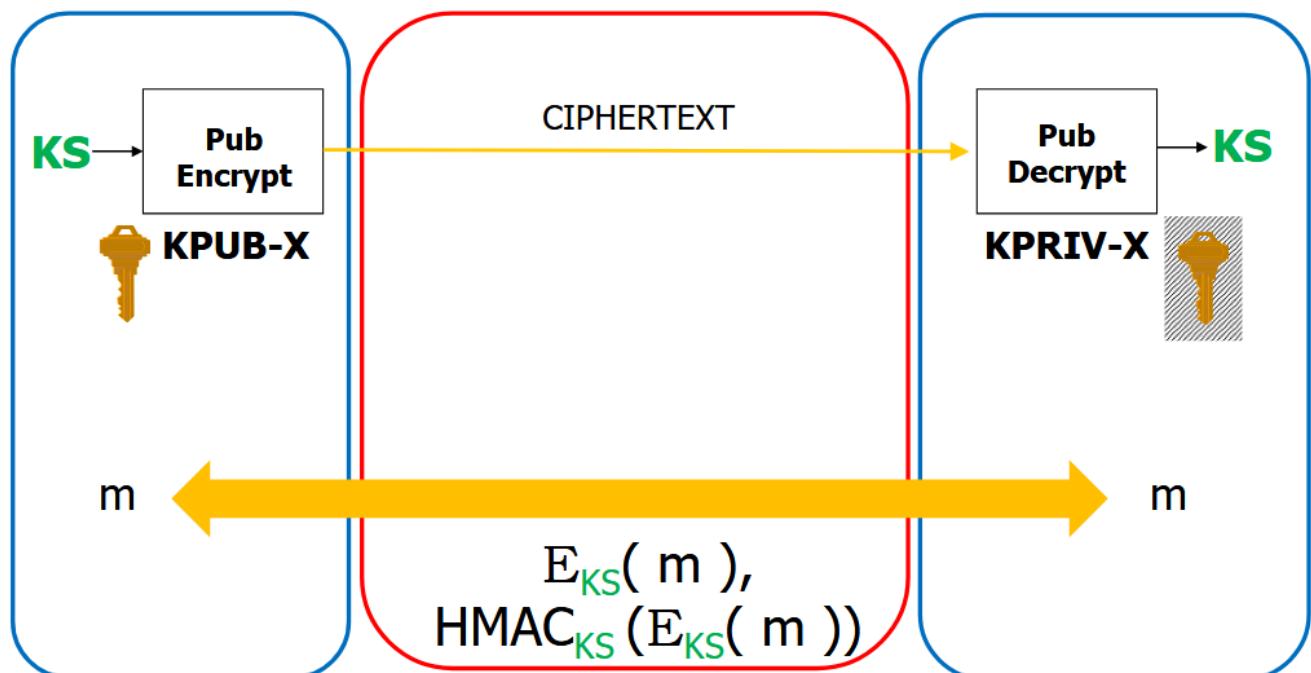
Esempio e Cenni Storici. Gli algoritmi moderni di crittografia a chiave pubblica sono stati inventati nel 1976 da Diffie e Hellman, Markle da un punto di vista *concettuali*, tuttavia a livello implementativo non erano riusciti a capire quali algoritmi usare. Dopodiché Rivest, Shamir e Adleman sono riusciti a inventare l'algoritmo *RSA* che implementa la crittografia a chiave pubblica con grande successo e infatti sono diventati ricchi . In realtà, si è scoperto che uno studente li aveva inventati prima di loro in un tirocinio estivo in un'organizzazione di intelligence britannica, tuttavia per motivi di lavoro non poteva pubblicarlo.

Osservazione. La *"chiave privata"* della crittografia a chiave pubblica non è correlata in nessun modo alla *"chiave privata"* della crittografia a chiave privata.

Terminologia. La *"crittografia a chiave asimmetrica"* è sinonimo di *crittografia a chiave pubblica*, invece la *"crittografia a chiave simmetrica"* è sinonimo di *crittografia a chiave privata*. I termini appena detti sono utilizzate nelle normative italiane.

Osservazione. La crittografia a chiave pubblica è più lenta della crittografia a chiave pubblica in 1-2 ordini di grandezza (x10-100)

Osservazione. Non garantiscono comunque integrità, né tantomeno l'autenticazione; bisognare usare HMAC!



Certificati e Certification Authority

X

Problema motivante: distribuzione delle chiavi pubbliche. Certificati e Certification Authority: definizione di certificato, certification authority. Procedura di rilascio dei certificati da parte dei CA, due casi. Esempi: DNS-name e firme digitali.

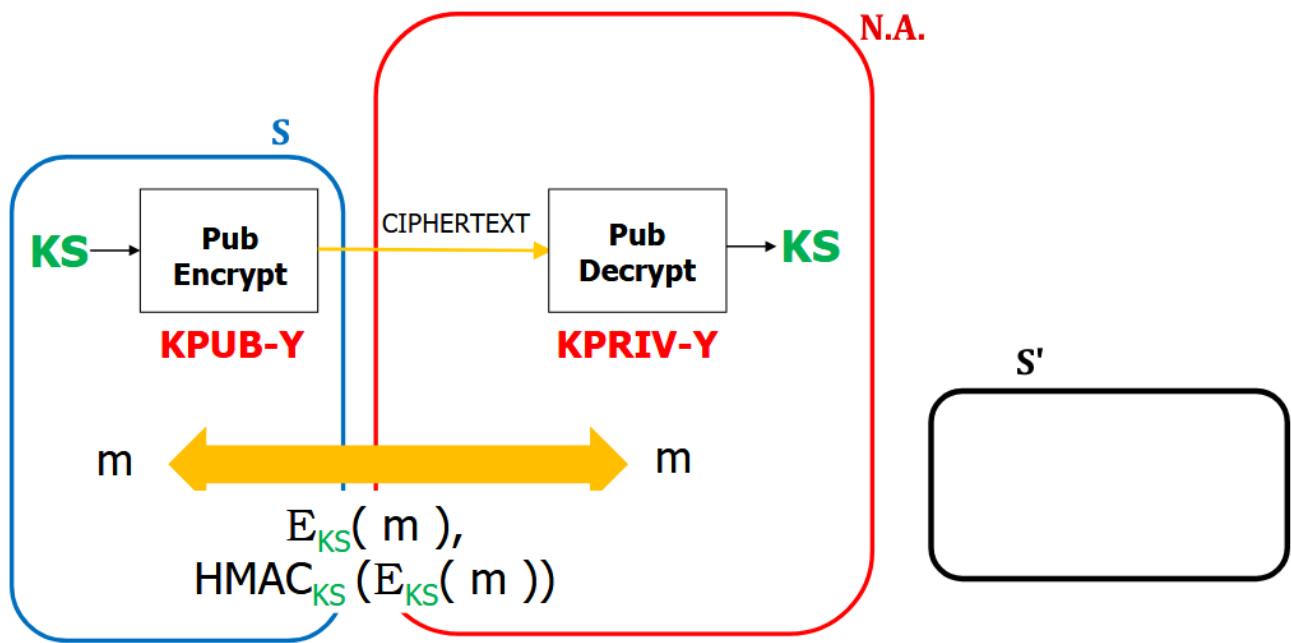
X

0. Voci correlate

- Crittografia a Chiave Pubblica
- Introduzione alla Network Security

1. Distribuzione delle Chiavi Pubbliche

Osservazione. Col strumento della crittografia è chiave pubblica è possibile realizzare le proprietà di sicurezza. Tuttavia, ciò assume che di base le chiavi pubbliche vengano *distribuite!* Questo costituisce un altro scoglio nel nostro cammino, siccome il network attacker è in grado di *distribuire delle chiavi pubbliche "false"* annullando così tutte le proprietà di sicurezza realizzate.



Una soluzione teorica al problema è quella di usare un *canale sicuro aggiuntivo*; tuttavia, questa è impossibile.

Vediamo dunque un *approccio moderno* al problema.

Approccio. Sia DNS-Name il *Subject*, e il server che *"possiede"* il DNS name deve distribuire l'associazione (DNS-name, K_{PUB}). Gli step che deve realizzare sono le seguenti:

- *Punto d'arrivo*: il Subject Fisico S' è in grado di comunicare la coppia (**DNS-name**, K_{PUB}) sul canale non sicuro al Subject Fisico S, che poi verificherà se è valido o meno "*per conto suo*", e nel caso di esito positivo concordano la *chiave privata simmetrica*. In questo caso, il network attacker sarà solo in grado di impedire la comunicazione, siccome non sarà né in grado di alternare né di falsificare la coppia che "*certifichi*" il diritto di utilizzare un certo subject.
- *Punto di partenza*: Il server S' è "*configurato in modo sicuro*" a priori e ha usato un *canale sicuro di comunicazione* solo una volta a priori
- Poi vediamo di mettere tutto assieme!

L'approccio si generalizza anche su *subject* di tutti i tipi, per ora ci focalizziamo di questo tipo.

X

2. Certificati e Certification Authority

Vediamo adesso il "*punto di partenza*" della scaletta stilata prima, per affrontare il problema della distribuzione delle chiavi pubbliche. Facciamo un paio di definizioni preliminari.

Certificato, Certification Authority

Un *certificato* è un *file* che contiene la coppia (**SUBJECT**, K_{PUB}^S) codificato mediante dei protocolli specifici; inoltre contengono il nome dell'*entità* che garantisce la veridicità dell'associazione (quindi ad essere pignoli è una tripla).

L'entità che garantisce la veridicità sono le *certification authority*, e sono le stesse entità che generano i *certificati* (in linea di principio...)

Il Certification Authority garantisce quindi la seguente catena di legami, mediante il certificato:

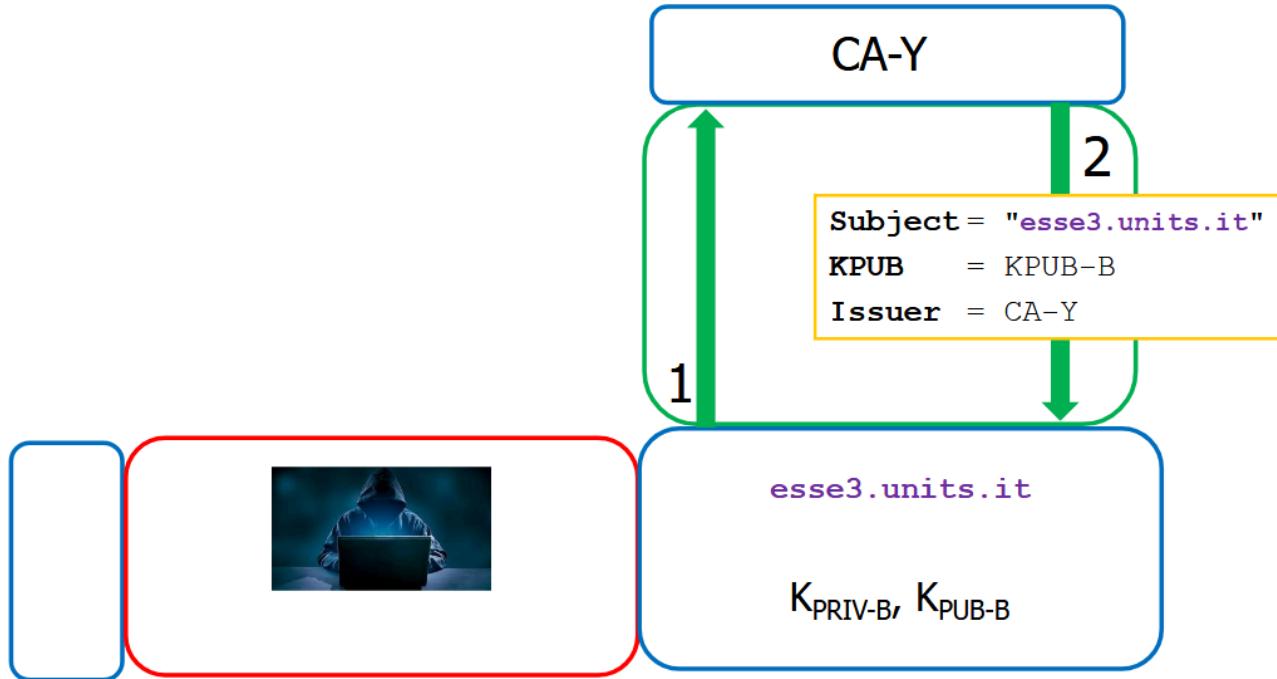
$$\text{Subject Fisico} \longleftrightarrow \text{Subject} \longleftrightarrow K_{\text{PUB}}^{(S)}$$

Ovvero:

- **Subject Fisico** \longleftrightarrow **Subject** vuol dire "*Subject Fisico ha diritto di utilizzare una certa stringa come Subject*"
- **Subject** \longleftrightarrow $K_{\text{PUB}}^{(S)}$ vuol dire *"*Subject deve conoscere la chiave privata in relazione a $K_{\text{PUB}}^{(S)}$* "

Q. Supponendo che un server abbia la coppia chiave privata e pubblica, come può il server S (il subject fisico) a farsi rilasciare un *certificato* da una *certification authority*?

