

Reti di calcolatori

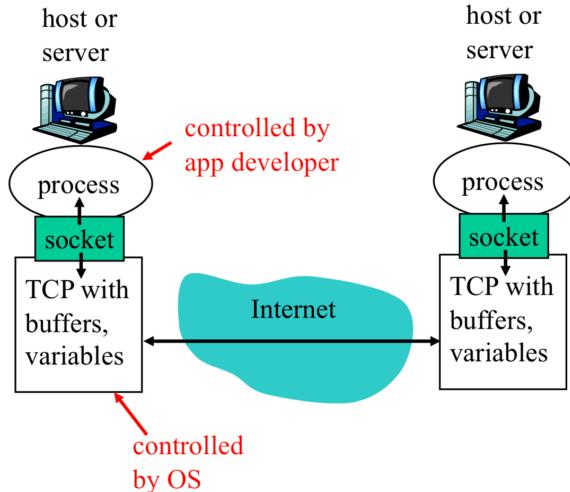
Application layer

Si occupa di fornire supporto alle applicazioni di rete, consentendo di interpretare i dati ricevuti e gestendo il dialogo tra i dispositivi.

Processi che girano sullo stesso dispositivo comunicano tramite meccanismi definiti dall'OS, processi che girano su host diversi comunicano tramite i protocolli definiti dall'application layer.

Per far questo utilizzano i **socket**¹ che sono come delle porte di accesso al sistema.

Lo **user agent** è l'interfaccia tra l'utente e l'applicazione (ex. Google chrome).



Un app-layer protocol definisce:

- Tipo del messaggio => request / response
- Sintassi messaggio => delinea i campi del messaggio
- Semantica del messaggio => significato delle informazioni

Requisiti delle applicazioni:

- Data loss (perdita dati)
- Throughput (banda)
- Timing (ritardo)
- Security

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	loss-tolerant	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Inoltre è importante fare distinzione tra le app che usano il protocollo UDP e TCP per la strato di trasporto

¹ A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

HTTP (HyperText Transfer Protocol)

Protocollo utilizzato per il web, utilizza TCP ed è stateless, ovvero non mantiene informazioni sulle richieste precedenti del client.

Di 2 tipi:

- **Non Persistent HTTP (1.0)** => ogni scambio di messaggi (request, response) richiede apertura e chiusura del canale.

Ogni richiesta necessita quindi di 2 RTTs² (uno per l'invio del messaggio ed uno per stabilire la connessione).

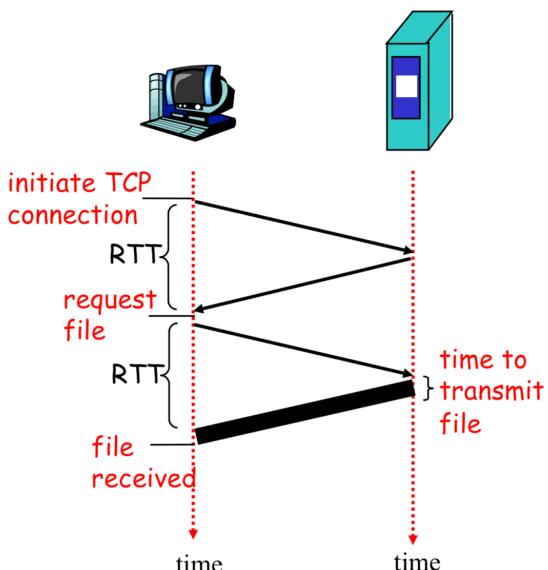
- **Persistent HTTP (1.1)** => mantiene il canale aperto fino alla chiusura da parte del server, permettendo lo scambio di più messaggi sulla stessa connessione.

Si ha un solo RTT di apertura del canale per tutti i messaggi, quindi ogni messaggio successivo necessita di un solo RTT

Esempio trasmissione non persistente (scambio 10 messaggi)

OSS se fosse persistente il punto 1 andrebbe eseguito una sola volta

- 1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80
- 1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client
2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index
3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
4. HTTP server closes TCP connection.
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
6. Steps 1-5 repeated for each of 10 jpeg objects

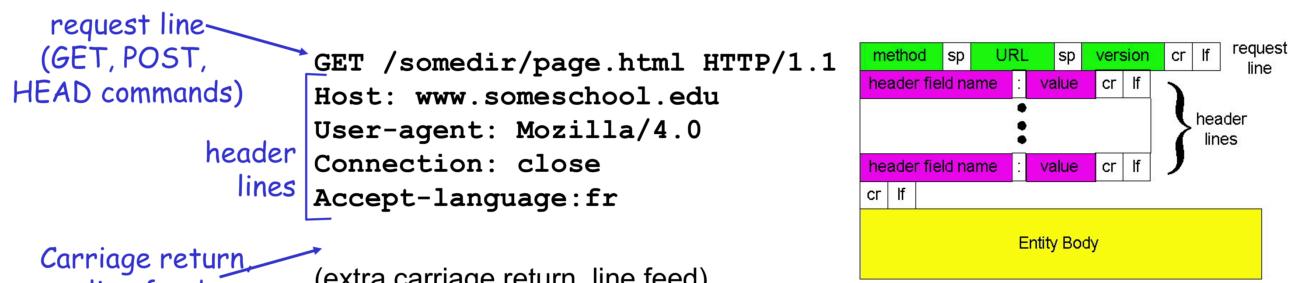


Tempo totale di trasmissione per ogni messaggio =
(2 RTTs) + tempo di trasmissione

² RTT => tempo di attraversamento del pacchetto, da client a server e ritorno

Request message

Messaggi di richiesta inviati dal client al server, formato ASCII



SAFE METHODS	GET	HTTP/1.1 MUST IMPLEMENT THIS METHOD
NO ACTION ON SERVER	HEAD	INSPECT RESOURCE HEADERS
MESSAGE WITH BODY	PUT	DEPOSIT DATA ON SERVER – INVERSE OF GET
SEND DATA TO SERVER	POST	SEND INPUT DATA FOR PROCESSING
	PATCH	PARTIALLY MODIFY A RESOURCE
	TRACE	ECHO BACK RECEIVED MESSAGE
	OPTIONS	SERVER CAPABILITIES
	DELETE	DELETE A RESOURCE – NOT GUARANTEED

Response message



1. Telnet verso un Web server:

telnet www.dis.uniroma1.it 80 Apre connessione TCP verso la porta 80 (default) prso www.dis.uniroma1.it.
Tutto quanto viene digitato è inviato alla porta 80 di www.dis.uniroma1.it

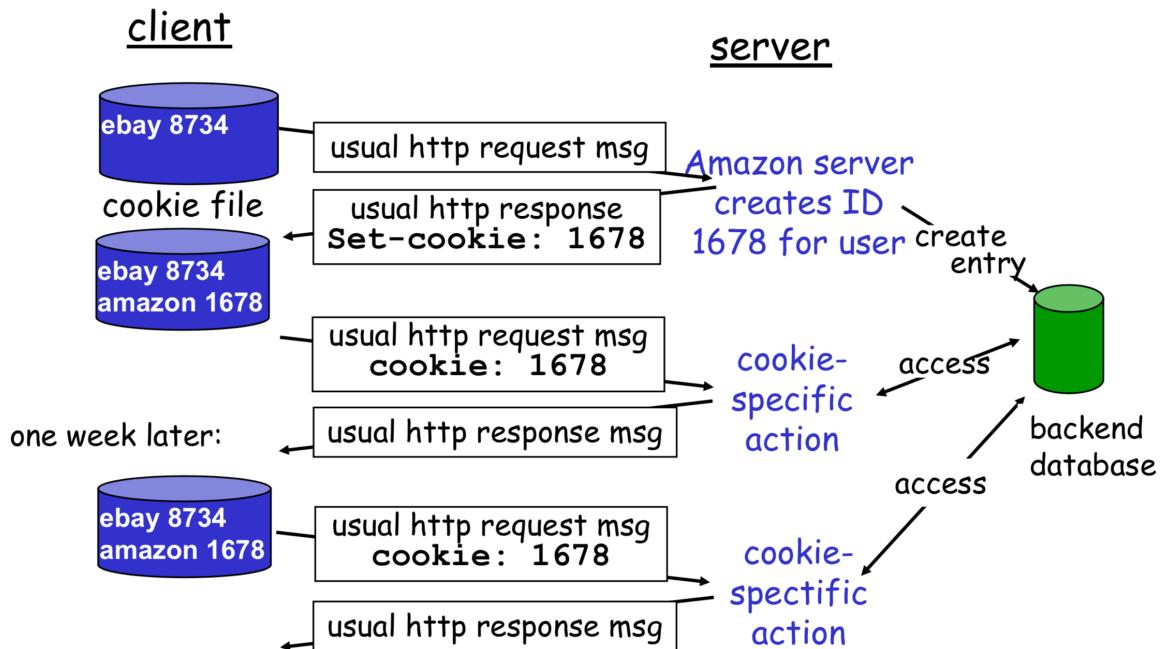
2. Si digita una richiesta http GET:

GET /~leon/index.html HTTP/1.0 Digitando ciò (carriage return due volte), si invia una richiesta GET al server http

Cookies

Permettono agli user agent di mantenere dati (di ogni tipo) riguardanti l'utente.
ex. Nel caso dei permessi di accesso, quando il client si connette al server, questo gli invia una key registrata nel suo database tramite la quale effettuerà i prossimi accessi senza richiedere altre autenticazioni.

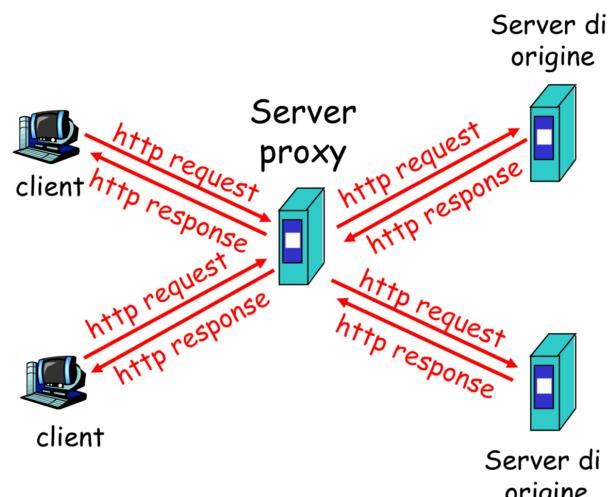
In questo modo però si inviano più dati al sito, limitando la privacy



Web caching (proxy server)

Permettono di rispondere alle richieste del client senza coinvolgere il server d'origine. Il server proxy si comporta infatti come una cache, permettendo di velocizzare le connessioni e diminuendo il traffico verso il server d'origine.

Quando la cache riceve una richiesta dal client restituisce l'oggetto richiesto se presente, altrimenti effettua a sua volta una richiesta verso il server d'origine.



Per caricare oggetti presenti in cache evitando richieste al server d'origine si può usare il get condizionale.

