

Reti di Calcolatori

Carmelo Gugliotta 213477

Anno 2021/2022

Contents

1 Lezione n1 03 03 2022	6
1.1 Concetti Introduttivi	6
1.1.1 Struttura di rete in breve	6
1.1.2 Reti di accesso e mezzi fisici	7
1.1.3 Il nucleo della rete	7
2 Lezione n2 04 03 2022	9
2.1 Struttura di internet	9
2.2 Panoramica del ritardo e delle perdite nelle reti a commutazione di pacchetto	9
2.2.1 Throughput	11
2.3 Introduzione organizzazione delle reti	11
2.4 Modello ISO/OSI	12
3 Lezione n3 10/03/2022	14
3.1 Modello TCP/IP	14
3.2 Incapsulamento	15
3.3 Servizi con connessione	15
3.4 Servizi senza connessione	15
3.5 Qualità del servizio	16
3.6 Confronto TCP/IP e ISO-OSI	16
3.7 Livello Applicazione	17
3.7.1 Creare un'applicazione di rete	17
3.7.2 Processi Comunicanti	17
3.7.3 Indirizzamento	17
3.7.4 Socket	17
3.7.5 Programmazione dei socket	18
3.7.6 Programmazione Socket con TCP	19
3.7.7 Programmazione Socket con UDP	20
3.8 Stream	21
3.9 Conclusioni	21
3.10 HTTP	22
3.10.1 Panoramica HTTP	22
3.10.2 Schema del tempo di risposta	22
4 Lezione n4 11/03/2022	24
4.1 Messaggi HTTP	24
4.1.1 Upload dell'input di un form	24
4.1.2 Tipi di metodi	25
4.1.3 Interazione Utente-Server: i Cookie	25
4.1.4 Cache Web (Server Proxy)	26
4.1.5 Get Condizionale	28
4.2 FTP: File transfer Protocol	28
4.2.1 Comandi e Risposte FTP	29
4.3 Posta Elettronica	29

5 Lezione n5 17 03 2022	30
5.1 SMTP	30
5.1.1 Confronto con HTTP	31
5.2 Formato dei messaggi di posta elettronica	31
5.3 Protocolli di accesso alla posta	31
5.3.1 POP3	32
5.3.2 Piccola descrizione di IMAP	32
5.4 DNS: Il servizio di directory di Internet	33
5.4.1 Servizi DNS	33
5.4.2 DNS: struttura gerarchica	34
5.4.3 DNS Caching	36
5.4.4 Record DNS	36
5.5 Applicazioni P2P	36
5.5.1 Distribuzione di file	36
5.5.2 Bit-torrent	38
5.5.3 P2P: Ricerca d'informazioni	39
6 Lezione n6 24 03 2022	41
6.1 Ancora sul P2P	41
6.1.1 Query Flooding	41
6.1.2 Copertura Gerarchica	42
6.1.3 Skype	43
6.2 Sicurezza nelle Reti	43
6.2.1 Princìpi di crittografia	44
6.2.2 Algoritmi a chiave segreta	45
6.2.3 Algoritmi moderni a chiave segreta	46
7 Lezione n7 01 04 2022	47
7.1 Algoritmi RSA e chiave pubblica	47
7.2 Confronto	48
7.3 Integrità del messaggio	48
7.3.1 Funzioni hash crittografiche	48
7.3.2 MD5 e SHA1	49
7.3.3 Codice di autenticazione dei messaggi (MAC)	50
7.4 Firme Digitali	50
7.4.1 Certificazione della chiave pubblica	51
8 Lezione n8 08 04 2022	53
8.1 Autenticazione end-to-end	53
8.1.1 Autenticazione con uso di un KDC	54
8.2 Rendere sicura la posta elettronica	55
8.2.1 E-mail sicure	55
8.3 PGP	57
8.4 Posta elettronica certificata (PEC)	58
8.4.1 PEC: Componenti	58
8.4.2 PEC: Processo	59
8.5 Cloud computing	60
8.5.1 Caratteristiche essenziali	60