Generalmente, il *certification authority* prima si accerta che il *Subject Fisico "abbia diritto"* di autenticarsi come *Subject X*, poi nel caso positivo dell'accertamento il CA rilascia il *certificato* mediante un *canale di comunicazione sicuro*.



Notiamo che quindi in questo caso il server ha "*utilizzato una sola volta*" un canale sicuro di trasmissione.

Q. In che senso "*avere diritto*" di identificarsi come Subject X? Come si fanno i controlli? Come si accerta che il subject fisico sia chi dice di essere?

Purtroppo questi dettagli dipendono dall'*uso del certificato*, facciamo dei cenni.

Esempio. (*DNS name*)

1. CA comunica un *numero casuale* al Subject Fisico
2. Il Subject Fisico crea un *RR* di tipo *TXT* di nome *DNS-name* con valore del numero casuale comunicato prima
3. Il CA effettua una resolution in *DNS-name* con tipo *TXT*. Se lo trova, OK.

Esempio. (*Firma digitale con validità legale*)

1. Il CA incontra il subject fisico in persona o mediante una call telematica ed effettua un controllo della documentazione personale

Osservazione. Notiamo che abbiamo sempre assunto che all'inizio il subject fisico abbia la coppia *chiave privata-chiave pubblica* già all'inizio. Nella realtà abbiamo un altro caso molto più comune, ossia è il Certification Authority a rilasciare la *coppia delle chiavi* e poi li fornire al subject fisico, poi per cancellarli.

Comunque esistono i casi in cui il Subject Fisico abbia già la *key pair*, in questo caso li mette in un file di formato "*Certificate Signing Request*" e poi lo invia al CA e richiede di verificarlo.

Distribuzione e Validazione dei Certificati

X

Distribuzione e validazione dei certificati: problema, ipotesi "magica" (cenno).

X

0. Voci correlate

- Certificati e Certification Authority

1. Distribuzione e Validazione dei Certificati

Supponiamo che una CA abbia rilasciato un *certificato* che attesti il diritto del subject fisico di riconoscersi come subject X, che a sua volta conosce la chiave privata.

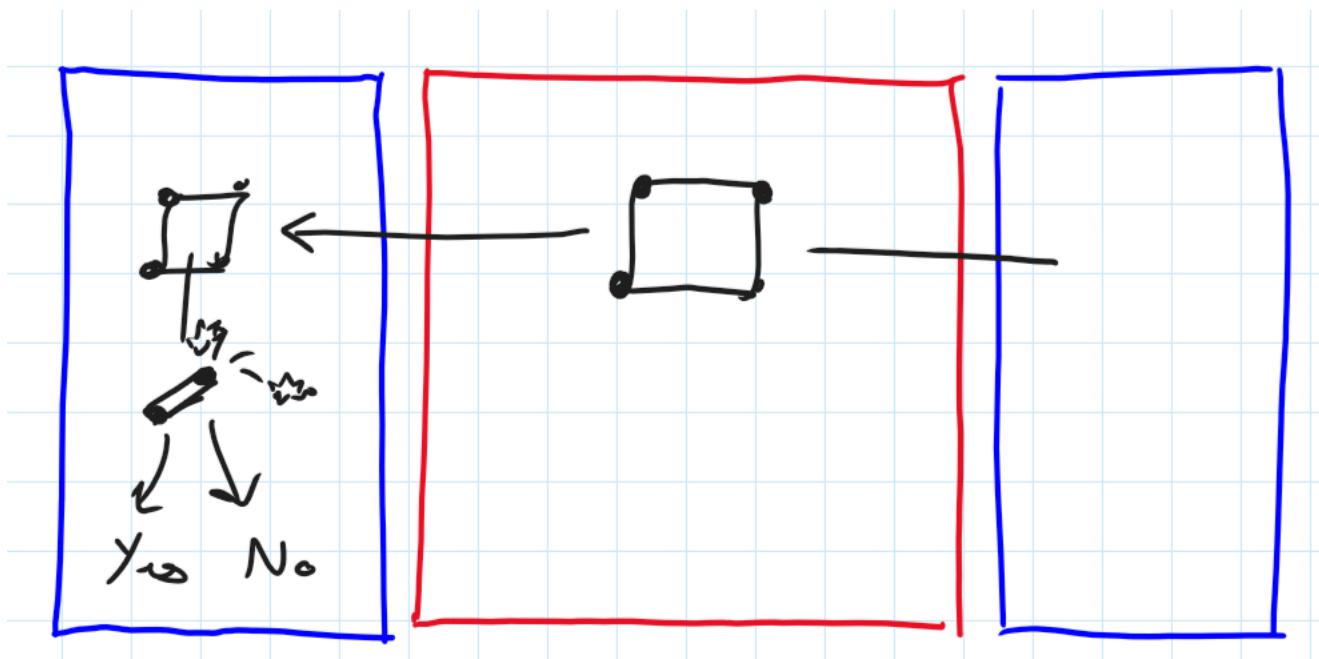
Q. Come si distribuiscono i certificati?

Siccome è possibile avere ulteriori inghippi con l'attacker, sarebbe da stare attenti siccome si potrebbe falsificare o modificare il certificato, annullando dunque l'*autenticità* e l'*integrità*.

Per risolvere il problema, facciamo la seguente ipotesi "*magica*"

ⓘ Ipotesi Magica

Si suppone che, dato un *certificato*, ogni subject è in grado di verificare che sia *autentico* e *integro* con uno "*strumento magico*", e può farlo indipendentemente dal canale dal quale "*esce*" il certificato.

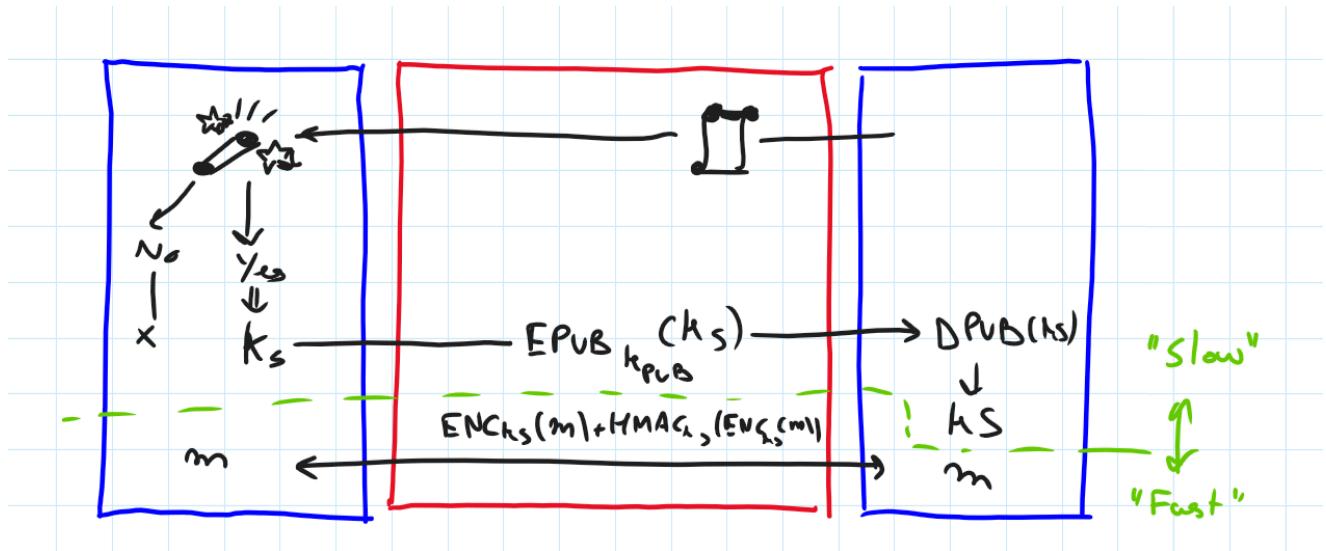


Vedremo dopo come sarà possibile implementare questo "*test magico*".

Osservazione. Notiamo che supponendo pure vera questa ipotesi, è comunque possibile che le CA rilascino dei "*certificati fasulli*" (per errore o frode); in questo caso, facciamo finta di nulla e assumiamo che "*Ogni CA dice sempre la verità*". Per quanto riguarda la praticità, vediamo dopo....

Osservazione. Alla fine, siamo riusciti a realizzare un percorso che implementi le *funzionalità di sicurezza* sui flussi di byte. Ad esempio, un procedimento è come segue:

1. Server ottiene un *certificato* con una chiave pubblica, poi inoltre possiede la chiave privata associata
2. Server manda a Client il *certificato*
3. Client verifica il certificato
4. Nel caso positivo, client sceglie un *valore* per la *chiave simmetrica* e lo comunica al Server mediante crittografia a chiave pubblica
5. D'ora in poi Server e Client si comunicano mediante la *crittografia a chiave privata*



Notiamo che le parti 1-3 è rappresentano parte "*lenta*" della comunicazione, e viene usata solo per scegliere la chiave simmetrica; invece le parti 4-5 sono "*veloci*" e si usano per comunicare i *dati*.

Il procedimento appena descritto è una *tecnica generale* di cui si basano "*quasi*" tutte le applicazioni pratiche degli strumenti matematici appena descritti. Infatti:

- Rendiamo il canale non sicuro in sicuro con la *crittografia chiave privata*
- La comunicazione per condividere la *chiave privata simmetrica* viene fatta con la *crittografia a chiave pubblica*
- Si distribuiscono le chiavi pubbliche su *canali non sicuri* in "*modo sicuro*" mediante i *certificati*

Osservazione. I certificati sono pubblici... per forza; non c'è nulla di segreto

Esempi. Vediamo alcuni *standard* di certificati:

- X509: Molto comune e hanno validità legale in Italia
- PGP: Meno comune, presenti solo su alcuni software

Osservazione. La verifica dei certificati è ancora più complessa, infatti ci sono almeno altri due aspetti da vedere (non li vedremo, adesso accenniamo):

- *Data di scadenza*: ogni certificato ha una *periodo di validità* da una data X a data Y; se il software nota che il "*giorno corrente*" non sta nel periodo di validità, allora ritiene invalido il certificato
- *Scopo*: ogni certificato è rilasciato per *alcuni usi*, ad esempio un certificato per "*server auth*" si ritiene invalido per uso di "*digital signature*".

Crittografia a Chiave Pubblica in Pratica

X

Aspetti pratici della crittografia a chiave pubblica. Concetto intuitivo di Trust Set e Key Set, test di validità dei certificati. Esempio fondamentale: certificato valido e certificato non valido. Significato di certificato non valido in pratica. Esempio di certificati non validi: certificati HTTPS scaduti. Definizione formale di Trust Set, Key Set e Personal Set.

X

0. Voci correlate

- Certificati e Certification Authority
- Crittografia a Chiave Pubblica
- Introduzione alla Network Security

1. Aspetti Pratici della Crittografia a Chiave Pubblica

Premessa. Ogni certificato in arrivo proviene da un *canale non sicuro*, non lo disegneremo esplicitamente.