Con *if-modified-since: <date>* il proxy manda semplicemente un codice 304 *not modified* se non sono state apportate modifiche dall'ultima richiesta.

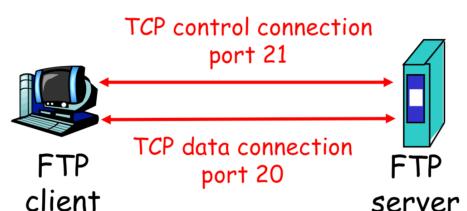
FTP (File Transfer Protocol)

Il client apre 2 connessioni:

- Controllo (porta 21) => permette lo scambio di messaggi di controllo tra server e client
- Dati (porta 20) => trasferisce dati

Il server ftp mantiene informazioni sulla directory corrente e sull'autenticazione.

Si necessita dell'apertura di una nuova connessione per ogni nuovo file da trasferire.



E-mail

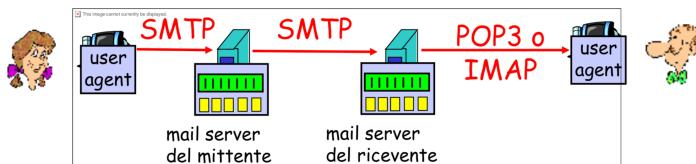
La trasmissione è effettuata tramite i mail server.

Per caricare i messaggi dallo user agent al server mittente e poi spedirli a quello destinazione si usa il protocollo **SMTP** (simple mail transfer protocol), che funziona tramite TCP (porta 25), con formato ASCII 7 bit.

Per file multimediali si usa formato **MIME**.

A destinazione, per accedere al mail server si possono usare 2 protocolli:

- **POP** (post office protocol) => protocollo più semplice, permette solo autenticazione e scaricamento messaggi
- **IMAP** (internet mail access protocol) => permette di gestire caselle di posta remote, mantenere una gerarchia di cartelle e scaricare solo parti dei messaggi



DNS (Domain Name System)

Funzione base di internet implementata come protocollo applicativo (su UDP) che permette la traduzione da hostname a IP corrispondente.

Consiste in un database distribuito gerarchico gestito dagli ISP, quando il DNS server locale non riesce a risolvere l'indirizzo, poiché appartiene ad un'altra rete, contatta uno dei root DNS che gli indica in quale sottorete cercare. Quindi nel caso in cui sia necessario contattare più server si parla di iterative query. Il percorso appena trovato viene salvato nella cache del server dove

rimarrà per qualche giorno, usare un unico server diminuirebbe prestazioni e scalabilità.

I record delle tabelle DNS hanno il formato:

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

Con:

- Tipo A => nome = host, valore = IP
- Tipo NS => nome = dominio, valore = IP name server di riferimento
- Tipo CNAME => nome = alias, valore = nome reale
- Tipo MX => valore è il nome del mailserver associato al nome

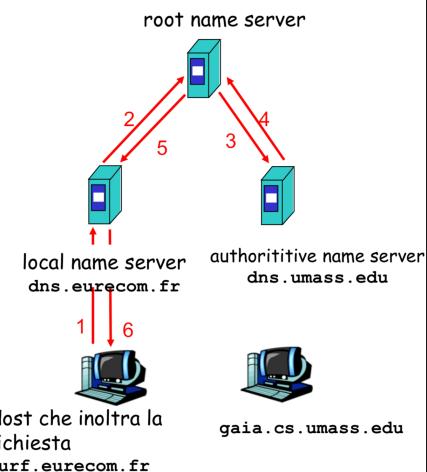
Per far funzionare il protocollo host e servers si scambiano messaggi richiesta (query) e risposta (reply), con il formato in figura.

Risposta o richiesta è specificato tra i flags insieme alle informazioni di iterative query, il record con formato RR viene inserito in answer.

Esempio

L'host `surf.eurecom.fr` vuole l'indirizzo IP di `gaia.cs.umass.edu`

1. Contatta il server DNS locale, `dns.eurecom.fr`
2. `dns.eurecom.fr` contatta il root name server, se necessario
3. Il root name server contatta il name server di riferimento, `dns.umass.edu`, se necessario
4. Il name server di riferimento contatta il local name server, `dns.eurecom.fr`
5. Il local name server contatta il host che inoltra la richiesta, `surf.eurecom.fr`
6. Il host inoltra la richiesta, `surf.eurecom.fr`



identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

↑
12 bytes
↓

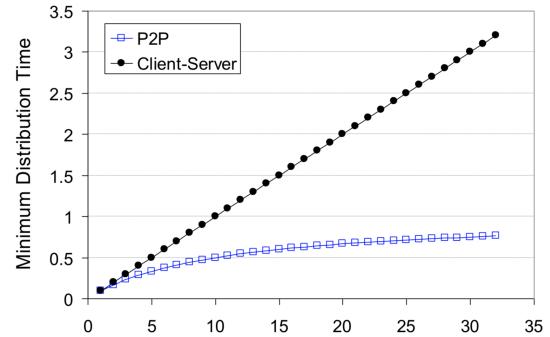
Architetture P2P

Il P2P computing o networking è un architettura distribuita che partitiona il carico di lavoro equivalentemente tra tutti i peers, senza la necessità della coordinazione da parte di un server centrale.

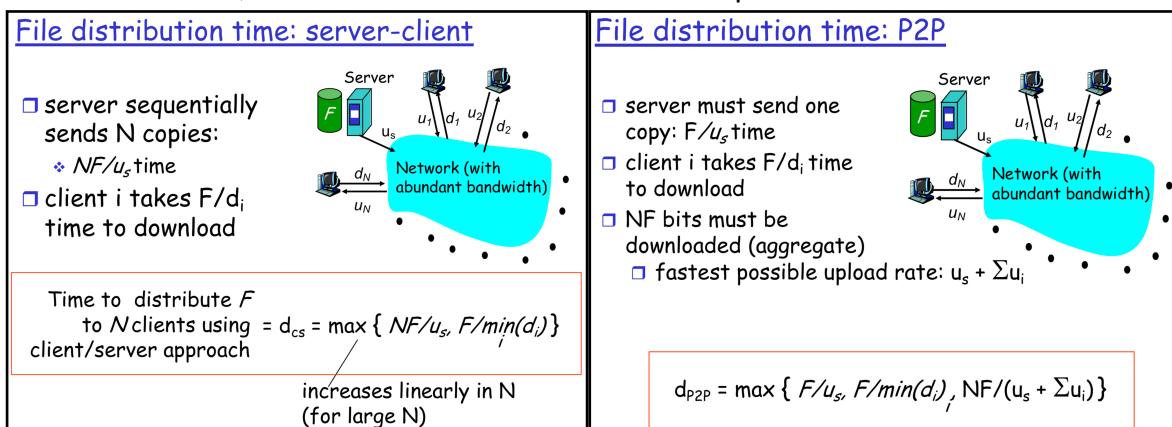
I peers si connettono tramite un **overlay network**, ovvero una rete logica sopra lo strato fisico in TCP.

caratteristiche:

- Scalabilità
- Condivisione/riduzione costi
- Disponibilità del servizio
- Autonomia di ogni peer



Molto utile per la distribuzione di file di grosse dimensioni, infatti invece di avere un solo server che invia il file a tutti, si ha che ogni peer ridistribuisce agli altri qualsiasi porzione di file abbia ricevuto, diminuendo il carico sul server di partenza.



Uno tra i più diffusi protocolli di distribuzione di file P2P è BitTorren, in cui ogni torrent (gruppo di peers che si scambiano un file) ha un nodo di infrastruttura chiamato tracker al quale si registra ogni nuovo utente che si connette alla rete.

Lo scambio di messaggi avviene tramite TCP, i file sono divisi in blocchi da 256kb, come un peer riceve un blocco inizia a ritrasmetterlo agli altri peers.

DHT (tabelle hash distribuite)

Database distribuito contenente tutti i peers connessi alla rete.

Non potendo associare ad ogni peer una lista di tutti i partecipanti si associa ad ognuno solo un piccolo sottoinsieme della totalità delle coppie (chiave, valore).

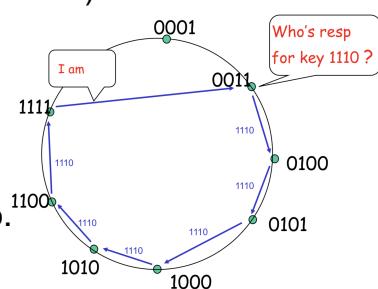
Per decidere a chi assegnare ogni nuova chiave si utilizza una funzione hash, ovvero, una volta associato un intero ad ogni chiave, la si associa al peer con il numero più vicino.

- **DHT circolare** => ogni peer tiene traccia solo del predecessore e del successore, quindi ad ogni richiesta si invia il messaggio circolarmente fino a trovare il peer richiesto.

Nelle grosse reti è possibile inserire scorcatoi.

- **Peer churn** => tiene conto del fatto che ogni peer può uscire dalla rete senza preavviso, si mantengono quindi sia il successore che il predecessore del successore, inoltre vengono eseguiti ping periodici per constatare la presenza del peer.

Quando un nuovo peer si vuole aggiungere copia le informazioni del predecessore e successore, i quali a si aggiornano a loro volta

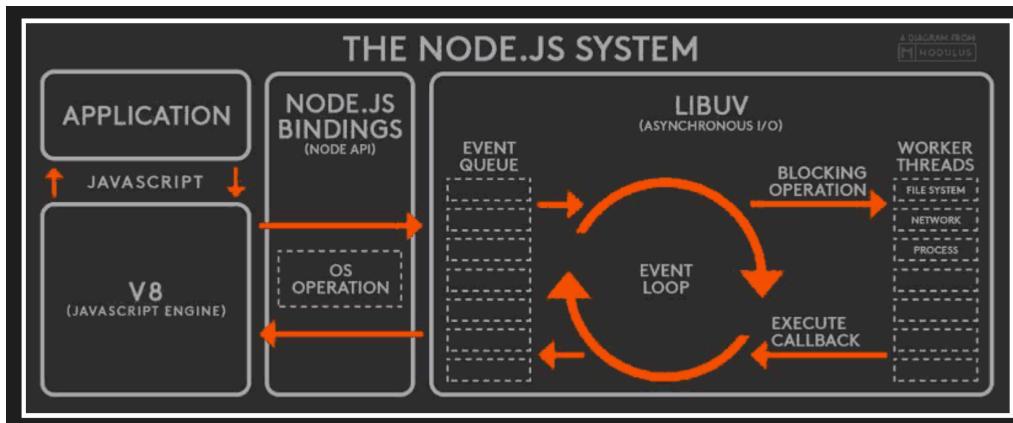


Application server

Mentre i web server possono gestire solo risorse statiche (pagine web), gli application server sono dei software framework che permettono di generare pagine dinamiche in accordo con uno specifico business logic.

Questo poiché, prima di generare la pagina, l'AS accede al sistema informativo e poi rappresenta le informazioni secondo il formato designato.