9 Lezione n9 21 04 2022	62
9.1 Ancora sul Cloud Computing	62
9.1.1 Motivazioni economiche	62
9.1.2 Modelli di deployment	62
9.1.3 Modelli di servizio	63
9.1.4 Ruoli	65
9.1.5 Accesso ai sevrizi	65
9.1.6 Architetture	65
9.1.7 Virtualizzazione	66
9.1.8 Interoperabilità	66
9.1.9 Cloud vs Grid computing	67
9.2 Sistemi P2P strutturati : Introduzione alle Distributed Hash Tables	67
9.2.1 Introduzione	67
9.2.2 Analisi delle soluzioni e Confronti	68
9.2.3 DHT: Indirizzamento	69
9.2.4 DHT: Bilanciamento del carico	70
9.2.5 DHT: Routing	70
9.2.6 DHT: Memorizzazione Diretta e Indiretta	71
9.2.7 DHT: Inserzione di nuovi nodi	71
9.2.8 DHT: Ritiro/Fallimento di nodi	71
9.2.9 DHT: Applicazioni	72
9.2.10 Conclusioni	72
10 Lezione n10 22 04 2022	73
10.1 Sistemi P2P strutturati: Chord	73
10.1.1 Introduzione	73
10.1.2 Applicazioni	73
10.1.3 Topologia	73
10.1.4 Overlay	74
10.1.5 Routing	75
10.1.6 Auto Organizzazione	76
10.1.7 Conclusioni sull'inserimento/Caduta dinamica dei nodi	79
10.1.8 Fallimento di Nodi	79
10.1.9 Uscita volontaria di un nodo	80
10.1.10 Replicazione dei dati	81
10.1.11 Scelte di progetto	81
10.1.12 Simulazione	81
10.1.13 Conclusioni	81
11 Lezione n11 06 05 2022	83
11.1 Il caso dei Bitcoin	83
11.1.1 Payment transaction: Who trust whom?	83
11.1.2 Bitcoin main goal	84
11.1.3 Perchè il bitcoin divenne così popolare?	85
12 Lezione n12 13/05/2022	86
12.1 Il problema principale della Moneta Digitale	86
12.1.1 Ledger of transaction distribuito	86
12.1.2 The Bitcoin blockchain	89

12.1.3	Block and transaction	89
12.1.4	Proof of Work (Pow)	90
12.1.5	Miner tasks	92
12.1.6	How to choose the difficulty	93
12.1.7	Incentives: Block rewards	94
12.1.8	Blockchains forks	94
13 Lezione n13 19 05 2022		95
13.1	Livello di trasporto	95
13.1.1	Come funziona il demultiplexing	96
13.1.2	Trasporto senza connessione UDP	96
13.2	Protocolli con pipeline	98
13.2.1	GoBackN	99
13.2.2	Selective Repeat	99
13.3	Trasporto orientato alla connessione TCP	100
13.3.1	Panoramica	100
13.3.2	Tempo di andata e ritorno e timeout	101
13.3.3	Trasferimento dati affidabile	102
13.3.4	Gestione della connessione TCP	103
14 Lezione n14 26 05 2022		104
14.1	Livello di rete	104
14.1.1	Indirizzamento IPv4	105
14.1.2	Traduzione degli indirizzi di rete (NAT)	106
14.1.3	Port forwarding	107
14.1.4	Internet Control Message Protocol (ICMP)	107
14.2	Firewall	108
14.3	Attacchi Informatici	108
14.3.1	Denial of Service (DoS) e IP spoofing	109

1 Lezione n1 03 03 2022

1.1 Concetti Introduttivi

Una **Rete di calcolatori** è un insieme di computer indipendenti e interconnessi tra loro che possono scambiarsi informazioni. Generalmente utilizzati per la Comunicazione, Condivisione di risorse, suddivisione di compiti (Modello client-server) o E-commerce. Un esempio di reti di calcolatori che interconnette miliardi di dispositivi è Internet. I dispositivi, vengono identificati con l'appellativo **host** (Sistema Terminale o end system) che eseguono **programmi applicativi**, e sono connessi tra di loro tramite una **rete di collegamenti** e **router** (commutatori di pacchetti o packet switch). I **Collegamenti** possono essere di molti tipi, costituiti da varie tipologie di mezzi fisici, tra i più importanti si ha fili di rame, fibra ottica, onde elettromagnetiche che si differenziano tra di loro per caratteristiche come la *velocità di trasmissione* e *ampiezza di banda* (Capacità del canale). I **Router** instradano i pacchetti verso la loro destinazione finale, ove vengono riassemblati per ottenere i dati originari.

Si ricorda infatti che quando un end system vuole inviare dati, allora avviene una suddivisione degli stessi in sottoparti, ognuna caratterizzata da un intestazione, e l'insieme d'informazioni risultanti è detto pacchetto.

Internet sta per rete delle reti, costituito da una struttura gerarchica, ed è bene distinguere **Internet Pubblica** (ampia rete disponibile per tutti) dall' **Intranet Private** (rete progettata per un determinato gruppo di utenti contenente informazioni specifiche dello stesso). Inoltre Internet si presenta come una perfetta infrastruttura di comunicazione per applicazioni distribuite (web, voip, e-mail ecc) alla quale è possibile offrire una moltitudine di servizi tra i quali troviamo:

- Best Effort servizio non affidabile, senza connessione che in qualunque effettua la consegna delle informazioni, ma non si ha nessuna garanzia sulla corretta ricezione degli stessi.
- Servizio affidabile dalla sorgente alla destinazione.