In pratica, non è vero che "*ci fidiamo*" sempre delle Certification Authority o che sia *sempre possibile* effettuare il test di validità di un certificato. Nella realtà:

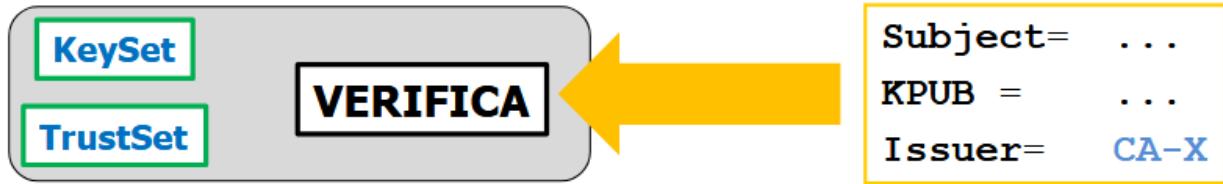
- Ci fidiamo solo dei *certificati* prodotti da *alcuni Certification Authority*, in particolare devono trovarsi all'interno dell'insieme "*Trust Set*" (vediamo dopo cos'è veramente)
- Possiamo eseguire il test di accertamento delle certificazioni solo quelli prodotti da *alcuni Certification Authority*, in particolare devono stare nel "*Key Set*"

Gli insiemi sono delle *strutture dati* predefinite: infatti, ricordiamo che i subject sono "*configurati in modo sicuro*", e quindi vuol dire che gli insiemi Trust Set e Key Set siano "*correttamente*" definiti.

Un software esegue il seguente algoritmo per verificare un certificato:

Algoritmo.

1. CA-x sta nell'intersezione tra Trust Set e Key Set?
2. Se sì, proseguire; altrimenti abortire il programma
3. Effettuare il test sul certificato
4. Se sì, finire e usare il certificato



Osservazione. Notiamo che CA-x non sta in Trust set o Key Set vuol dire che *potrebbe* essere falso o che *non posso effettuarci il test*, ma non vuol dire necessariamente che sia veramente falso o che sia veramente modificato o che non sia autentico.

Osservazione / Richiamo. Se un certificato è *valido* allora si garantisce la seguente catena di associazioni:

$$\text{Subject Fisico} \longleftrightarrow \text{Subject} \longleftrightarrow K_{\text{PUB}}^{(S)}$$

Quindi, ragionando in maniera completa:

- Chi ha richiesto il certificato ha il diritto di usare Subject come identificatore
- La chiave pubblica $K_{\text{PUB}}^{(S)}$ è veramente la chiave pubblica del Subject

Q. Se invece un certificato *non* è valido? Cosa vuol dire?

Vuol dire che uno dei test descritti nell'algoritmo (^8dc091) ha fallito:

1. CA non sta nel Trust Set
2. CA non sta nel Key Set
3. CA sta nel Key Set ma il test ha restituito un esito negativo

Ovvero non ho più la seguente catena di garanzie:

$$\text{Subject Fisico} \not\longleftrightarrow \text{Subject} \not\longleftrightarrow K_{\text{PUB}}^{(S)}$$

In pratica consideriamo la possibilità che il *Subject Fisico* non abbia veramente diritto di identificarsi come *Subject X*.

Q. Cosa fa il software quando riceve un certificato non valido?

Dipende dalla configurazione del software, fa uno tra le due opzioni:

1. Abortisce l'esecuzione del programma (procedimento "*matematico*")
2. Chiede all'utente se procedere, usando il certificato potenzialmente falso (*caso più comune*)

Esempio. Un caso piuttosto comune è quello di vedere dei *certificati scaduti* su HTTPs, e l'utente può scegliere di connettersi col web server nonostante l'avviso del certificato invalido.

Osservazione. La "*non validità*" o "*validità*" di una certificazione è una proprietà relativa alla configurazione del nodo di subject, siccome è possibile definire insiemi Trust Set o Key Set diversi.

2. TrustSet, KeySet e PersonalSet

Vediamo di definire formalmente il concetto di *TrustSet* e *KeySet*. Vedremo inoltre la necessità di definire l'insieme delle "*proprie chiavi*", chiamato *PersonalSet*.

🔗 TrustSet e KeySet

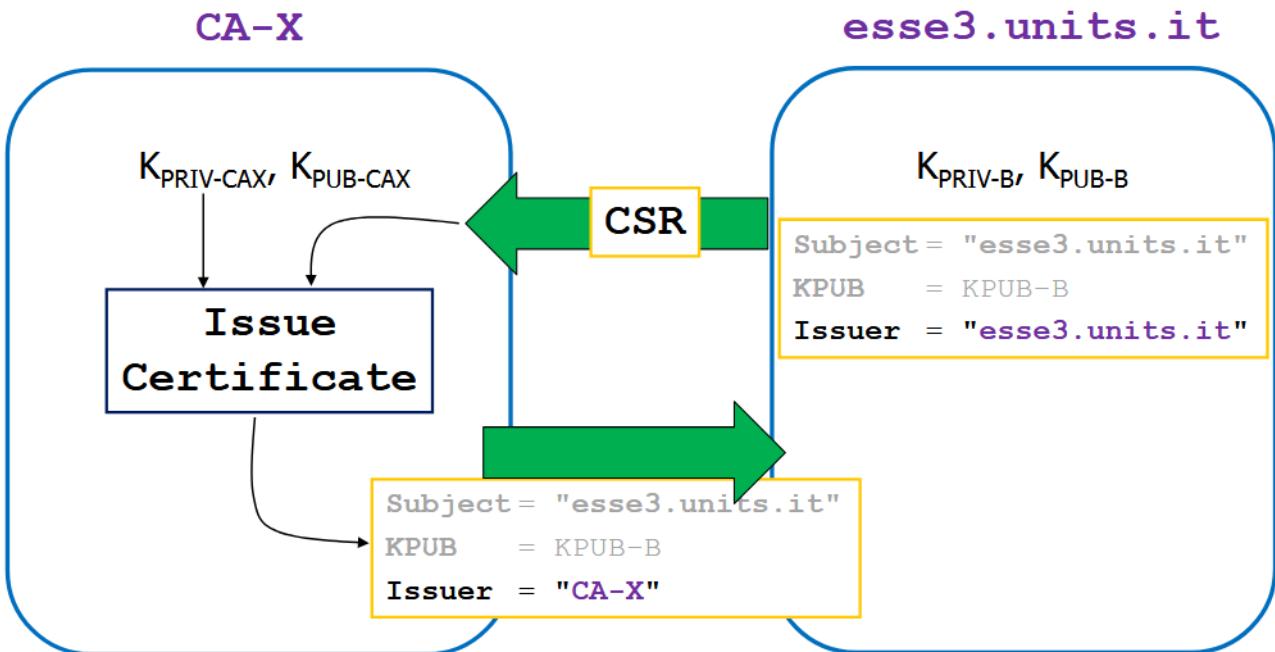
TrustSet e KeySet sono *insiemi di certificati autofirmati* con chiave pubblica.

Capiremo perché siano "*self-signed*" dopo...

Gli insiemi TrustSet e KeySet stanno in una cartella che dipende dal *sistema operativo* o dal *software specifico*. Ad esempio, in openssl si ha un file con una sequenza di certificati autofirmati.

Adesso vediamo cosa vuol dire "*configurazione sicura*" dei subject nuovamente, tornando al caso "*poco comune*" di rilascio certificati:

- Il server genera un *certificato autofirmato* e lo invia al CA
- Il CA effettua dei controlli, e nel caso di esito positivo genera un *certificato* con una sua *chiave privata* (capiremo meglio perché)



Una volta che il certificato viene mandato da parte del Certification Authority, il subject server lo preserva assieme alle chiavi associate in una struttura dati chiamata *Personal Set*.

🔗 Personal Set

Un *personal set* è un insieme delle "*proprie chiavi*", ovvero ogni elemento consiste nella

tripla $(K_{\text{PUB}}, K_{\text{PRIV}}, \text{CERTIFICATE})$.

I *PersonalSet* sono salvati come dei "file sparsi".

Protocollo TLS

X

Applicazione dei strumenti matematici per realizzare un canale di comunicazione sicuro: protocollo TLS. TLS da un punto di vista funzionale: definizione e proprietà. Osservazioni: il lato server dimostra di conoscere la sua chiave privata ma non lo comunica, prima istanza di autenticazione lato server. Esempio di protocollo TLS applicato: HTTPS, aspetti funzionali. Implementazione di TLS: procedimenti, osservazioni.

X

0. Voci correlate

- Introduzione alla Network Security
- Crittografia a Chiave Pubblica
- Crittografia a Chiave Privata e HMAC
- Crittografia a Chiave Pubblica in Pratica
- Certificati e Certification Authority
- Comunicazione tra Processi

1. TLS

Applichiamo tutti gli strumenti matematici appena visti per realizzare un canale di comunicazione sicuro.

1.1. Aspetti Funzionali

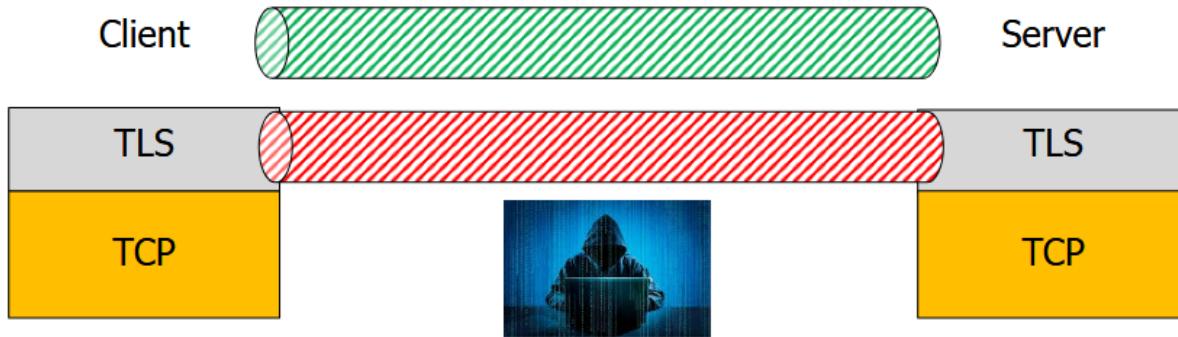
Lo vediamo prima di tutto da un punto di vista "funzionale", ovvero "come si usa" TCP.

TLS (Transport Layer Security)

TLS è un protocollo che usa delle tecniche "crittografiche" che "irrobustire" le connessioni TCP, che a loro volta sono dei *canali di comunicazione vulnerabili*.

Cenni Storici. Inizialmente si chiamava SSL, dopodiché c'erano varie versioni fino ad oggi con TLS.

Intuitivamente, TLS è un "*layer aggiuntivo*" alla pila dei protocolli Internet.



A livello di implementazione, *"basta"* riscrivere i programmi (le librerie) in un modo da poter essere compatibili con TLS. Nei dettagli più specifici, bisogna consultare gli RFC.

Proprietà. Il protocollo TLS è in grado di garantire le seguenti proprietà di sicurezza:

- *Secrecy*: Si usa la *chiave privata simmetrica* per comunicare dati e la si cambia di frequente
- *Integrity*: Se un lato della connessione si *"accorge"* che i dati vengono alterati in transito, si chiude immediatamente la sessione.
- *Authentication (Server)*: Si usa un *certificato* sia per verificare se il *server che comunica abbia veramente diritto di usare il DNS name* e con la chiave privata si dimostra che nel *lato server ci sia veramente il lato server*.

Osserviamo che non si garantisce la *availability*, anzi *"viene di meno"*. Infatti, un possibile attacco è quello di alterare leggermente i bit nella comunicazione in una maniera *"costante"*, impedendo la comunicazione tra i due subject (attacco noto come *"Denial of Service"*).

Osservazione. Se si usano certificati invalidi, crolla *solo* la certezza del DNS-name.

Osservazione. L'attaccante non può impersonare il lato server, infatti deve dimostrare di conoscere la *chiave privata giusta*. Inoltre, notiamo che il *server* dimostra di conoscere la chiave privata, ma non lo comunica! Questo è una differenza dal caso di *client-side authentication*.

Osservazione. Questo è il primo esempio in cui si autentica il *server*.

Esempio. (*HTTPs*)

HTTPs è un primo esempio di protocollo applicativo che comunica su TLS. Da un punto di vista funzionale, il browser:

1. Risolve DNS-name
2. Si connette al server con porta 445
3. Usa funzioni send()/receive() con proprietà TLS

1.2. Aspetti Implementativi

⚠ Argomento Semplificato

N.B. Questo argomento è una semplificazione della realtà, adeguata solo per scopi didattici.

Per vedere l'implementazione di TLS vediamo *quali siano le funzioni* in TLS.