Per questo corso di fa riferimento al framework **node.js** per javascript (asynchronous)



Caratteristiche:

- Single thread
- Non si mette in attesa, usa gli eventi (compresi eventi IO)
- JS è naturalmente scritto per lavorare in modo asincrono
- Per bloccare le operazioni utilizza le thread pool che aspettano la fine delle operazioni

Tutto questo è realizzato tramite le **callback functions**

When the readFile has finished
It calls the callback function

```
fs.readFile('/foo.txt', function(err, data){  
    // TODO: Error Handling Still Needed!  
    console.log(data);  
});
```

The cost of I/O

L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41 000 000 cycles
Network	240 000 000 cycles

Per bloccare un operazione è possibile utilizzare le funzioni timeout

```
setTimeout(function(){  
    console.log('(A) prima accedo');  
}, 2000);  
  
setTimeout(function(){  
    console.log('(B) poi leggo');  
}, 500);
```

A AFTER B

Altro concetto importante in node sono le **Promise** che rappresentano un proxy per un valore non necessariamente noto quando la promise è stata creata.

Consentono di associare degli handlers con il successo o il fallimento di un'azione asincrona (e il "valore" in caso di successo, o la motivazione in caso di fallimento). Questo in pratica consente di utilizzare dei metodi asincroni di fatto come se fossero sincroni: la funzione che compie del lavoro asincrono non ritorna il valore di completamento ma ritorna una promise, tramite la quale si potrà ottenere il valore di completamento una volta

che la promise sarà terminata.

```
new Promise(function(resolve, reject) { ... });
```

REST (REpresentational State Transfer)

Un RESTful web service è uno stile architetturale di sistema che fornisce interoperabilità tra diversi sistemi in internet.

Il sistema richiedente accede e manipola la rappresentazione delle risorse web tramite un set uniforme e predefinito di operazioni stateless, usando standard tipo HTTP.

Funzionamento:

1. Il client richieda una risorsa specifica sul server
2. Il server risponde con una rappresentazione richiesta della risorsa
(il server è stateless, non mantiene informazioni sul client)

Principi REST:

- Client/server => ognuno con funzioni indipendenti
- Stateless => scalabilità
- Cache => riduce latenza
- Layered system => componenti nascosti ed incapsulati
- Code on demand => estendibilità e configurazione del sistema
- Uniform interface => tutti i componenti seguono le stesse regole di comunicazione

Il funzionamento di una RESTful application si basa sull'utilizzo dell **API** (application programming interface) ovvero endpoint che permettono a parti di software di comunicare con altre. Generalmente la comunicazione avviene tramite codice **JSON** (javascript object notation) oppure **XML**.

La gestione delle informazioni avviene mappando i metodi HTTP sullo standard **CRUD**

CRUD	REST	
CREATE	POST 	Create a sub resource
READ	GET 	Retrieve the current state of the resource
UPDATE	PUT 	Initialize or update the state of a resource at the given URI
DELETE	DELETE 	Clear a resource, after the URI is no longer valid

In contrapposizione allo stile REST c'è il protocollo standard SOAP (Simple Object Access Protocol), più robusto ma aumenta la complessità del codice

PaaS

"Platform as a Service", è una categoria di cloud computing che fornisce agli sviluppatori una piattaforma e un ambiente per costruire applicazioni e servizi su Internet. I servizi PaaS vengono ospitati su cloud e gli utenti vi accedono tramite il proprio browser web. Consente agli utenti di creare applicazioni software utilizzando gli strumenti forniti dal provider. I servizi PaaS possono offrire funzioni preconfigurate da scartare o includere in base alle proprie necessità. Di conseguenza, i pacchetti possono variare tra l'offerta di semplici strutture "point and click" che non richiedono al cliente alcuna competenza in fatto di hosting fino alla fornitura di opzioni di infrastrutture per attività avanzate di sviluppo.

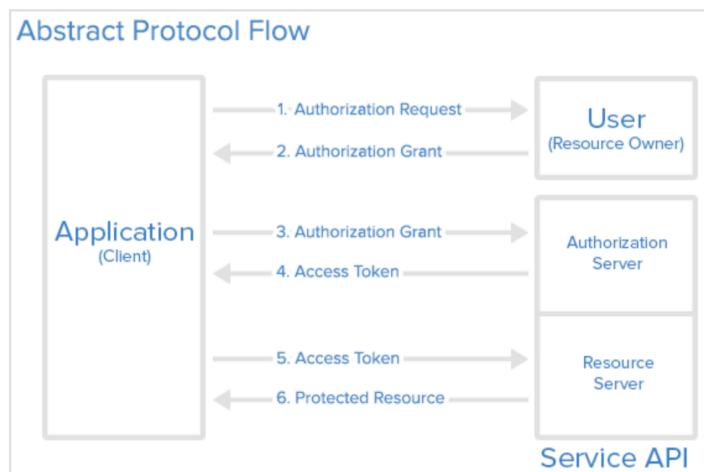
Oauth

È uno standard aperto per delegare l'accesso, usato comunemente dagli utenti per accedere ad i servizi delle applicazioni senza fornire una password.

L'implementazione avviene tramite Oauth2, un framework di autenticazione che permette all'applicazione di avere un accesso limitato alle informazioni personali dell'utente su servizi come google, facebook,

Funzionamento:

1. L'applicazione richiede il consenso per l'accesso
2. Se l'user accetta, invia un authorization grant
3. l'applicazione richiede un access token dal server di autorizzazione presentando la sua identità ed il grant ricevuto dal user
4. Se i dati sono validi il server concede l'autorizzazione
5. l'applicazione richiede le risorse al resource server presentando il token per l'accesso
6. Se il token è valido, il server invia le risorse



Prima di poter utilizzare questo procedimento si necessita di registrare l'applicazione al servizio che fornisce l'autenticazione.

Fornendo:

- App name
- App website
- Redirect URI e Callback URI

Il servizio userà come credenziali il client ID (equivalente a username) ed il client Secret (equivalente alla password)

Esistono 4 tipi di **authorization grant**:

- *authorization code* => più diffuso, usato per server-side app
- *Implicit* => applicazioni mobili o web application (che girano sul device dell'user)
- *Resource owner password credentials* => applicazioni di fiducia (dello stesso produttore)
- *Client credentials* => application API access

Node.js utilizza il middleware passport per oath

Web Sockets

Il WebSocket protocol è un miglioramento di HTTP, abilita le pagine web ad usare un canale di comunicazione full-duplex, ovvero un canale di comunicazione in grado di trasmettere le informazioni in entrambe le direzioni tramite un unico socket.

Inoltre, invece di utilizzare il metodo di polling e richiedere un refresh continuo, server e client inviano nuovi dati solo quando sono presenti aggiornamenti.

Questo favorisce le applicazioni real-time e riduce la latenza ed il traffico non necessario sul web che si avrebbe mantenendo 2 connessioni aperte ed usando il polling.

Inoltre questo protocollo permette di oltrepassare firewalls e proxies, poiché, una volta rilevato il proxie, chiede automaticamente di istaurare una connessione diretta.

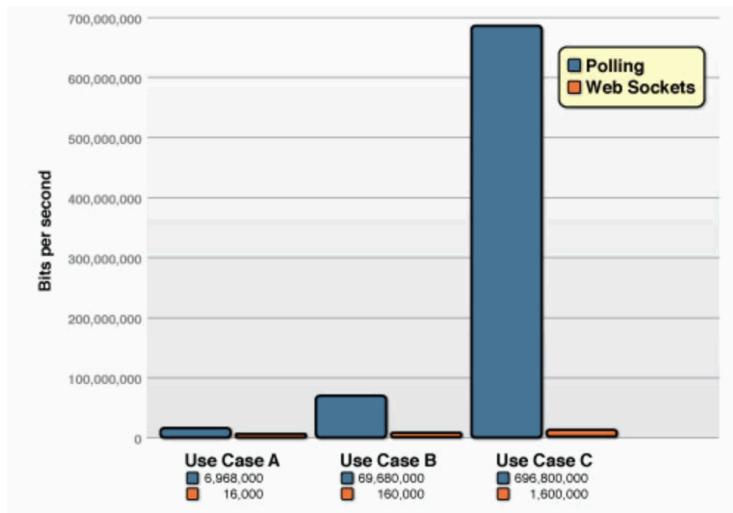


Figure 3 — Comparison of the unnecessary network throughput overhead between the polling and the WebSocket applications

Come si vede in figura, il metodo di polling di HTTP, richiedendo sempre request e response richiede molto più tempo, con web socket invece ogni messaggio viene inviato come è pronto

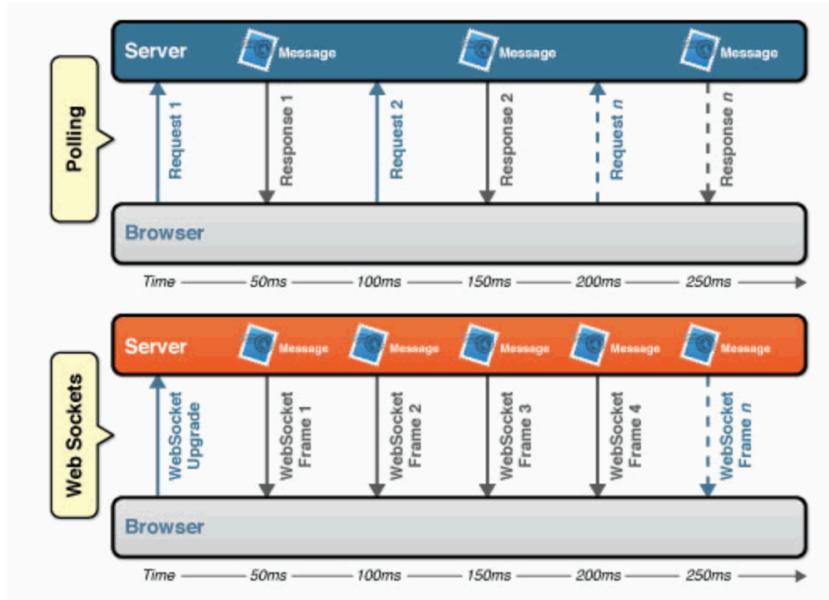


Figure 4 — Latency comparison between the polling and WebSocket applications

Il tutto rimane comunque sincrono

AMQP (Advanced Message Queuing Protocol)

HTTP è intrinsecamente sincrono, il client effettua una richiesta ed attende per una risposta, grazie ad AMQP è possibile instaurare una comunicazione asincrona.