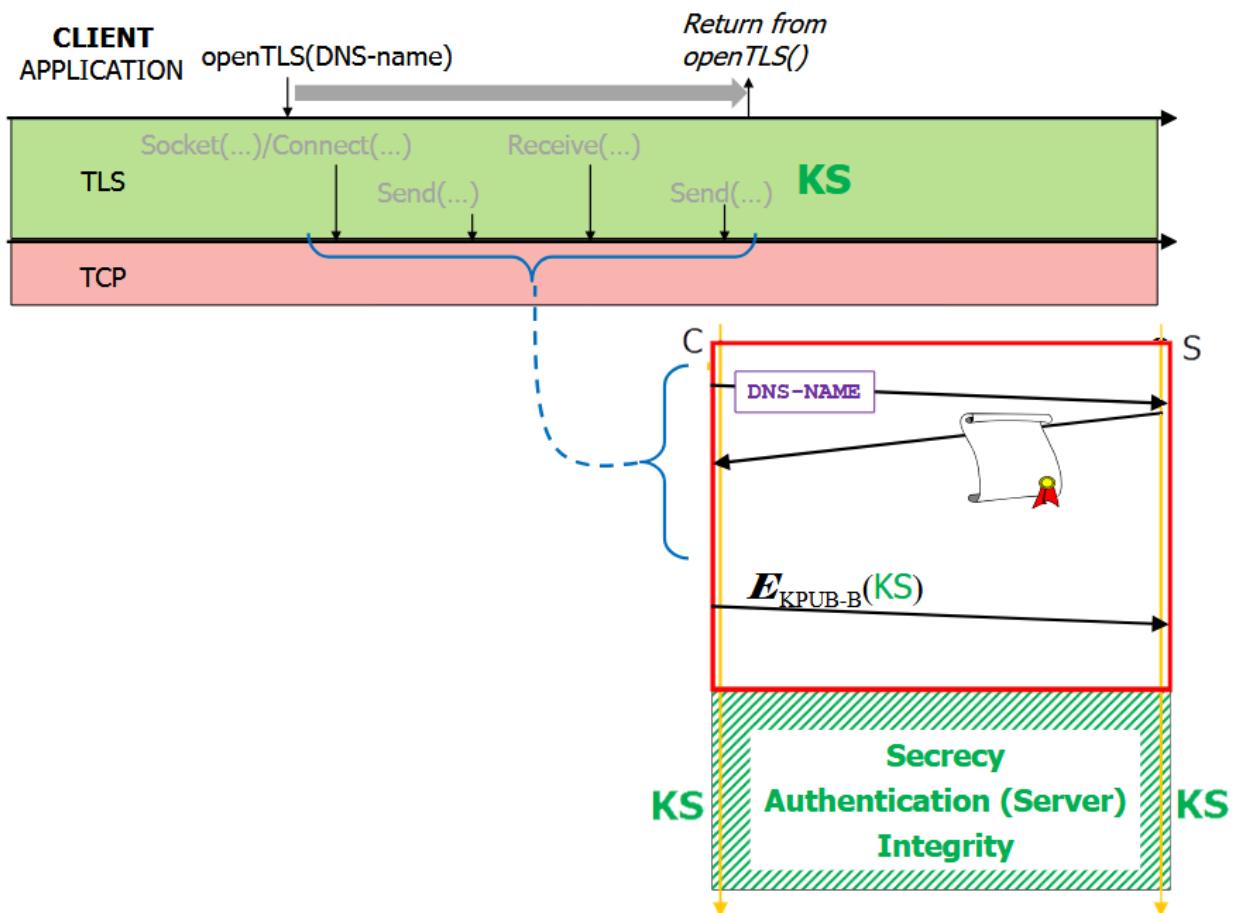
openTLS(DNS-name): Prima di tutto, per aprire una *connessione TLS* si effettuano gli seguenti step:

- Il Client invia il *DNS-name* dato in configurazione
- Server invia il *certificato* associato a *DNS-name*
- Client verifica il certificato e *anche il subject* del certificato (!)
- Client sceglie e critta la chiave simmetrica K_S , lo invia al server

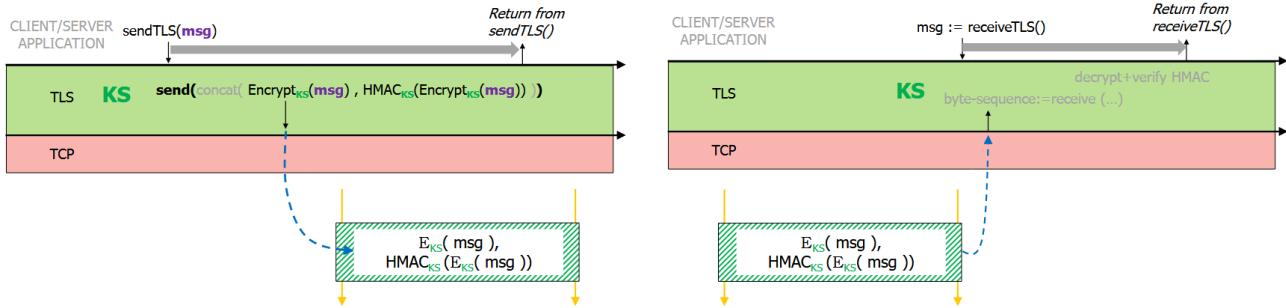
Q. Come mai il lato client deve ri-inviare il DNS-name?

Questo perché il server potrebbe gestire più domini, quindi conoscendo il DNS-name sa quale certificato inviare al lato client. Esempio: web hosting.

Altrimenti, se il client accettasse un qualsiasi certificato valido allora il network attacker "*man in the middle*" potrebbe inviare un certificato valido ma che *non centri nulla* col DNS-name richiesto.



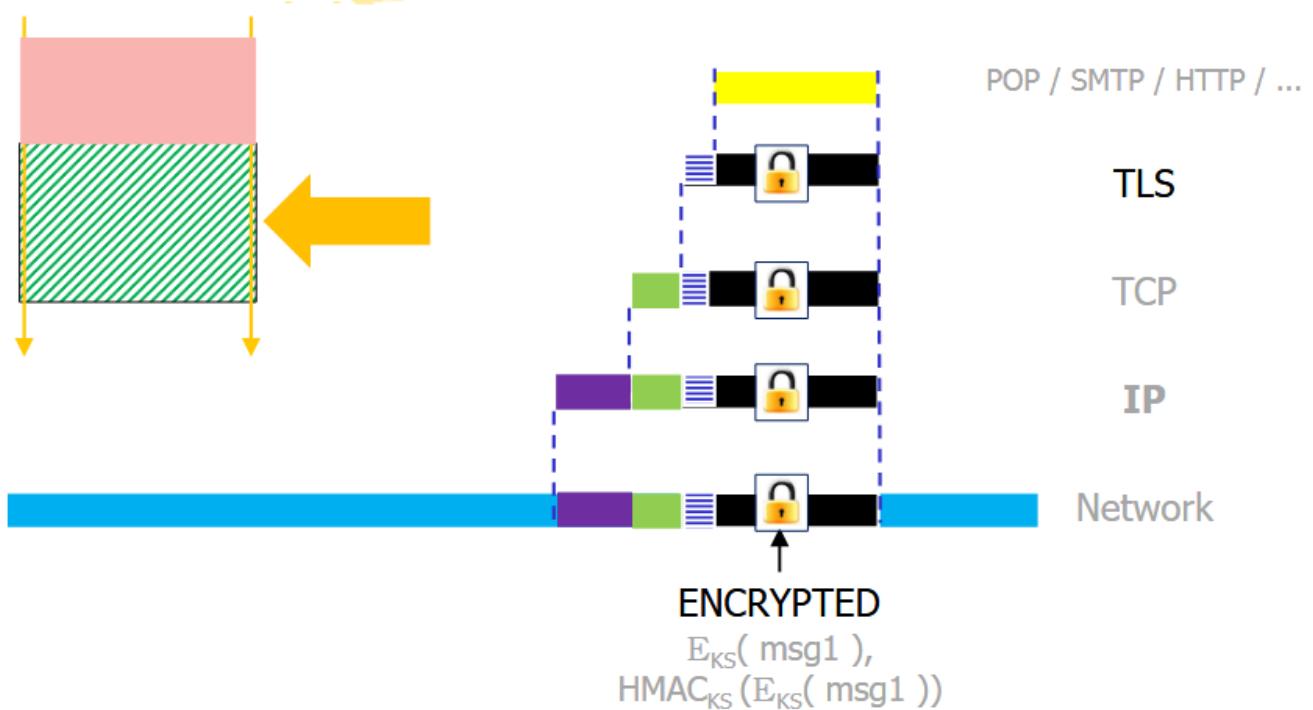
send(m) / receive(m): Una volta eseguito `openTLS(DOMAIN-NAME)`, si può iniziare a trasmettere dati. In particolare, con `send(m)` invia il messaggio crittato con chiave simmetrica e con HMAC, invece con `receive(m)` riceve il messaggio, decrittata il messaggio e verifica i valori HMAC.



Vediamo meglio un paio di aspetti di TLS, essendo consapevoli dell'implementazione:

Osservazione. Il web server conosce di dimostrare il suo "*segreto*" conoscendo la sua chiave simmetrica.

Osservazione. Si ha un ulteriore strato di incapsulamento del pacchetto applicativo, fatto durante la *fase di apertura* della connessione TLS.



✗ Bocciatura Automatica

Soltanto il *payload del protocollo applicativo* è crittato, non tutto il payload del frame network! Se lo fosse crittato tutto, i router non sarebbero grado di capire *come* instradare il pacchetto...

Osservazione. Come funzionerebbe TLS con *payload proxy*? In TLS assumiamo che c'è solo una connessione TLS... mistero!

Osservazione. Una "*buona pratica*" da assumere come proprietà è quello di cambiare la chiave simmetrica ad ogni connessione TLS su TCP, ovvero in ogni connessione TLS dev'essere invocata la funzione `openTLS()` nuovamente, indifferente dal fatto che prima si abbia già aperto una connessione TLS e quindi avere già una chiave simmetrica concordata.

Aspetti pratici di TLS. Difesa contro DNS spoofing. Uso di HTTPs: livello di riservatezza, rubare cookie a siti HTTPs aggirando le difese TLS (crittografia).

0. Voci correlate

- Protocollo TLS
- Aspetti Strategici del DNS
- Protocollo HTTP
- Sessioni HTTP
- Proprietà di Sicurezza

1. Attacchi Significativi TLS

Conviamoci che il *protocollo TLS* funzioni veramente per difendersi dagli network attacker, garantendo le proprietà di sicurezza di *riservatezza, integrità e autenticazione (lato server)*. Lo facciamo presentando un esempio ed esplorando tutti gli esiti possibili

DNS Spoofing. Supponiamo che un *processo client* si colleghi ad un *processo server* via TCP su TLS. Supponiamo che l'RR per cui si identifica il nodo del processo server sia **www.a.com A IP-A**

Adesso supponiamo che il *network attacker* sia in grado di "*falsificare*" la risposta DNS, portando dunque al *processo client* di contattare in realtà **IP-other**.

Cosa può fare, per "*comunicare*" col client?

1. Inviare un certificato con DNS name diverso da **www.a.com**
2. Possedere un "*certificato valido*" per **www.a.com** ma con chiave pubblica del network attacker
3. Possedere un certificato "*proprio*" con issuer *CA-s* che appartiene al *trust set* del client
4. Come 3. ma *CA-s* non sta nel *trust set* del client

Notiamo che ogni tentativo fallisce, infatti:

1. Il client si accorge la discrepanza tra i subject DNS-name, termina quindi la connessione
2. Impossibile, si assume che i certificati prodotti da una CA che sta nel *trust set* siano sempre "*veritieri*"
3. Impossibile, come sopra
4. Fallisce in quanto il client rileva l'invalidità del certificato

Vediamo che quindi con TLS il *browser* è in grado di accertare che stia veramente comunicando col "*proprietario legittimo*" del name server dell'URL contenuto nell'address bar!

Osservazione. Notiamo che comunque rimane una problematica irrisolta legata al *sistema DNS*, ovvero il network attacker potrebbe acquistare un *dominio DNS* con nome "*simile*" a **www.a.com** (come **www.a_.com**) e comprarsi dei certificati validi (che è possibile, siccome in questo caso è vero che NA possiede il dominio DNS "*ingannevole*") e quindi infine riuscire ad ingannare il client di comunicare con **www.a.com**.

Tuttavia, tutte le proprietà di sicurezza funzionano via HTTPS.

X

2. Uso di HTTPS

Vediamo adesso le *implicazioni pratiche* di applicare lo "strato" *TLS* su comunicazione *HTTP*.

Come in HTTP, ogni documento è identificato da un URL, ed il protocol name (la prima parte) diventa **https**. La porta su cui si effettua la comunicazione HTTPS è 443. Fino ad oggi, HTTPS ha "*quasi sostituito*" il protocollo HTTP; si stima infatti che in Europa l'uso dell'HTTPS è previsto nel 95.7% dei casi [1].

Abbiamo visto che HTTPS garantisce le proprietà di sicurezza previste da TLS, ovvero *riservatezza, autenticazione ed integrità*. In altre parole, HTTPS utilizza *tecniche crittografiche* per "*irrobustire*" una connessione TCP, che è invece *vulnerabile*.

2.1. Proprietà di Sicurezza di HTTPS

Vediamo di approfondire bene le proprietà di sicurezza.

Q. Se visito un *server* con un certo nome su *TLS/HTTPS*, è possibile che il Network Attacker sappia della visita?

La risposta è sì, per almeno due motivi:

- Nella fase di apertura della comunicazione *TLS* (ovvero il client invoca la funzione **open(TLS)**) il client deve inviare il *DNS-name* al server
- Il Network Attacker può tracciare le comunicazioni *DNS*, quindi vedere quali nome vuole risolvere il client

Q. Invece per pagine specifiche? E' possibile che il Network Attacker conosca la *local URL* specifica?

La risposta è *no* in questo caso, siccome questa informazione è contenuta solamente nelle richiesta/risposta HTTPS, che sono criptate.

Q. Come mai si deve usare HTTPs anche quando *non sono autenticato*? In questo caso, sembrerebbe fattibile l'approccio di usare HTTP.

In realtà ricordiamo che HTTPs garantisce *integrità* e *autenticazione (lato server)*, quindi il *network attacker* non è in grado di "*portarmi vuole dove vuole lui*". Oppure, è anche importante per *inviare i dati* (quindi prima di prelevare un form)! In questo modo, si evita che il *network attacker* manipoli il FORM facendoci inviare le credenziali a lui, oppure che aggiunga un *keylogger JS* per tracciare i nostri dati.

Quindi vediamo che chiaramente è importante che tutto il traffico sia in *HTTPs*, non solo per prelevare pagine di URL protetti.

2.2. HTTPs e Cookie

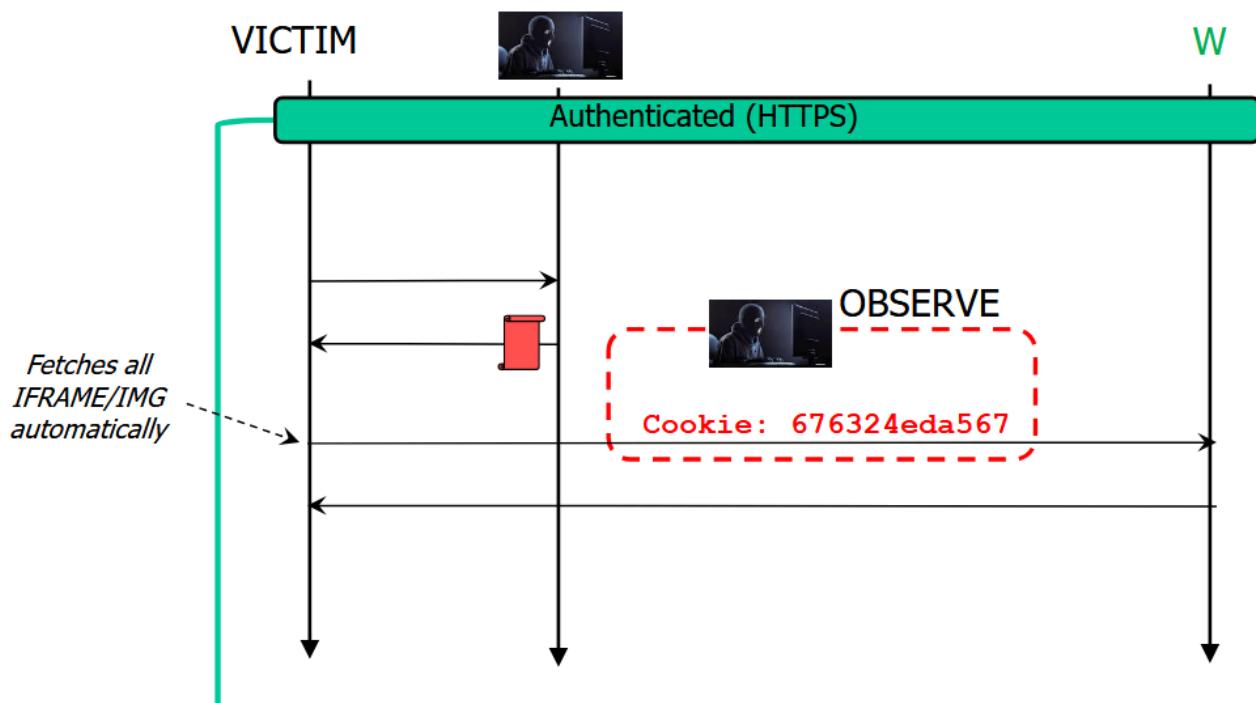
Fino ad'ora abbiamo rivisitato le proprietà di sicurezza garantite da HTTPs, evidenziando la loro importanza. Adesso vediamo un altro aspetto: i *Cookies*.

Supponiamo di avere il seguente scenario:

- Vittima è autenticata sulla webapp W
- W mantiene *sessioni autenticate* su HTTPs

Se il *network attacker* è in grado solo di osservare il traffico, non può far nulla per "*rubare*" il cookie della vittima. Tuttavia, se possedesse un *URL* controllato da lui allora potrebbe essere in grado di rubare il cookie!

La pagina web "*infettata*" del network attacker può contenere un tag *IFRAME*, *IMG* o simile in riferimento ad un URL di W, specificando il *protocollo HTTP*. In questo caso, se la sessione HTTPs è ancora mantenuta allora la vittima visiterà W "*a causa dei tag sul sito web infettato*" ed il gioco è fatto, siccome il network attacker può osservare il traffico HTTP col cookie.



Notiamo che questo è un *problema intrinseco ad HTTP*, ed il network attacker ha aggirato l'ulteriore layer di sicurezza sfruttando problema.

Una possibile difesa su questo attacco è quello di attribuire un ulteriore attributo sui cookie chiamato "*secure*", per cui se è vera allora il *client browser* si impegnerà a non inviarlo mai via HTTP. In altre parole, bisogna estendere ulteriormente il protocollo HTTP.

Nel caso appena visto, se la vittima visita W su HTTP invia effettivamente delle richieste HTTP, però senza il cookie con attributo "*secure*".

Firma Digitale

X

Firma digitale: ulteriore strumento matematico. Contesto, idea algoritmica della firma digitale.

Osservazione: nessuna ipotesi sul canale di comunicazione. Firma digitale in pratica: autenticazione dei subject.

X

0. Voci correlate

- Proprietà di Sicurezza
- Introduzione alla Network Security

1. Contesto della Firma Digitale

Vediamo un altro strumento che applicheremo nell'ambito della *network security*, per implementare le proprieità di sicurezza. Fino ad ora, abbiamo solo assicurato l'*integrità* e *autenticazione* nel contesto *TCP*, ossia le proprietà sono verificabili *solo* dalle estremità delle connessioni e *solo* durante la connessione.

Consideriamo uno scenario diverso; si vorrebbe che l'*integrità e l'autenticazione* di un flusso di byte sia "*permanente*" e "*verificabile da chiunque ed ovunque*", generalizzando sul canale di comunicazione.

Esempio. *File* (referto medico, verbale d'esame, programma esecutibile), oppure una *mail*. Si potrebbe decidere i prelevare questi file in più modi, come:

- Memory pen
- Server con *HTTPs*
- altri mezzi

L'unico modo in cui possiamo usare TLC è la seconda casistica, ovvero server con *HTTPs*. Al massimo, riuscirò a garantire le proprietà di sicurezza rispetto al *browser*.

Per dare una descrizione più generale, abbiamo:

1. *Subject A* crea una sequenza di byte *B*, non indirizzata a nessuno in specifico
2. *B* viene spostato nello *spazio* e nel *tempo*
3. Dopo una quantità arbitraria di tempo chiunque può verificare l'*autenticità* e l'*integrità* di *B*, indipendentemente dal suo percorso di spazio o tempo.

X

2. Firma Digitale: Idea

Lo strumento matematico per lo scenario appena descritto è la *firma digitale*.

IDEA. Si ha un *algoritmo a due parti*, ove:

1. Chi apporta firma ha *chiave privata e pubblica*
2. Chi verifica la firma ha la *chiave pubblica*

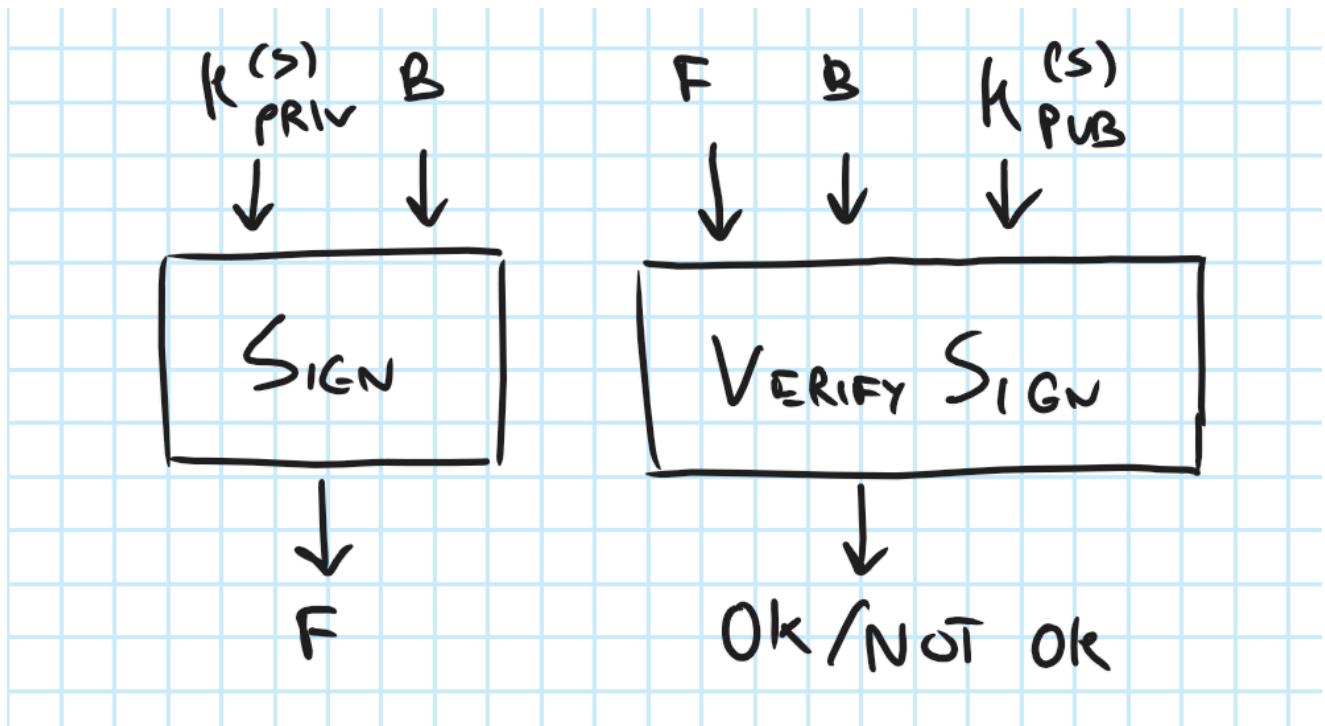
L'algoritmo di *firma* accetta B e viene parametrizzata secondo la chiave privata $K_{\text{PRIV}}^{(x)}$, ed in output ritorna una sequenza di byte a *lunghezza costante* (tipicamente 128 byte). D'altronde, l'algoritmo di *verifica* delle firme accetta in input *il flusso di byte e la firma* ed è parametrizzata secondo la chiave pubblica $K_{\text{PUB}}^{(x)}$, ed in output esce un bit (valore booleano) per indicare se la verifica è andata a buon fine o meno.