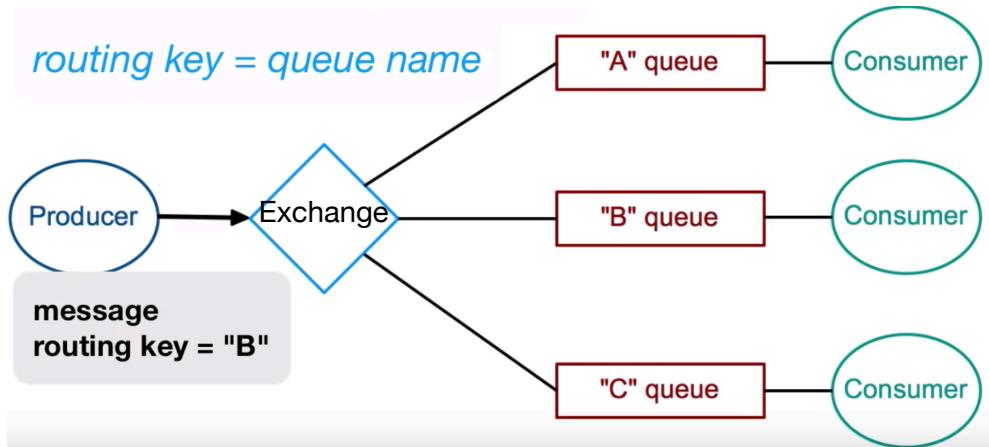
AMQP è di fatto uno protocollo dell'app layer per lo scambio di messaggi, generalmente l'implementazione avviene tramite RabbitMQ.

Componenti:

- **Producer** => host che invia il messaggio
 - **Exchange** => si occupa di inoltrare il messaggio a seconda del tipo
 - **Queue** => coda per immagazzinare i messaggi
 - **Consumer** => host che riceve il messaggio
- OSS possono essere presenti più producer e più consumer

Tipi di exchange:

- fanout => indica all'exchange che il messaggio deve essere inviato a tutte le code
- Direct => invia il messaggio solo alle code la cui binding key corrisponde con la routing key
- Topic => tipo direct ma con un match parziale
- Header => utilizza l'header del messaggio al posto della routing key
- Default (nameless) => associa la routing key alla queue name invece che alla binding key della coda(come ex)



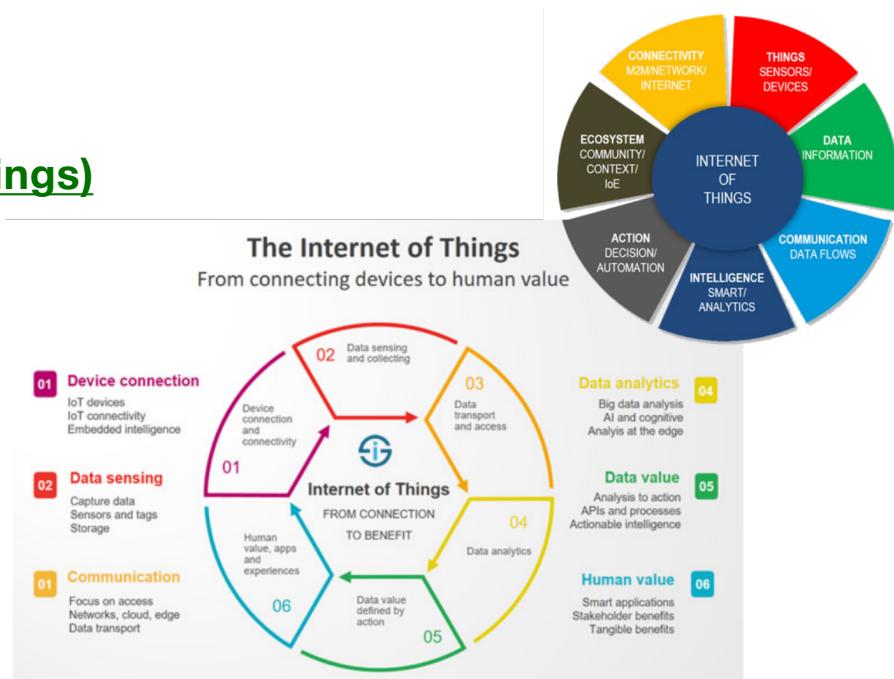
<https://www.rabbitmq.com/getstarted.html>

Microservices

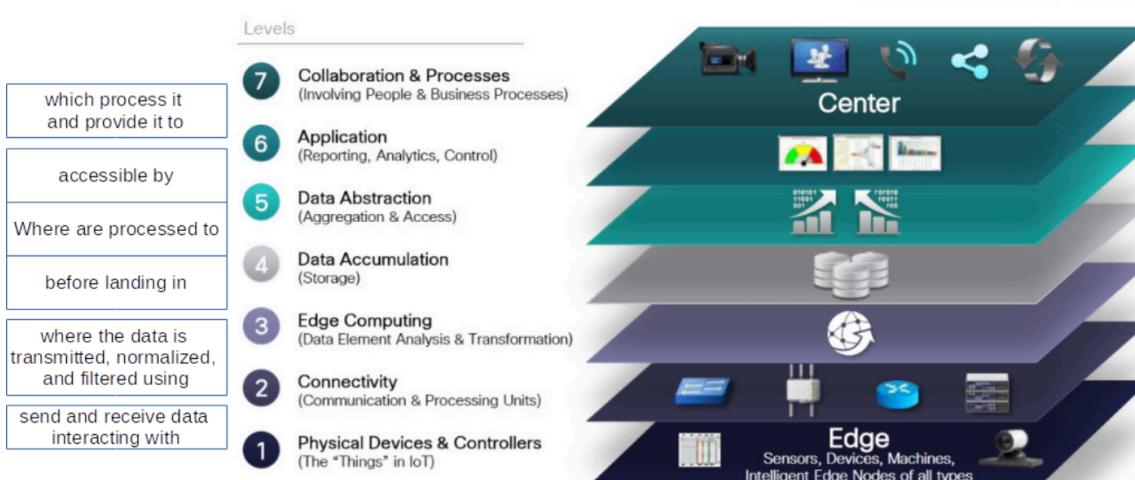
È un metodo di developing delle applicazioni software come un insieme di moduli indipendenti in cui ogni servizio agisce come unico processo e comunica con gli altri per creare un servizio più complesso

IoT (internet of things)

La miniaturizzazione, la riduzione del costo dell'hardware e la possibilità di interconnettere miliardi di dispositivi ha permesso di creare l'internet of things, ovvero una rete di dispositivi ed embedded systems in grado di scambiarsi dati per migliorare la vita delle persone.



IoT World Forum Reference Model



I protocolli più utilizzati sono:

- MQTT => gestisce domande/risposte in modo asincrono ed a basso consumo senza preoccuparsi del contenuto dei messaggi, sviluppo più semplice e ed utilizzo più leggero rispetto ad HTTP (ottimo per embedded)
- COAP => a differenza di MQTT si basa su UDP, anche in questo caso lo sviluppo e l'utilizzo sono più semplici rispetto ad HTTP tramite header binari (QoS).
Può quasi essere considerato come un alternativa ad HTTP ed essere utilizzato per servizi REST, inoltre tutti i nodi hanno anche ruolo di server.

Block chain

Basate sulla **Ledger**, ovvero una lista comprendente tutti gli utenti ed il valore posseduto da ognuno di essi.

Ogni utente ha una copia e ad ogni transazione invia l'aggiornamento della del proprio ledger in broadcast a tutti gli utenti appartenenti alla rete (ex. Bitcoin).

Per mantenere l'autenticità si utilizza una firma digitale che cambia ad ogni transazione.

Ogni transazione è numerata, la disponibilità sul conto di una persona è data dalla somma di tutte le sue transazioni precedenti, questo permette di evitare la duplicazione delle transazioni (ovvero usare 2 volte la stessa moneta).



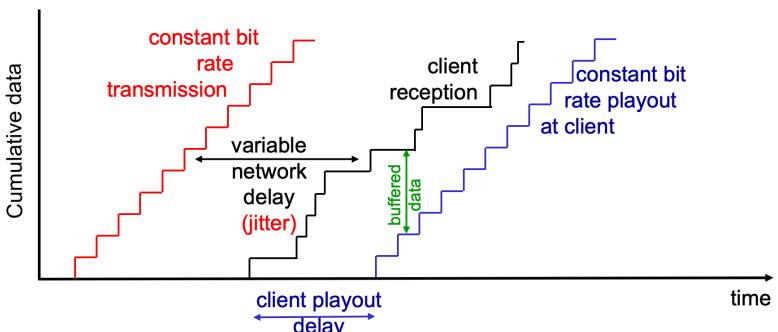
Multimedia networking

Streaming stored video

Permette di riprodurre contenuti video immagazzinati su un server.

Il ritardo del network può essere variabile (**jitter**) e l'utente è interattivo (può mettere in pausa ed andare vanti/indietro), quindi per il corretto funzionamento si necessita di un **client-side buffer** e di un **playout delay**,

ovvero, anche se il video può essere riprodotto mentre è ancora in download, il client inizia la riproduzione solo dopo che buona parte di esso è già stata scaricata.



La trasmissione può essere realizzata con:

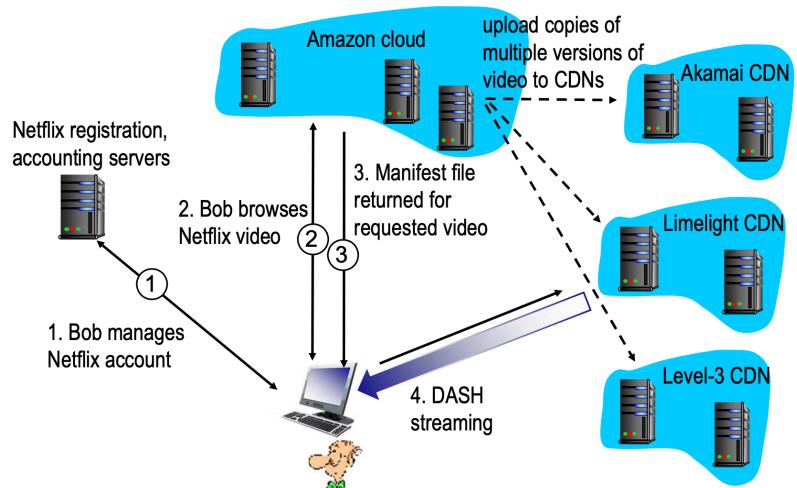
- UDP => il server invia al rate appropriato per il client in modo costante con un breve playout delay (2-5 sec) per rimuovere il jitter.
UDP potrebbe però non passare il firewall
- HTTP => file ricevuti tramite una /get HTTP alla velocità massima concessa dal TCP, può incontrare problemi di congestione ed ha un payout lungo, però non ha problemi con i firewall
- **DASH** (Dynamic, Adaptive Streaming over HTTP) =>
 - Lato server => divide il video in segmenti ognuno di essi immagazzinato e codificato ad un rate differente e con un URL differente
 - Lato client => misura periodicamente il bandwidth e richiede il segmento con il rate più adatto in ogni momento (si ha quindi client intelligente)

CDN (Content Delivery Network)

Una rete distribuita per la condivisione dei contenuti, invece di usare un solo server di origine, le copie dei video sono disseminate in moltissime ubicazioni fisiche, le quali possono rispondere direttamente alle richieste degli utenti finali al posto dell'origine essendo più vicini a questi ultimi.