Usiamo le seguenti notazioni

- $F := \text{SIGN}_{K_{\text{PRIV}}^{(x)}}(B) \mapsto 2^N$
- $\text{VERIFYSIGN}_{K_{\text{PUB}}^{(x)}}(B, F) \mapsto \{0, 1\}$

In particolare se la verifica:

- Ritorna OK, allora la firma è *autentica ed integra*
- Ritorna NOT-OK, allora la firma o non è autentica o non è integra

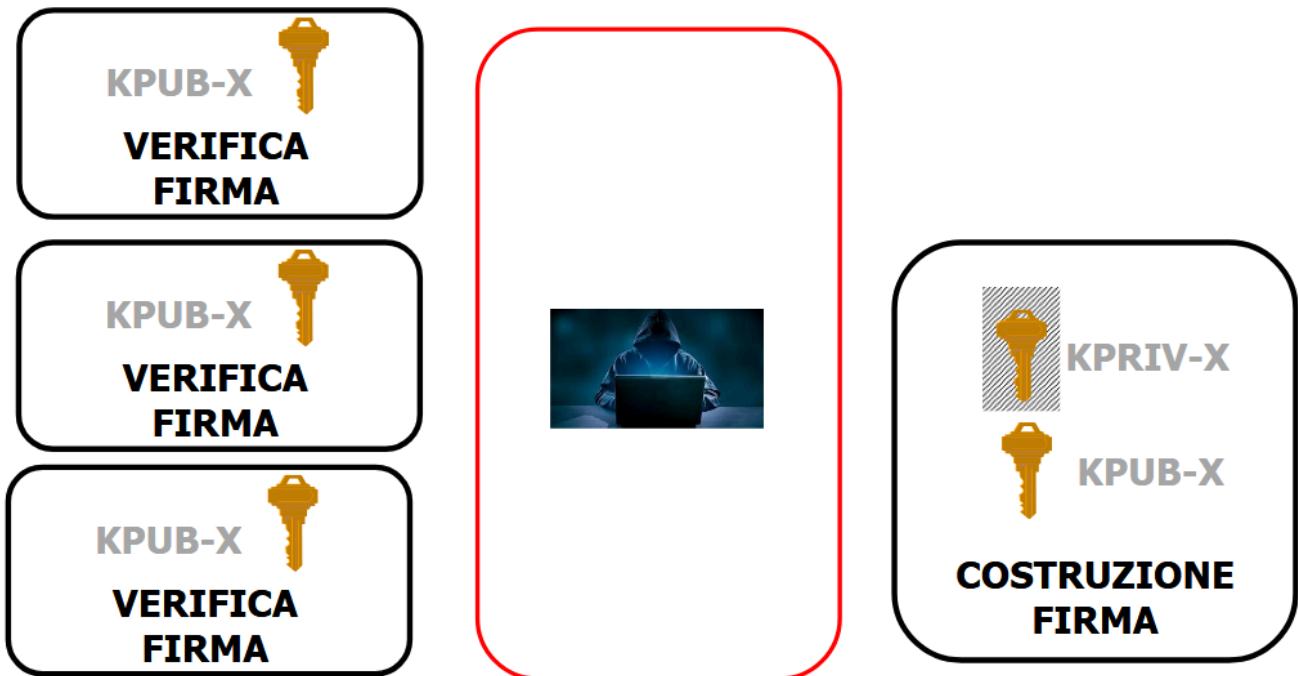


Per garantire la proprietà di integrità ed autenticazione, si dimostra che:

- Calcolare la *firma "corretta"* senza la chiave privata è "*praticamente impossibile*"
- Dedurre la *chiave privata* dalla *firma* o dalla *sequenza di bytes* è "*praticamente impossibile*"

Osservazione. Non vedremo cosa c'è dentro l'algoritmo nello specifico, vediamo solo "*come si usa*" la firma digitale. A livello implementativo, accenniamo che d'entro c'è la crittografia a chiave pubblica ed "*altro*".

Osservazione. L'applicazione delle firme digitali richiede uno scenario di partenza in cui il subject "*verificante*" possiede la chiave pubblica del subject "*firmatario*".



Osservazione. Chiunque può verificare, invece solo chi conosce la chiave privata può costruire la firma digitale. Questo ovviamente per garantire la proprietà di *autenticazione*

Osservazione. Fino ad ora, la *firma digitale* non richiede nessuna ipotesi sul canale di comunicazione! Al massimo, richiede la distribuzione delle chiavi pubbliche (vedremo dopo com'è realizzata). Quindi funziona anche su HTTPS, SMTP, flussi di byte "*consegnati a mano*" ed eccetera...

Notiamo che quindi il *flusso di byte* e la *firma* possono arrivare in ordine diverso, garantendo le proprietà di sicurezza lo stesso!

Aspetti Pratici della Firma Digitale

X

Aspetti pratici della firma digitale. Ulteriore requisito di proprietà di sicurezza: autenticazione sul subject fisico. Modalità di utilizzo pratiche delle firme digitali. Applicazione delle firme digitali. Considerazioni pratiche sulle proprietà di sicurezza garantite. Esempi di ricapitolazione.

X

0. Voci correlate

- Firma Digitale
- Certificati e Certification Authority

1. Autenticazione Subject Fisico

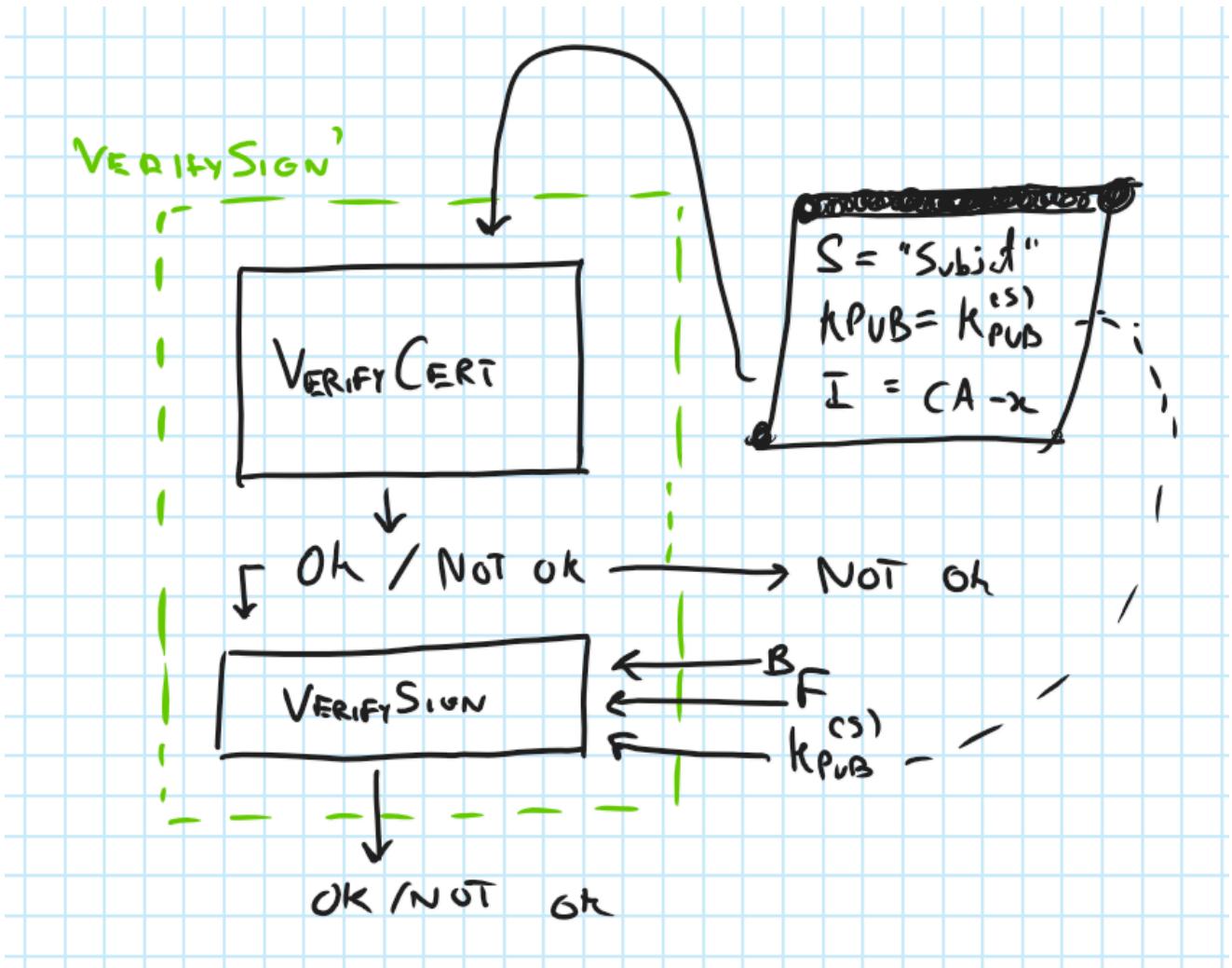
Vediamo la *firma digitale in pratica*.

Notiamo subito che bisogna richiedere anche l'*autenticazione del subject*, ovvero assicurarsi che un subject fisico abbia veramente diritto di utilizzare un certo subject.

In questo caso useremo i *certificati*, ovvero oltre ad inviare una firma per assicurare l'*integrità* ed *autenticazione* bisogna anche inviare il *certificato*.

Quindi in realtà la *verifica* delle firme digitali avviene come segue:

- Verifica il *certificato*, quindi assicurarsi che l'issuer del certificato sia nel *Trust set* e nel *Key set* e usare la "*bacchetta magica*" per verificare che sia effettivamente autentico ed integro
- Verificare la firma con la *chiave pubblica* nel certificato



Osservazione. Questo approccio ci ha appena risolto uno dei problemi di prima, ovvero la necessaria ipotesi della distribuzione di chiavi pubbliche. Usando i certificati, oltre a garantire l'associazione Subject Fisico e Subject, siamo anche in grado di specificare la *chiave pubblica*.

Osservazione. Se il certificato è valido e la verifica della firma ha successo, allora vuol dire che la firma F costruita da S è veramente costruita da S , non è stata modificata e S è veramente chi penso che sia

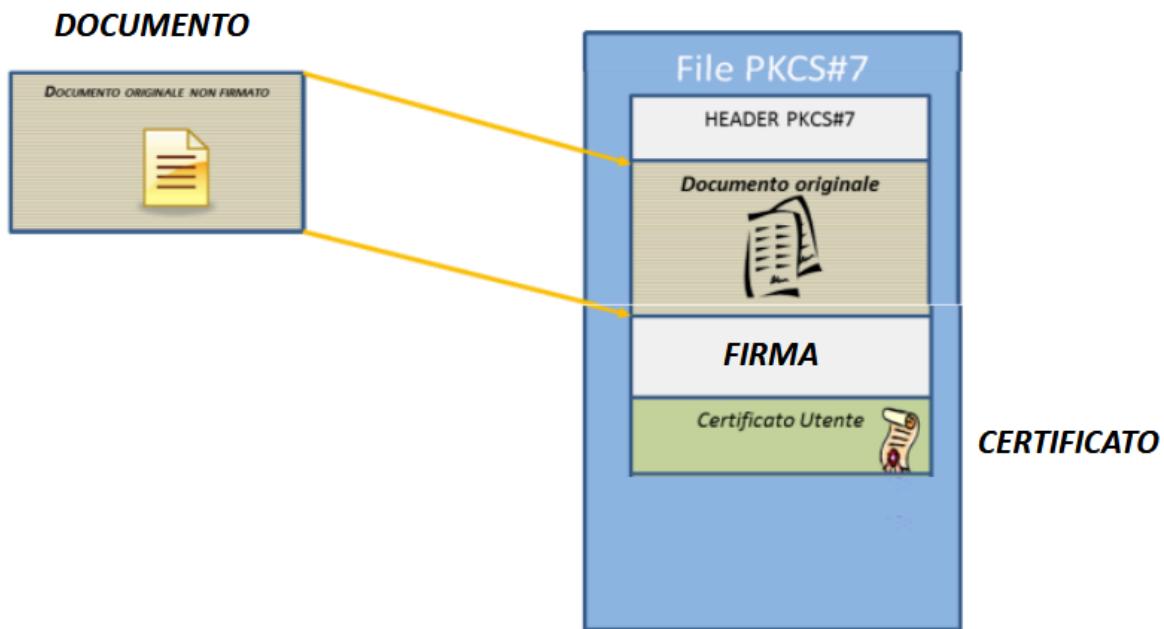
Se invece il *certificato non è valido* ma la *firma è comunque verificata*, vuol dire che F è costruita da S e non è mai stata modificata, tuttavia non posso garantire chi sia veramente S ! Quindi non mi fido dell'associazione tra subject fisico e subject...