CDN riduce il traffico per il server principale, migliorando la web experience con conseguenti vantaggi sia per il provider di contenuti che per gli utenti finali. In sua assenza, i server di origine dovrebbero rispondere a ogni singola richiesta degli utenti finali, il che si traduce in un traffico notevole verso l'origine ed aumenta le probabilità di un guasto del server in caso di picchi di traffico estremamente elevati o di un carico persistente.

Protocollo utilizzato anche da Netflix tramite servers Amazon collegati a loto volta con altri server CDN.



VoIP (Voice-Over-IP)

Servizio per la comunicazione in tempo reale tramite internet, è limitato dal best-effort del protocollo IP, che fa del suo meglio per recapitare i datagrammi nel minor tempo possibile, senza però fornire assicurazioni su ritardo o sulla perdita dei pacchetti.

Problemi:

- Perdita pacchetti => dovuta all'utilizzo del protocollo UDP, non è possibile usare TCP a causa della lentezza nella ritrasmissione
- Ritardo end-to-end => dovuto all'accodamento nei router, ritardi inferiori a 150 ms non sono percepibili all'orecchio umano, se superiori a 400 ms vengono scartati
- Jitter => come per lo streaming video, risolvibile tramite playout delay e numerando i pacchetti per evitare che si confonda l'ordine

Il **playout delay** può essere di 2 tipi:

- Fixed => ogni blocco viene riprodotto esattamente q ms dopo esser stato generato, quindi se arriva in tempo viene riprodotto al tempo $t+q$ (t tempo di generazione), altrimenti viene scartato.
- Adaptive => il ritardo viene stimato all'inizio di ogni periodo di attività vocale, facendo in modo che le pause siano compresse e prolungate.

Segnando con:

- t => istante in cui è generato il pacchetto
 - r => istante in cui viene ricevuto il pacchetto
 - p => istante in cui è riprodotto il pacchetto
 - Alpha, beta e K => costanti
- si ha che il ritardo è dato da

$$d_i = (1-\alpha)d_{i-1} + \alpha(r_i - t_i)$$

la deviazione media del ritardo invece è data da

$$v_i = (1-\beta)v_{i-1} + \beta|r_i - t_i - d_i|$$

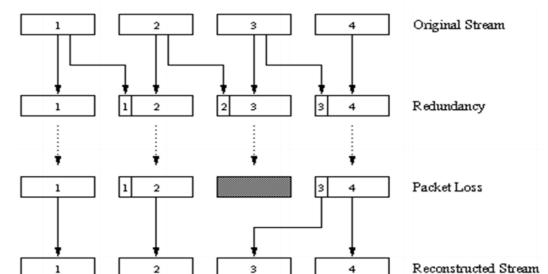
quindi l'istante di riproduzione del pacchetto è dato da

$$\text{playout-time}_i = t_i + d_i + Kv_i$$

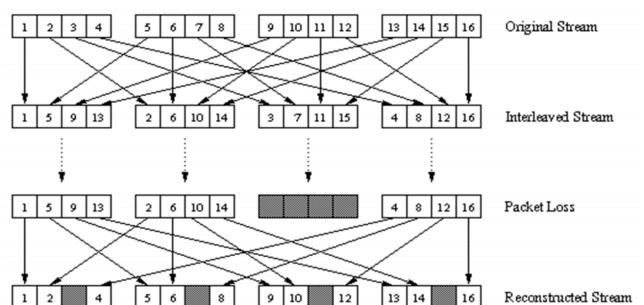
Il recupero dei pacchetti persi può avvenire tramite:

- FEC (forward error correction) => aggiunge informazioni ridondanti ai pacchetti:

- V1 => ogni n blocchi invia un blocco ridondante ottenuto tramite un OR esclusivo sugli n blocchi originali (ricostruisce fino ad 1 pacchetto perso)
- V2 => ad ogni pacchetto viene aggiunta come ridondanza una versione a bassa qualità del pacchetto prima, in questo modo se il pacchetto viene perso si sostituisce con la ridondanza del successivo



- Interleaving (terfogliazione) => ogni blocco viene suddiviso in blocchi più piccoli sparsi nei vari pacchetti trasmessi, così ogni pacchetto contiene solo piccole unità di ogni blocco. In questo modo se un pacchetto viene perso non si nota troppo la differenza nella conversazione



skype

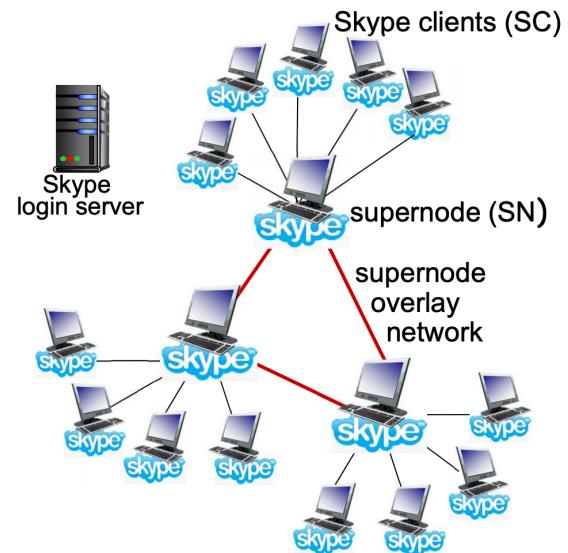
I pacchetti di controllo vengono scambiati tramite TCP, quelli audio e video sono generalmente inviati tramite UDP, ma se bloccati dal firewall anche questi usano TCP. Per il recupero dei pacchetti usa FEC.

È un servizio P2P formato da:

- Clients (SC) => peers connessi direttamente per effettuare chiamate VoIP
- Super nodes (SN) => permettono di bypassare il NAT ed effettuare connessioni tra host interni a reti private
- Overlay network => rete di connessione tra SN
- Login server => permette di loggare Skype

Operazioni per instaurare la connessione:

1. Il client joint skype contattando il suo SN tramite TCP
2. Log-in al server centrale
3. Il client ottiene l'IP per la chiamata
4. Inizia la chiamata diretta tra i peer



Protocolli per conversazioni real-time

RTP (Real-Time Protocol)

Pacchetti incapsulati nei segmenti UDP, li estendono definendo una struttura per i pacchetti che si occupano del trasporto audio e video.

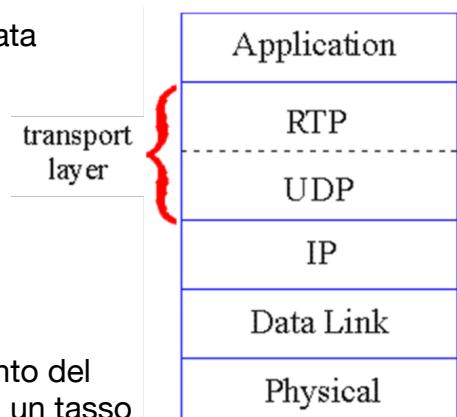
Poiché l'incapsulamento è visibile solo agli end systems e non ai router di passaggio, non provvede nessun meccanismo per garantire il tempo di consegna dei dati o altri QoS.

Un header RTP è formato da:

- Payload type (7 bits) => indica il tipo di codifica utilizzata

Payload type 0: PCM mu-law, 64 kbps
Payload type 3: GSM, 13 kbps
Payload type 7: LPC, 2.4 kbps
Payload type 26: Motion JPEG
Payload type 31: H.261
Payload type 33: MPEG2 video

- #sequenza (16 bits) => incrementa ad ogni invio
- Timestamp (32 bits) => riporta istante di campionamento del primo byte nel pacchetto dati RTP, viene aumentato di un tasso costante anche se la sorgente è inattiva
- SSRC (32 bits) => identifica la sorgente



payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields
--------------	-----------------	------------	---------------------------	----------------------

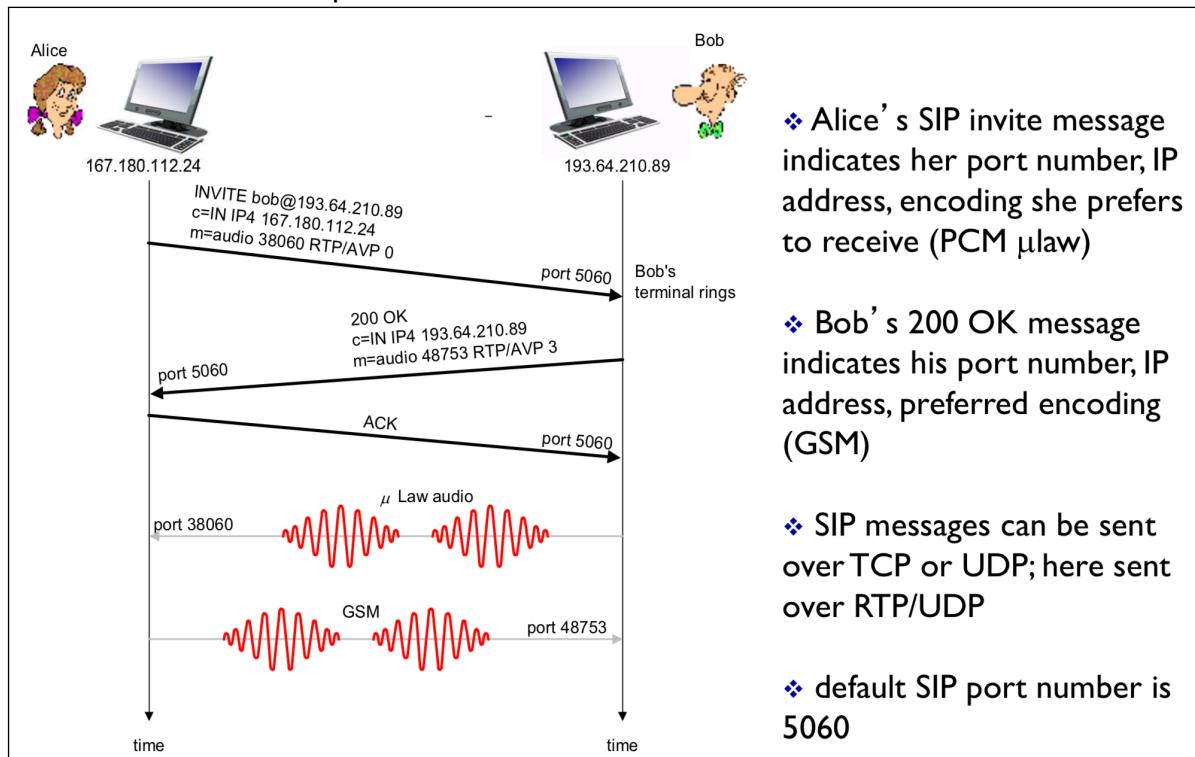
RTCP (Real-Time Control Protocol)

Aggiunta agli RTP inviati periodicamente da ogni partecipante per il controllo

SIP (Session Initiation Protocol)

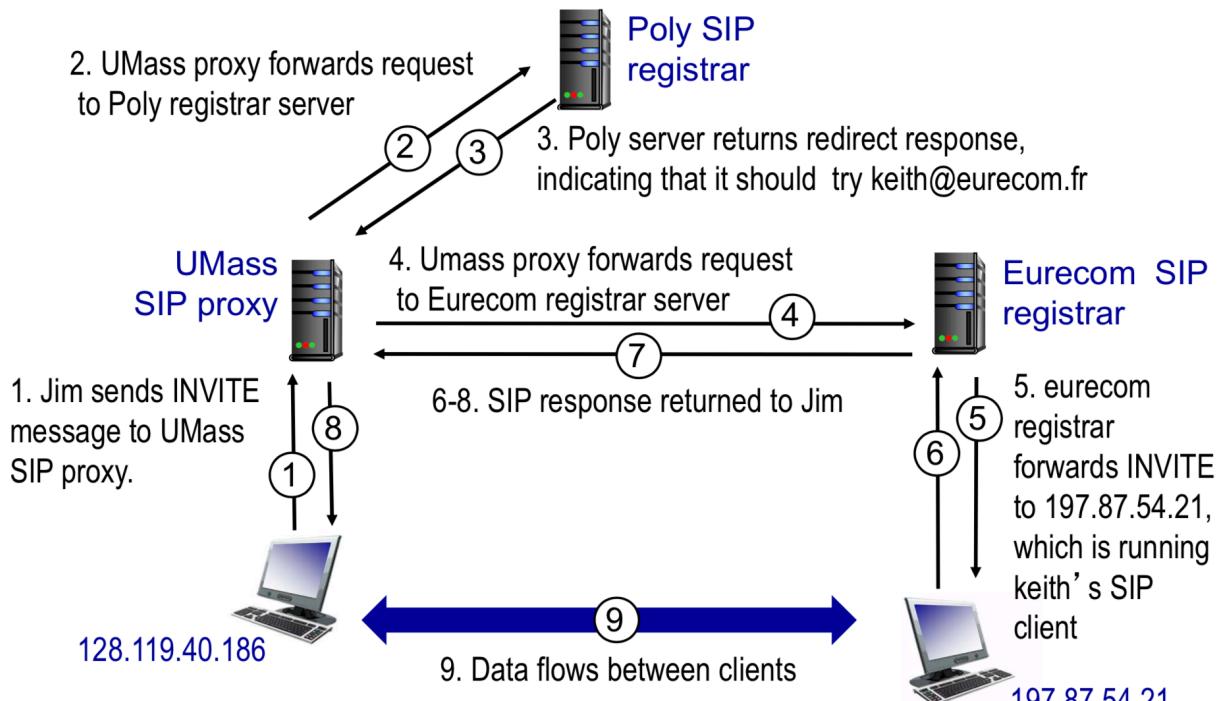
Protocollo che permette al chiamante di individuare l'indirizzo IP del chiamato e notificargli che vuole iniziare una chiamata, inoltre permette di stabilire le codifiche per i contenuti multimediali.

Per trovare l'IP fa uso del protocollo DHCP.



Tramite proxy:

SIP example: `jim@umass.edu` calls `keith@poly.edu`



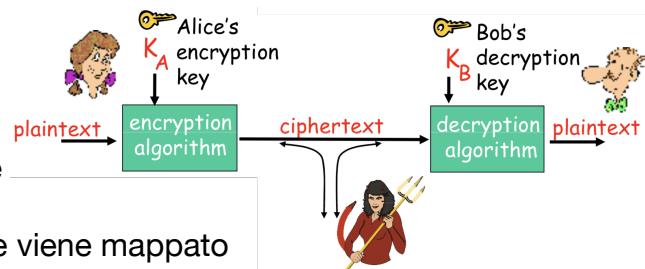
Network security

Principi di sicurezza:

- **Disponibilità** => rendere disponibili a ciascun utente abilitato le informazioni richieste
- **Integrità** => impedire alterazioni dei dati da parte di utenti non autorizzati o malfunzionamenti
- **Riservatezza** => solo mittente e destinatario devono poter interpretare i messaggi, criptandoli e decriptandoli
- **Autenticazione** => ciascun utente deve poter verificare l'autenticità delle informazioni

Le minacce per la sicurezza possono essere:

- Eavesdrop => intercettazione messaggi
- Inserimento attivo di altri messaggi nella connessione
- Impersonation => inserire indirizzi sorgente falsi nei pacchetti
- Hijacking => dirottare la connessione verso un altro indirizzo (per aumentare le visualizzazioni su quella pagina)



Crittografia

Semplici esempi di algoritmi sono:

- **Cifrario di cesare** => ogni carattere viene shiftato di una costante k
- **Monoalphabetic cipher** => ogni carattere viene mappato sempre sullo stesso, con un alfabeto di 26 lettere si hanno $26!$ possibilità di accoppiamento, che si riduce trovando le ricorrenze
- **Polialphabetic cipher** => combina i 2 cifrari di sopra associando ad ogni posizione una cifratura diversa

Esistono diversi modi per bypassare questi sistemi:

- Analisi statistica
- Conoscendo alcuni possibili accoppiamenti
- Conoscendo la forma codificata e decodificata di un singolo messaggio

Gli algoritmi di criptaggio sono generalmente noti, le chiavi possono essere:

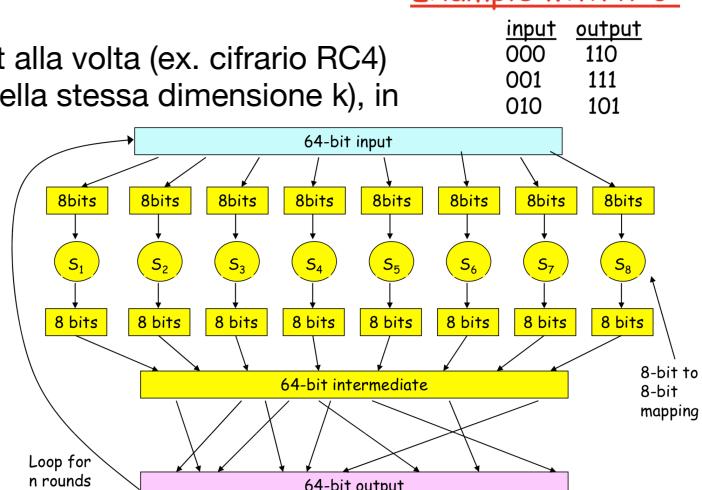
Chiave simmetrica

Il criptaggio può essere seguito su un bit alla volta (ex. cifrario RC4) oppure su blocchi del messaggio (tutti della stessa dimensione k), in cui ognuno viene cifrato

indipendentemente usando una tabella. In generale si hanno quindi $2^k!$ diverse permutazioni, questo metodo è molto efficace ma poco efficiente, si utilizzano quindi funzioni che simulano in modo casuale piccole tabelle permutate che vengono poi rimescolate per produrre l'uscita del numero di bit desiderata. Il procedimento viene quindi iterato per aumentare le permutazioni.

Questo metodo, leggermente migliorato e con l'aggiunta di una chiave simmetrica da 56bit è usato dal protocollo **DES (Data Encryption Standard)**.

Example with k=3:



Nel caso di messaggi molto lunghi si avrebbero blocchi uguali facilmente comprensibili, si utilizza quindi i **CBC (cipher block changing)** che funziona così:

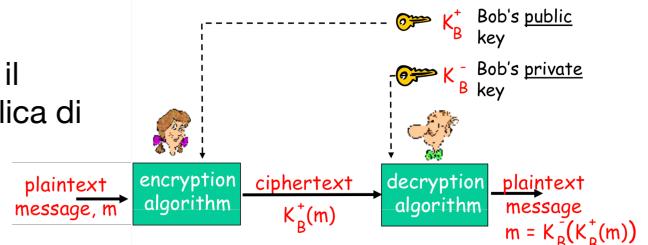
1. Il mittente genera una stringa di k bit (chiamata $c(0)$) e la manda al destinatario
2. Poi calcola $[m(1) \text{ OR esclusivo } c(0)]$ e lo usa come input per l'algoritmo di cifratura a blocchi ottenendo $c(1)$
3. Ripete ricorsivamente

così anche se lo stesso messaggio è mandato più volte la versione criptata è sempre differente.

Chiave asimmetrica

Mittente e ricevente non si scambiano chiavi, il secondo infatti possiede sia una chiave pubblica di codifica che una chiave privata di decodifica.

La chiave totale per il decriptaggio è data da una funzione che elabora le 2 chiavi insieme.



L'algoritmo che permette questi scambi è RSA, facendo uso dell'operazione $\text{mod } n$.

Per generare le chiavi il ricevente deve:

1. Scegliere 2 numeri primi p e q , più alto è il valore più è efficace ma la cifratura più lenta
2. Calcola $n = pq$ e $z = (p-1)(q-1)$
3. Sceglie un numero $e < n$, diverso da 1 e primo rispetto a z ($e \Rightarrow$ encryption)
4. Trovare d (decryption) tale che $ed-1$ sia divisibile per z (ovvero $[ed \text{ mod } z = 1]$)
5. La sequenza (n, e) è la chiave pubblica, (n, d) la chiave privata

RSA: Encryption, decryption

0. Given (n, e) and (n, d) as computed above
 1. To encrypt message $m (< n)$, compute
 $c = m^e \text{ mod } n$
 2. To decrypt received bit pattern, c , compute
 $m = c^d \text{ mod } n$
- Magic happens! $m = (\underbrace{m^e \text{ mod } n}_c)^d \text{ mod } n$

L'efficacia dell'algoritmo sta nel fatto che, per decriptare il messaggio bisognerebbe fattorizzare n senza conoscere p e q ma non si conoscono algoritmi veloci per la fattorizzazione di numeri interi.

Siccome DES è 10 volte più veloce di RSA, si ha che RSA viene usato per scambiare solo la chiave simmetrica e poi continuare lo scambio tramite DES.

RSA example:

Bob chooses $p=5, q=7$. Then $n=35, z=24$.
 $e=5$ (so e, z relatively prime).
 $d=29$ (so $ed-1$ exactly divisible by z).

Encrypting 8-bit messages.

encrypt:	bit pattern	m	m^e	$c = m^e \text{ mod } n$
	00001000	12	24832	17
decrypt:	c	c^d	$m = c^d \text{ mod } n$	12

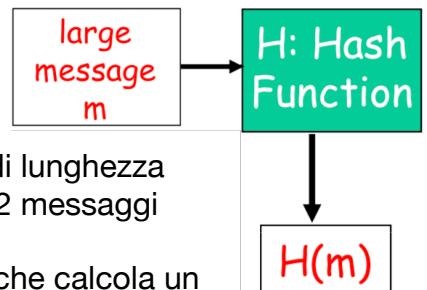
Integrità messaggi

Abilita le 2 entità comunicanti a verificare che i messaggi ricevuti siano autentici.