X

2. Modalità di Utilizzo Pratiche

Vediamo le principali modalità di utilizzo pratiche della *firma digitale*.

Innanzitutto è comune che il *documento B*, la *firma digitale F* e il *certificato cert* siano concatenate in un'unica sequenza di byte per questioni di comodità. Saranno quindi necessari dei *standard* per descrivere la struttura della sequenza, come ad esempio *PKCS7* e *p7m*.



Questo avviene molto più comunemente dei *file PDF* firmati digitalmente.

Per quanto riguarda l'*apposizione* e la *verifica* delle firme digitali, si ha che essi o sono delle *funzionalità integrate in programmi non specifici* o essi sono funzionalità di *programmi specifici*.

Come esempio di "*programma non specifico*", si ha *Word*, *Excel*, *PDF readers* e solitamente accettano il documento unico con documento, firma e certificato. Inoltre, deve leggere il *personal set* per apportare firme e leggere il *Key Set / Trust Set* per verificare le firme.

Invece, alcuni programmi come *openSSL* sono "*fatti apposta*" per creare e verificare le firme digitali, e hanno moltissime varianti per file con documento, file e certificato.

Osservazione. Facciamo un paio di osservazioni terminologiche nel linguaggio comune

- Dire "*firmare col certificato*" è errato, siccome in realtà si usa la *chiave privata* per apporre una firma digitale
- Dire "*acquistare/creare la firma digitale*" è altrettanto sbagliato, in quanto non esiste "*la*" firma digitale, in realtà si intende l'acquisto della *coppia delle chiavi assimetriche* e *certificato con validità legale* per costruire la firma digitale.

Osservazione. (praticamente) In tutti i paesi hanno norme sugli *algoritmi*, *CA*, *distribuzione/revoca delle chiavi* per dargli una "*validità legale*", ovvero un file PDF con firma digitale diventa "*equivalente*" ad un documento cartaceo firmato. Naturalmente ci sono molti requisiti, e in Italia si prevede un albo delle *Certification Authority* con "*validità legale*". Non approfondiremo questo discorso.

Le firme digitali ritrovano applicazione in *moltissimi ambiti*.

- Come primissimo esempio le firme digitali sono utili per le *applicazioni su documenti con validità legale*, come referti medici o verbali degli esami.
- Anche importante per gli *software eseguibili*, infatti un *software* creato da uno sviluppatore effettua un *"salto temporale/spaziale"* lunghissimo! Immagina quanti router attraversa il software...
Pertanto è importante accertarsi che un'app scaricata sia sviluppata veramente dal developer del software. Nella pratica, vengono firmati o *dal sviluppatore software* o dal *gestore dello "app marketplace"*
 - Un sottoesempio più specifico è l'aggiornamento dei *sistemi operativi*; immagina cosa potrebbe fare un *network attacker* che riesce ad operare sui *"server di passaggio"*!

X

4. Considerazioni sulle Proprietà di Sicurezza

Facciamo delle ulteriori considerazioni sulle proprietà di sicurezza garantite dalla *firma digitale*.

L'*autenticazione* e *integrità* non implicano nessuna nozione sulla *"verità"* del contenuto di *B*. Per convincerci di questo, facciamo un paio di esempi:

- La stringa *"Gli asini volano"* è certamente un'affermazione falsa, ma può essere comunque *firmata digitalmente* con chiave privata e verificata con chiave pubblica
- Ancora più assurdamente, una delle stringhe tra *"Dio esiste"* e *"Dio non esiste"* è vera ma entrambe possono essere comunque *firmate digitalmente*

Di conseguenza, applicando alle applicazioni pratiche abbiamo che:

- I documenti con *validità legale* non sono garantiti ad essere completamente veritieri
- I software con firma digitale possono avere *malware*, *backdoor* o *bug* per più motivi.
Ragionando in negativo, un software *non firmato* può essere begnino e bug free.

Inoltre le proprietà di *autenticazione ed integrità* non bastano per alcune applicazioni.

Facciamo un esempio importante:

Esempio. Un file PDF è firmato e generato allo stesso istante di tempo *T*. Tuttavia, con la sola firma digitale è impossibile garantire che il file PDF sia veramente firmato nell'istante di tempo *T*, siccome la *"timestamp"* viene salvata solo nel contenuto *B*.

Pertanto richiediamo ulteriori *proprietà di sicurezza*, nel nostro caso possiamo risolvere istituendo le *"autorità temporali"* che generano delle *marche temporali firmate*.

X

5. TLS e Firma Digitale

Supponiamo che un browser preleva un documento D senza firma digitale con HTTPs. Quanti messaggi HTTPs sono firmati? Nessuno, ovviamente.

E invece se il documento fosse firmato? Anche qui, dire che i messaggi HTTPs sono firmati è un *errore grave* siccome non sono firmati in nessun modo. I certificati vengono usati in un'altra maniera, certamente non per apporre/verificare firme digitali!

Recap Security

X

Esempi di recap sulla network security.

X

0. Voci correlate

- Introduzione alla Network Security
- Firma Digitale
- Certificati e Certification Authority
- Crittografia a Chiave Pubblica
- Crittografia a Chiave Privata e HMAC

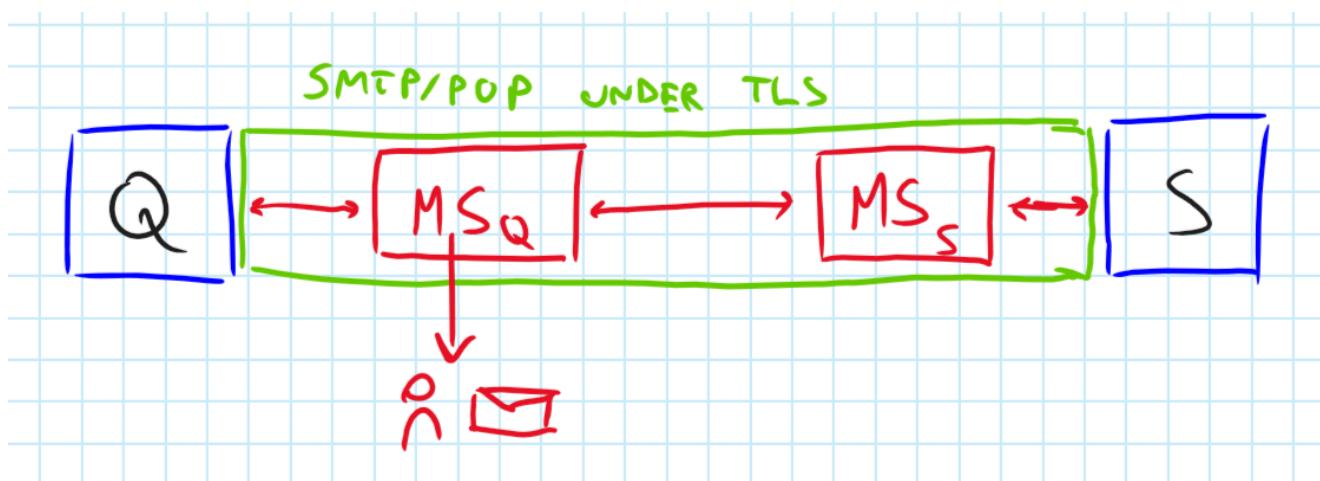
1. Recap Crittografia a Chiave Pubblica e Privata

Adesso facciamo un paio di esempi di ricapitolazione, per avere un modello mentale sulla *network security*.

Esempio. Supponiamo che la *stazione di polizia* comunichi con la *questura* via *mail*.

Supponiamo che il *network attacker* operi solo sul canale di comunicazione TCP; allora per garantire le proprietà di sicurezza, è sufficiente usare applicare *TLS* su *SMTP/POP*.

Se invece il network attacker operasse anche sui *mail server "intermedi"*? In questo caso le difese garantite da TLS crollerebbero, siccome per il network attacker è sufficiente sfruttare i *mail server* per osservare/modificare le mail inviate/ricevute.



In questo caso bisogna applicare un ulteriore strato di sicurezza aggiuntivo. Un esempio è come segue:

1. Questura e stazione di polizia si concordano una *chiave privata* tra loro (o usando *certificati validi* o *usando un canale di comunicazione sicuro aggiuntivo*)

2. Crittare le mail con la *chiave simmetrica* e applicare l'*HMAC*

Notiamo che è sufficiente che *solo uno dei due lati* abbia la coppia chiave privata-pubblica. Anche se entrambi avessero la coppia delle chiavi private-pubbliche, sarebbe comunque consigliabile eseguire il procedimento di prima invece di comunicare direttamente con le coppie di chiavi in quanto sarebbe troppo costoso e mancherebbe anche l'HMAC, che richiede una *chiave simmetrica*.

Supponiamo che adesso il network attacker diventi un *node attacker* e può controllare i calcolatori alle due estremità. Allora anche in questo caso le difese crollerebbero e purtroppo non può essere risolta con *solo crittografia*, infatti il *node attacker* è in grado di leggere le chiavi simmetriche sui nodi, rendendola inutile.

Osservazione. Network attacker con "*poteri*" diversi implica meccanismi di sicurezza diversi. Ciò implica che quindi non possiamo essere certi su come si comporta l'attaccante e dobbiamo fare delle ipotesi "*sensate*", infatti:

- Se "*sovraстимамо*", la realizzazione delle proprietà di sicurezza diventa molto costosa
- Se "*sottovalutiamo*", allora il network attacker è in grado di penetrare le difese

Nel corso assumiamo che il network attacker sia in grado *solo* di operare sul canale di comunicazione vulnerabile.

X

2. Recap Firma Digitale

Consideriamo un altro esempio.

Esempio. Adesso consideriamo un nuovo scenario. La *stazione di polizia* vuole inviare un *file audio* alla *questura* con garanzia di *autenticazione ed integrità*.

Si firma il file audio, e per farlo bisogna supporre che la *stazione di polizia* abbia la *coppia di chiave pubblica-privata* e un *certificato valido*.

Quindi la stazione di polizia *P* invia alla questura *Q* la *firma*, il *file audio* e il *certificato valido* con chiave pubblica. Quando *Q* riceve tutto, esegui gli seguenti step:

1. Verifica il certificato, ovvero se l'issuer sta nel Trust Set, Key Set e nel caso positivo verifica l'autenticità e l'integrità del certificato
2. Controlla se la firma è valido o meno

Osservazione. La chiave pubblica della *Certification Authority* non viene usata nello secondo step, infatti serve solo per verificare i *certificati*.

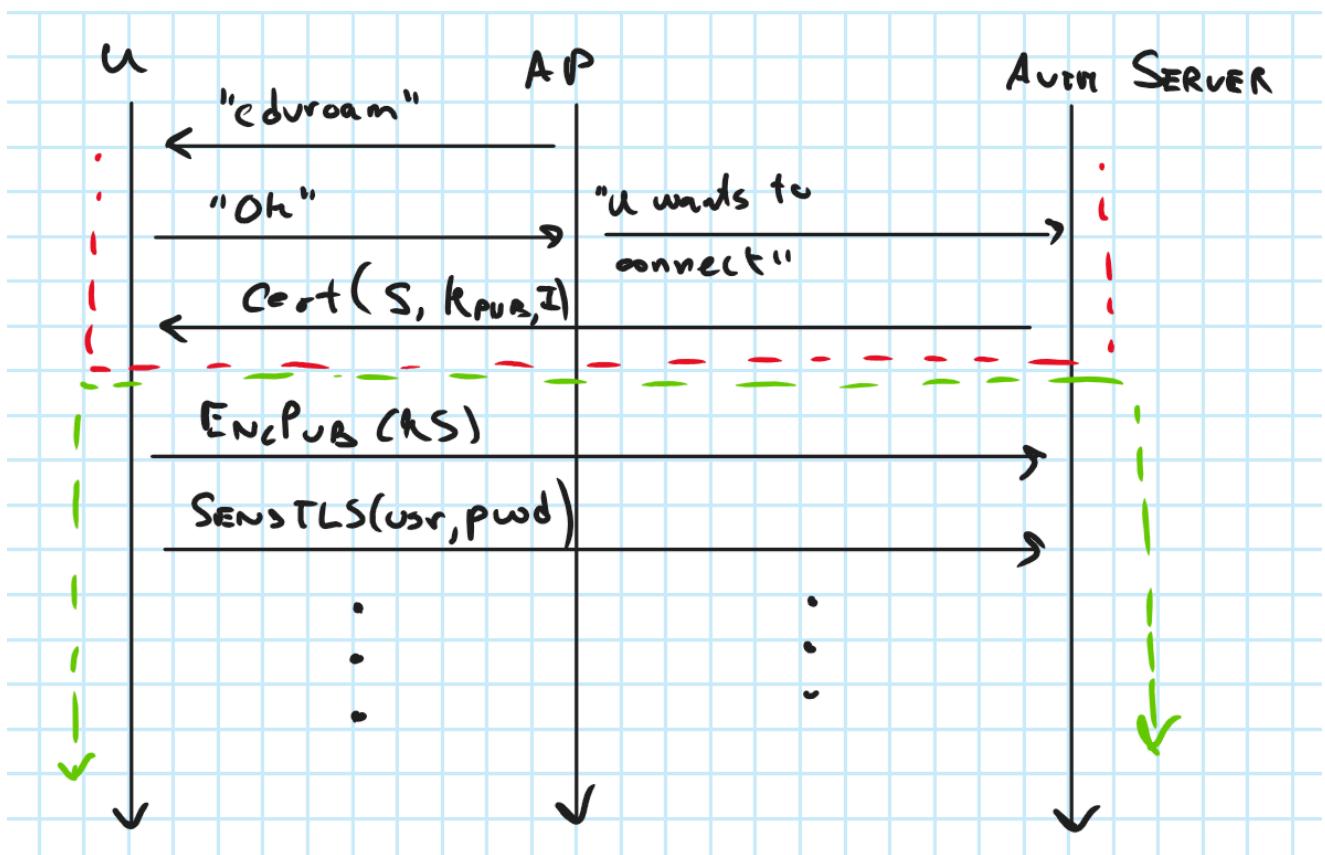
X

3. Curiosità Eduroam

Facciamo un ultimo esempio, come *curiosità*.