Per far questo si usano le funzioni Hash crittografiche che, prendono un messaggio m in input e calcolano una stringa di lunghezza fissata con le proprietà che deve essere impossibile trovare 2 messaggi diversi x e y tali che $H(x) = H(y)$.

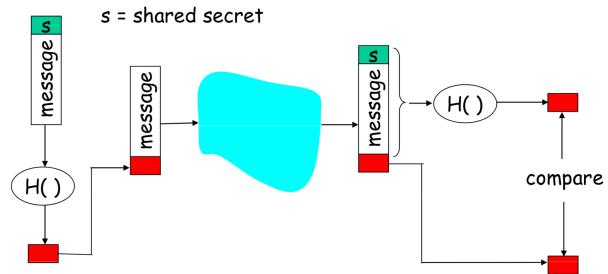
Il primo algoritmo utilizzato per l'hash dei messaggi è MD5, che calcola un hash di 128 bit in 4 fasi.

Una versione più sicura è il SHA-1 con 160 bit, utilizzato dalle applicazioni federali degli USA.



Usando solo la funzione hash chiunque potrebbe falsificare i messaggi, per questo si utilizza il **MAC (Message Authentication Code)** che aggiunge al messaggio una chiave di autenticazione condivisa tra gli utenti prima di calcolare l'hash.

Tramite questo meccanismo, se non si necessita riservatezza, si possono autenticare messaggi senza l'utilizzo di complessi algoritmi di cifratura.

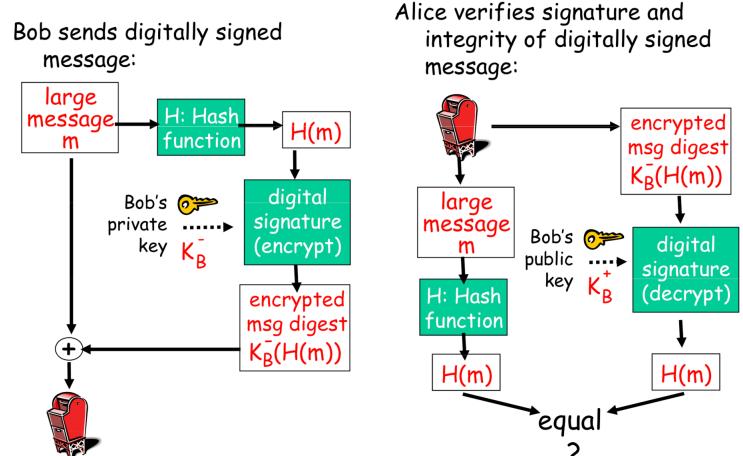
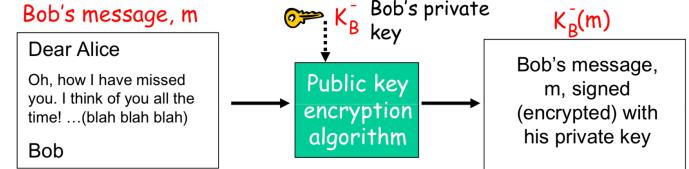


Firme digitali

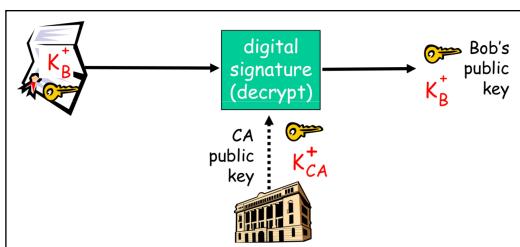
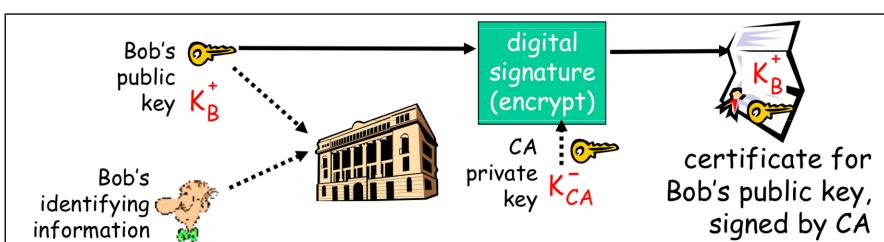
Il primo approccio è simile a quello delle chiavi asimmetriche, il firmatario infatti, al momento della firma, utilizza una chiave privata per la codifica.

La decodifica avviene unendo la chiave privata a quella pubblica, infatti l'unico modo per riottenere il messaggio è tramite la giusta chiave privata

Criptare un messaggio molto lungo può rappresentare un problema dal punto di vista computazionale. Un evoluzione del sistema consiste quindi nel mandare il messaggio non cifrato e cifrare solo l'hash del messaggio. Il ricevente quindi comparerà il messaggio con la cifratura e la chiave dell'hash del messaggio.



La relazione tra una chiave pubblica ed una determinata entità è stabilita da una **CA (Certification Authority)**, che verifica l'identità di una determinata persona e quindi rilascia un certificato che autentica la corrispondenza



Email sicure

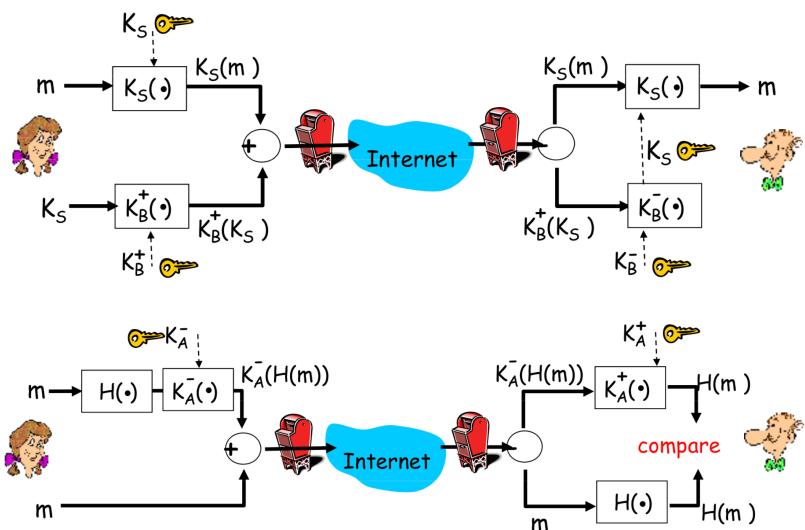
Mittente:

1. Genera randomicamente una chiave privata simmetrica k_s
2. Cifra il messaggio m con la chiave k_s
3. Cifra k_s con la chiave pubblica del ricevente
4. Li concatena ed invia

Ricevente :

1. Separa le informazioni
2. Usa la sua chiave privata k_b per ottenere la chiave k_s
3. Usa la k_s trovata per decodificare $k_s(m)$

Nel caso in cui i 2 siano più interessati all'autenticazione del mittente e l'integrità del messaggio si utilizza il metodo delle funzioni hash



SSL (Secure Sockets Layer)

Applicabile tramite API alle connessioni TCP per aumentarne la sicurezza e supportato dai browser più comuni (https).

Una versione semplificata si riassume in 3 fasi:

1. **Handshake** => il mittente stabilisce la connessione TCP con il destinatario, quindi invia un hello SSL per ricevere il certificato che lo riconosce.

A questo punto genera un MS (simile ad una chiave simmetrica) che verrà utilizzato solo per questa sessione SSL e dal quale deriveranno le altre chiavi, quindi lo cifra ed invia al destinatario

2. **Key derivation** => dal MS vengono quindi generate 4 nuove chiavi (K_c , M_c , K_s , M_s)

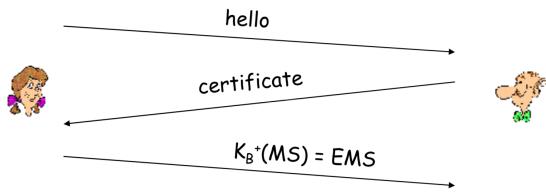
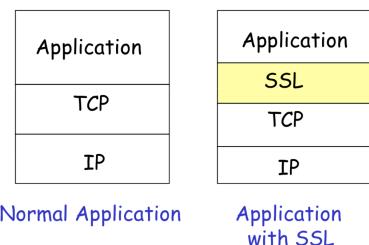
- K_c = encryption key for data sent from client to server
- M_c = MAC key for data sent from client to server
- K_s = encryption key for data sent from server to client
- M_s = MAC key for data sent from server to client

□ MS = master secret

□ EMS = encrypted master secret

3. **Data transfer** => dovendo inserire il controllo

MAC non è possibile cifrare i dati in stream, inoltre non è possibile inserirlo solo alla fine del messaggio, quindi si divide tutto in records, ognuno con il suo MAC e con un numero di sequenza per evitare manomissioni.



length	type	data	MAC
--------	------	------	-----

Nella versione reale non si usa un solo algoritmo, MAC e chiavi specifiche ma, durante l'handshake, ci si accorda su vari tipologie che verranno utilizzate durante la trasmissione. Questo viene fatto poiché, intercettando tutti gli scambi, sarebbe possibile ricreare la stessa sequenza e "rubare l'identità".

IPsec (IP security)

Alternativa alla creazione di una rete fisica privata, permette la creazione di **VPN (virtual private network)** sulla rete internet pubblica.

Può lavorare sia sotto UDP che TCP, incapsulando i datagrammi IPv4 in datagrammi IPsec con stessa intestazione ma payload cifrato.

Si può basare su 2 protocolli:

- AH => authentication header
- ESP => encapsulation security payload (più efficace e diffuso)

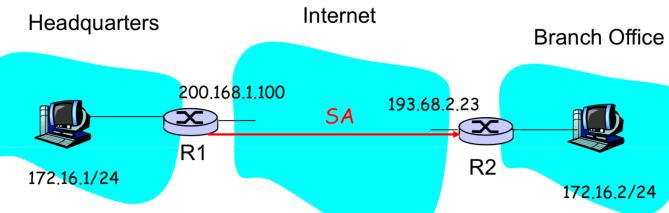
Prima di procedere con l'invio dei datagrammi, sorgente e destinazione, devono creare un canale logico a livello di rete chiamato **SA (security association)**.

I canali sono unidirezionali quindi se ne creano 2 per ogni scambio.

Quindi in una sottorete collegata con l'headquarter avente n host, esistono $(2+2n)$ SA.

Le entità IPsec memorizzano le informazioni di stato su tutte le loro SA nel **SAD (security association database)**.

Example SA from R1 to R2



R1 stores for SA

- 32-bit identifier for SA: *Security Parameter Index (SPI)*
- the origin interface of the SA (200.168.1.100)
- destination interface of the SA (193.68.2.23)
- type of encryption to be used (for example, 3DES with CBC)
- encryption key
- type of integrity check (for example, HMAC with MD5)
- authentication key

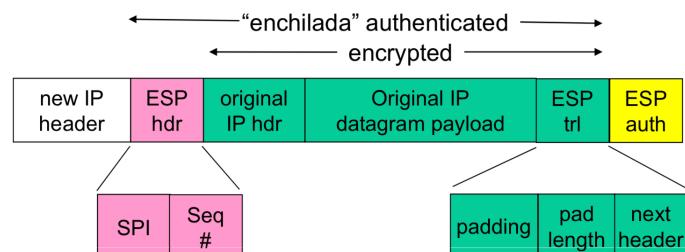
Datagramma IPsec

Per la conversione da IPv4 a IPsec:

1. Appende in fondo il campo "coda ESP" all'IPv4 originale
2. Cifra il risultato con algoritmo e chiave specificati dall'AS
3. Appende all'inizio il campo "intestazione ESP" all'IPv4 originale
4. Crea un MAC con algoritmo e chiave specificati dall'AS
5. Appende il MAC all'enchilada ed ottiene il payload
6. Crea una nuova intestazione IP classica

OSS Il datagramma creato corrisponde ad un IPv4 genuino con l'aggiunta di un payload

- ESP header => formato da:
 - SPI => indica l'entità ricevente e a quale SA appartiene il datagramma
 - Numero sequenza, per attacchi basati su ripetizioni
- ESP trailer => padding per la cifratura, next header indica il tipo di dati inseriti nel campo payload (ex. UDP)
- ESP/MAC auth => calcola il MAC su tutta l'enchilada, che consiste in un'intestazione ESP del datagramma originario, serve come chiave comune



Per stabilire se un datagramma deve essere convertito in IPsec le entità IPsec, oltre al database SAD, mantengono un'altra struttura chiamata **SPD (security policy database)**. Per la creazione automatica degli SA si utilizza il protocollo **IKE (internet key exchange)**.

Sicurezza reti wireless

Il sistema base per la sicurezza di queste reti è **WEP (wired equivalent privacy)** di IEEE 802.11, che presenta un approccio a chiave simmetrica condivisa.

Non specifica però l'algoritmo per la gestione delle chiavi, quindi host e stazione base devono concordarsi con procedure fuori banda.

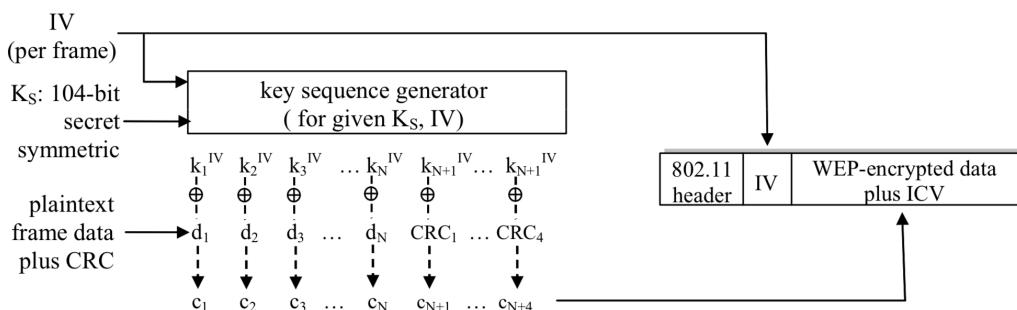
l'autenticazione avviene:

1. Terminale wireless richiede autenticazione
2. Il punto d'accesso risponde con un nonce a 128 byte
3. Host codifica nonce e chiave simmetrica condivisa, quindi lo ritrasmette
4. Punto d'accesso decifra il nonce

L'algoritmo di cifratura assume che entrambi conoscano una chiave condivisa di 104 bit a cui si aggiunge un vettore di inizializzazione (IV) a 24 bit, così da creare una chiave a 128 bit da usare per la codifica di ogni frame.

Siccome IV cambia da un frame all'altro, ognuno avrà una chiave diversa.

Quindi la cifratura avviene tramite un CRC per il payload, secondo lo schema in figura:



Avendo un IV di soli 24 bit è facile avere il riutilizzo dello stesso codice, inoltre IV è in chiaro, ovvero visibile a tutti, questo rende questo protocollo poco sicuro

IEEE 802.11i

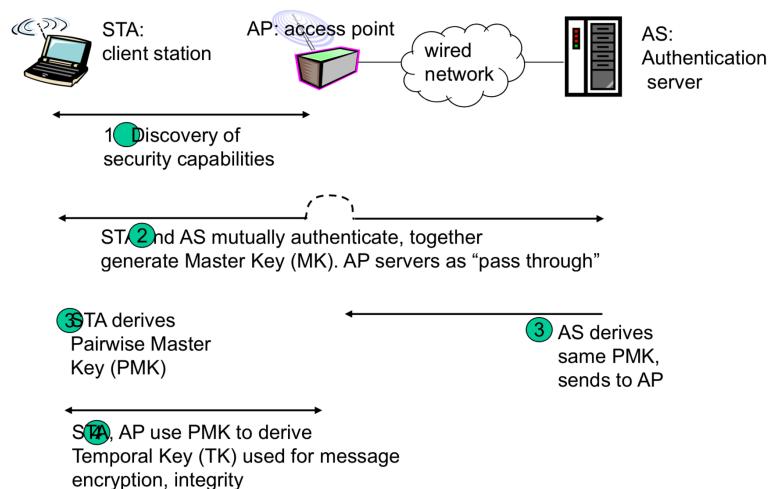
Miglioramento di IEEE 802.11, prevede 4 fasi:

1. Riconoscimento => l'AP notifica al client quali forme di autenticazione e crittografia può fornire, questo quindi indica la sua scelta
2. Mutua autenticazione e generazione **MK (master key)** => client comunica con l'AS, secondo uno standard definito dal **EAP (extendible authentication protocol)**.

I messaggi EAP sono incapsulati tramite EAPoL (EAP over LAN), inviati sul collegamento wireless, poi decapsulati dall'AP e re-incapsulati tramite il protocollo RADIUS³ per essere trasmetti all'AS.

Quindi il client genera una master key nota ad entrambi

3. Generazione master key accoppiata => nota la MS, l'AS genera la **PMK (pairwise master key)** da associare all'AP
4. Generazione chiave temporale => usando PMK si potranno generare addizionali tra cui la **TK (temporal key)** per gli host remoti

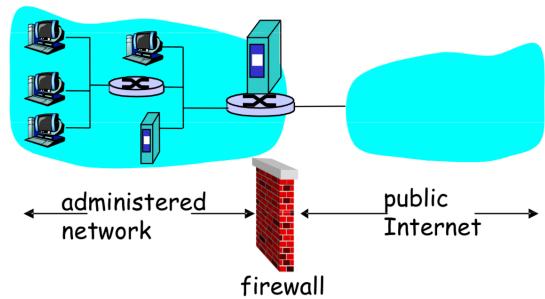


³ in fase di sostituzione con DIAMETER

Firewall

Combinazione di hardware e software che separa una rete privata dal resto di internet permettendo solo ad alcuni pacchetti di passare, e bloccando gli altri.

Permette di prevenire attacchi e fornisce l'accesso solo agli utenti autorizzati, possono essere di 3 tipi:



Stateless packet filter

Ogni rete privata è connessa al suo ISP tramite un router (con firewall), che tramite l'analisi dell'intestazione dei datagrammi, è responsabile del filtraggio dei pacchetti.

La decisione avviene analizzando:

- IP sorgente e destinazione
- TCP/UDP port number di sorgente e destinazione
- Tipo di messaggio ICMP
- TCP SYN e ACK bits

L'amministratore di rete quindi configura il firewall in base alle politiche richieste dall'organizzazione della rete privata, alcuni esempi sono:

<u>Policy</u>	<u>Firewall Setting</u>
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Le regole del firewall sono implementate nel router tramite le **ACL (access control list)** presenti su ciascuna interfaccia del router ed applicate ad ogni datagramma entrante ed uscente, un esempio di lista:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Stateful packet filter

Nella versione stateless ogni pacchetto viene filtrato indipendentemente, in questo caso invece si tiene traccia di tutte le connessioni TCP aggiungendo la tabella di connessione. Questo permette di “dare un senso” ai pacchetti, osservando se appartengono ad una connessione TCP già esistente, in questo modo è possibile bloccare i pacchetti malevoli che rispettano le regole del firewall ma non fanno parte di connessioni già esistenti. Inoltre determina quando la connessione è finita e blocca i nuovi pacchetti.

action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	✗
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	✗
deny	all	all	all	all	all	all	

Application level gateway

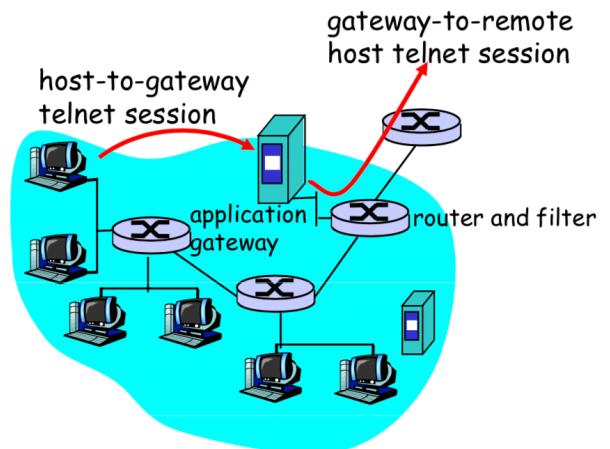
I metodi precedenti permettono il controllo solo sugli elementi di strato di rete e trasporto, questo modo invece permette di ottenere un livello di sicurezza più elevato combinando il filtraggio dei pacchetti con un gateway a livello applicativo, ovvero un server specifico attraverso il quale tutti i dati applicativi delle applicazioni sono vincolati a passare.

Per far questo generalmente si utilizzano server proxy come intermezzo tra il proprio host ed il servizio al quale si vuole accedere.

Questi infatti, conoscono le credenziali del richiedente, ma non le ritrasmettono al destinatario, mantenendo la privacy e l'anonimato.

Un esempio di questo utilizzo è il servizio TOR.

Il sistema comunque non è privo di svantaggi, poiché richiede un gateway diverso per ogni applicazione ed ha un costo in termini di prestazioni.



IDS e IPS

Invece di limitarsi solo al controllo dell' intestazione dei pacchetti, questi effettuano un controllo approfondito anche sul contenuto dei pacchetti.

I' **IDS (intrusion detection system)** genera allarmi quando osserva un traffico potenzialmente malevolo, l' **IPS (intrusion prevention system)** ne filtra il traffico.

Questi sistemi esaminano la correlazione tra più pacchetti e permettono di evitare:

- Port scanning
- Network mapping
- DoS attack

Generalmente si utilizzano più sistemi IDS nella stessa rete con funzioni leggermente diverse (in modo da non appesantire il carico), questi lavorano in maniera coordinata, mandando le informazioni a un processore IDS centrale che decide quando intervenire.