Esempio. Prendiamo uno scenario "*reale*". Un utente vuole collegarsi a *eduroam* mediante un'access point dell'università; quello che succede è che:

1. L'access point invia messaggio "*eduroam*" all'utente per dire che deve collegarsi
2. L'utente si collega, e l'access point segnala al *server di autenticazione* che l'utente vuole collegarsi
3. Ovviamente la coppia username-password non può viaggiare in chiaro nella comunicazione tra *user* e *Auth server*; quindi dovranno concordare una chiave simmetrica tra loro per garantire le proprietà di sicurezza nella comunicazione
4. L'*auth server* invia un *certificato valido* all'utente per dire che è il server di autenticazione a cui mandare le credenziali
5. L'utente lo accetta e quindi concorderà la chiave simmetrica da cui comunica con le funzionalità TLS



Osservazione. In mancanza di configurazione, l'utente non può sapere chi è il *subject* (che certamente non è eduroam)! Infatti, il seguente attacco è possibile:

1. Il Network Attacker "*funge*" da Access Point di eduroam
2. Invia un certificato valido con *subject "comprato"* (quindi che non sia veramente dell'Auth server)
3. L'user dovrebbe verificare il subject, ma essendo che non lo conosce (in quanto manca configurazione) lo accetta lo stesso

4. L'user comunica col network attacker

Morale della favola, è importante anche *configurare correttamente* eduroam, per evitare attacchi del genere.

Validazione dei Certificati

X

Validazione dei certificati "senza bacchetta magica": implementazione. Trust Set e Key Set in pratica.

X

0. Voci correlate

- Distribuzione e (cenni) Validazione dei Certificati
- Certificati e Certification Authority
- Crittografia a Chiave Pubblica in Pratica
- Firma Digitale

1. Validazione dei Certificati

Richiami. Ricordiamo che dato un *certificato* con la tripla *subject, chiave pubblica e issuer (certification authority)* il software esegue gli seguenti passaggi per verificare la *validità, integrità e autorizzazione del certificato*.

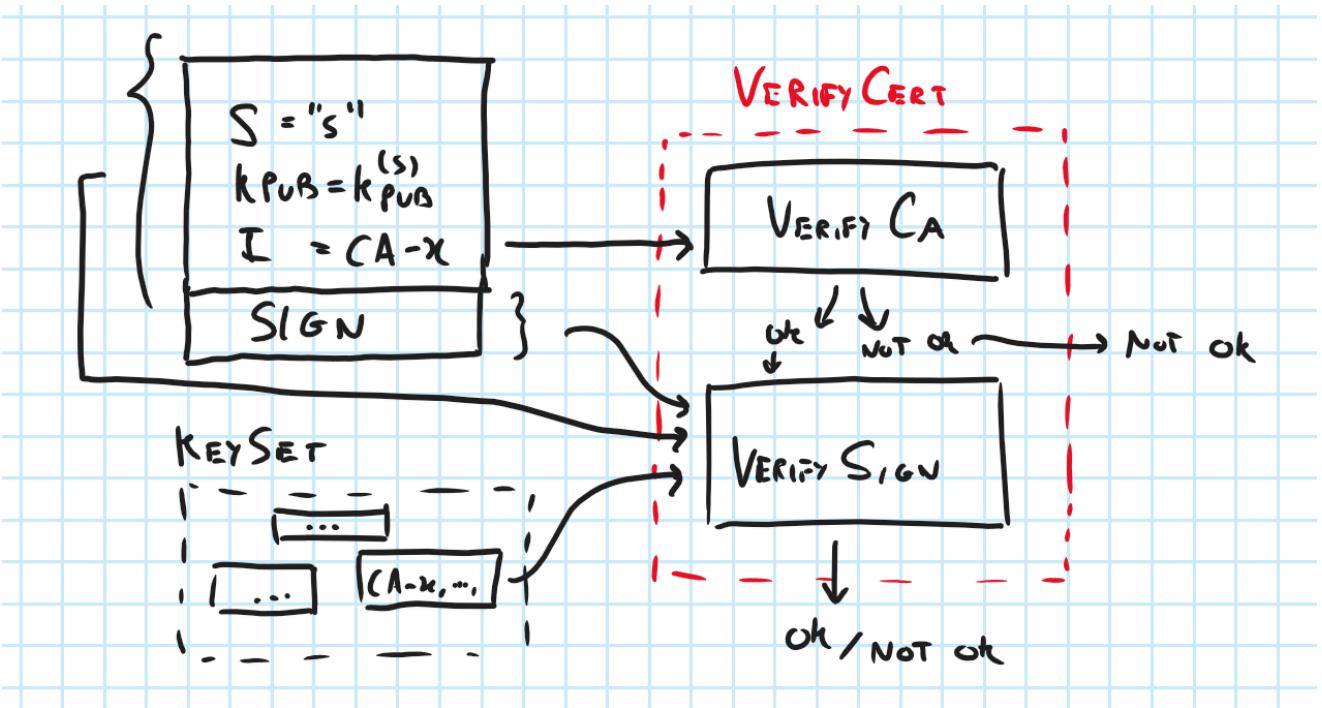
1. L'issuer appartiene al *Trust Set* o al *Key Set*? Se sì, continuare; altrimenti smettere
2. Verificare che il certificato sia *autentico ed integro* con uno "*strumento magico*" (!)

Adesso vediamo come realizzare lo step 2, ovvero verificare che un certificato sia *autentico ed integro*. Per farlo, basta applicare un ulteriore passaggio sui *certificati*: bisogna apporre una *firma digitale* e verificarla.

Tuttavia questo approccio comporta in sè un problema: per verificare la firma bisogna avere una *chiave pubblica*, da dove viene? La risposta sta nel *Key Set*, che era l'insieme che avevamo descritto come "*l'insieme delle CA su cui possiamo eseguire il test magico*".

Con l'ottica delle firme digitali, adesso possiamo capire bene *come* rendano possibile il test di integrità e autenticazione.

Riassumiamo adesso il "*vero procedimento*" col seguente diagramma.



Osservazione. "CA sta nel Key Set" \iff "Ho la chiave pubblica di CA?"

Osservazione. Per *validare* certificati uso solo il *test set e key set*, invece per *verificare firme* uso *B* e *chiave pubblica*, ma non c'entra il *Key Set*! Infatti, Il Key Set si usa solo per verificare il certificato da cui trago la *chiave pubblica*. Quindi, dopo la verifica del certificato non uso più *Key Set*.

X

2. Trust Set e Key Set in Pratica

Facciamo un paio di *considerazioni pratiche finali* sul *trust set* e sul *key set*.

2.1. Aspetti Strategici del Trust Set

Q. Cosa succede se un *certification authority* CA-x generasse *avvolte* dei certificati falsi?

Vorrebbe dire che non ho più nessuna garanzia della verità dei suoi certificati e quindi delle associazioni chiavi pubbliche-subject.

Pertanto, *inserire CA-x* in *Trust Set* vuol dire avere *un potere enorme sui calcolatori!* Ogni affermazione presa da CA-x viene presa per versi...

Se invece inseriamo *CA-x* in più *Trust Set*, allora ha ancora un potere più grande.

Facciamo alcuni esempi che sottolineano proprio la "*potenza strategica*" delle CA in Trust Set.

Esempio. In Kazakistan ogni nodo è obbligato ad avere un *certificato del governo in Trust Set*, ciò vuol dire che il governo può intercettare ogni comunicazione e "*impersonare*" chiunque [2]

Esempio. A causa della guerra in Ucraina (o "*operazione militare speciale*", a interpretazione libera...) e quindi delle sanzioni imposte, le *organizzazioni in Russia* non riescono più a pagare il rinnovo dei certificati. Come soluzione, il governo russo ha proposto di creare un *certificato nazionale* che vada a "*sostituire*" i certificati necessari. Notiamo che questa manovra funziona solo per "*nodi all'interno in Russia*", invece all'esterno rimangono "*inaccessibili*". [3]

Esempio. Nel 2023 l'Unione Europea aveva proposto una legge per cui ogni *governo in UE* può avere un certificato per tutti i browser che operano nel territorio UE. Questo ha attratto molti criticismi, come ad esempio da parte di Mozilla. [4]

Q. Come possiamo sapere che una CA-x in Trust Set dica sempre la verità?

Non si può, è un'assunzione data per scontata. Notare che tutto il sistema della crittografia asimmetrica si regge su questo presupposto "*irrealistico*".

2.2. Trust Set e Key Set in Pratica

Concludiamo il discorso considerando il *trust set* e *key set* in pratica.

Nella crittografia, ci sono tre *insiemi fondamentali*:

- Personal Set
- Trust Set
- Key Set

In astratto, sono degli insiemi "*unici*" e "*separati*"; tuttavia, nella realtà questi insiemi potrebbero:

- Avere nomi diversi
- Alcuni file possono contenere un "*mescolamento*" di elementi di Personal Set, Trust Set, Key Set, quindi gli insiemi possono essere mescolati tra loro
- Essere specificati con formati diversi

Solitamente, si trovano in *files*.

Esempio. Su Windows c'è una cartella predefinita e viene usata dalle "*applicazioni del sistema operativo*". Invece, le applicazioni "*esterne*" hanno solitamente (ad oggi) una loro cartella dedicata.

Q. Come bisogna proteggere gli insiemi Key Set / Trust Set?

Naturalmente li proteggiamo *solo* in scrittura, invece non serve proteggerli *in lettura* (anzi, non bisogna!) siccome dovranno essere letti dai software.

Q. Come arrivano i certificati? Ovvero, da dove vengono?

Arrivano *già installati nei sistemi operativi* o durante la fase dell'*installazione del programma*.

Quindi questa fase è *molto critica!* Infatti, stiamo decidendo chi mettere nel Key Set ma soprattutto nel Trust Set!!!

Ognugno può generare certificati autofirmati e metterli nel Trust Set / Key Set, quindi non avrebbe senso verificare i *certificati self-signed*.

Pertanto, bisogna assumere che ci sia un *canale di comunicazione aggiuntivo sicuro* tra il *fornitore del S.O.* e il *S.O.* stesso.

Per concludere, ripetiamo la citazione di R. Needham sulla crittografia:

"Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it"

X

1. https://research.mozilla.org/files/2025/03/the_state_of_https_adoption_on_the_web.pdf ↵
2. <https://www.itpro.com/network-internet/34051/kazakh-government-will-intercept-the-nation-s-https-traffic> ↵
3. <https://medium.com/coinmonks/russia-establishes-its-own-tls-certificate-authority-to-avoid-sanctions-a8221b72b729> ↵
4. <https://last-chance-for-eidas.org/> ↵