Si osservi tutta l'attività di comunicazione in Internet è governata dai protocolli. Un **Protocollo** definisce il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione, così come le azioni intraprese in fase di trasmissione e/o ricezione di un messaggio o di un altro eventi.

1.1.1 Struttura di rete in breve

Quindi "tirando le somme" di ciò che abbiamo detto nelle precedenti righe, una panoramica ad alto livello di Internet è la seguente:

- **Ai confini della rete:** Applicazioni eseguite/ospitate dai sistemi terminali (Host). L'infrastruttura può basarsi su una architettura **Client/Server** o **Peer to Peer**.
 - **Client/Server:** Tra gli Host vi è una suddivisione tra client e server. I primi sono host, generalmente caratterizzati da un indirizzo IP dinamico, che richiedono servizi comunicando con il server (possono contattarlo in qualunque momento), mentre i server si occupano di erogare servizi in quanto sono sostanzialmente macchine più potenti sempre attivi con un indirizzo IP fisso.

I client non comunicano direttamente tra di loro e vi può essere una server farm per creare un unico potente server virtuale.

- **Peer to Peer:** Non ci sono Server sempre attivi ma vi sono coppie arbitrarie di host, detti anche peer, che comunicano direttamente tra di loro, non necessariamente devono essere sempre attivi, e cambiano indirizzo IP. Caratterizzato da una **Dificile Gestione** ma è **Facilmente Scalabile**
- **Nucleo Della Rete:** Una maglia di commutatori di pacchetti e collegamenti che interconnettono gli end system.
- **Reti di Accesso, Dispositivi fisici:** Collegamenti cablati e wireless.

1.1.2 Reti di accesso e mezzi fisici

Esaminiamo ora le reti di accesso (Access Network), cioè la rete che connette fisicamente un sistema al suo **Edge Router**, ovvero il primo router sul percorso dal sistema Mittente a un qualsiasi altro sistema di destinazione collocato al di fuori della stessa rete di accesso. Vi sono tre tipi di reti di accesso:

- **Reti di accesso residenziale :** punto-punto e si ha il **Modem dial-up** fino a 56 kbps (banda ridottissima) e tramite la quale non era possibile navigare e telefonare allo stesso momento. **Digital Subscriber line** caratterizzata da una linea dedicata con Prestazioni molto vantaggiose, 1 Mbps in upstream e 20+ Mpbs in downstream (ADSL versione Asymmetrical poiché privilegia il download rispetto all'upload). E infine la fibra ottica in cui distinguiamo FTTC Fiber to the Cabinet fino a 100-200 Mbps e FTTH Fiber to the home (FTTH) fino a 1 Gbps.
- **Reti di accesso Aziendale:** Reti Locali (Lan), utile per collegare sistemi terminali di aziende e università all'edge router e la tecnologia utilizzata è la Ethernet con velocità di 100Mbps, che può arrivare fino a 1 o anche 10 Gbps. Vi sono anche le LAN Wireless basate su tecnologia IEEE 802.11 (Nota come wi-fi) all'interno della quale gli utenti trasmettono/ricevono pacchetti entro un raggio di poche decine di metri verso/da un access point wireless (base station).
- **Reti di accesso wireless geografica:** Gestita da un provider di telecomunicazioni, circa 75 Mpbs per i sistemi cellulari 4g e circa 1 Gbps per i sistemi cellulari 5g

Sul pdf è presente una rappresentazione delle componenti di una tipica rete da abitazione

1.1.3 Il nucleo della rete

Il nucleo della rete è una maglia di commutatori di pacchetti e collegamenti che interconnettono i sistemi periferici di Internet. Ma come avviene nello specifico il trasferimento dei dati attraverso la rete? Si hanno due soluzioni, ognuna delle quali ha un impatto diverso sulle prestazioni a livello applicativo.

- **Commutazione di circuito:** Circuito dedicato per l'intera durata della sessione (Il funzionamento è equivalente alla rete telefonica classica). Per il funzionamento corretto è necessaria l'impostazione della chiamata (creazione di connessione) e le risorse di rete (Ampiezza di banda, capacità del commutatore) sono suddivise in "pezzi" ognuno dei quali allocato ai vari collegamenti, se non utilizzate le risorse rimangono inattive (non c'è condivisione). Le prestazioni da circuito (La banda allocata) sono garantite.

La suddivisione della banda in "pezzi" può avvenire in due modi:

- **divisione di frequenza FDM:** Le risorse sono suddivise in "slice" dedicate a ogni utente, che può utilizzare la "porzione di frequenza associata per tutto il tempo che vuole.
- **divisione di tempo TDM:** La medesima risorsa viene suddivisa rispetto al tempo. Ogni utente può utilizzare tutte le frequenze però solo durante l'intervallo di tempo finito assegnatogli.

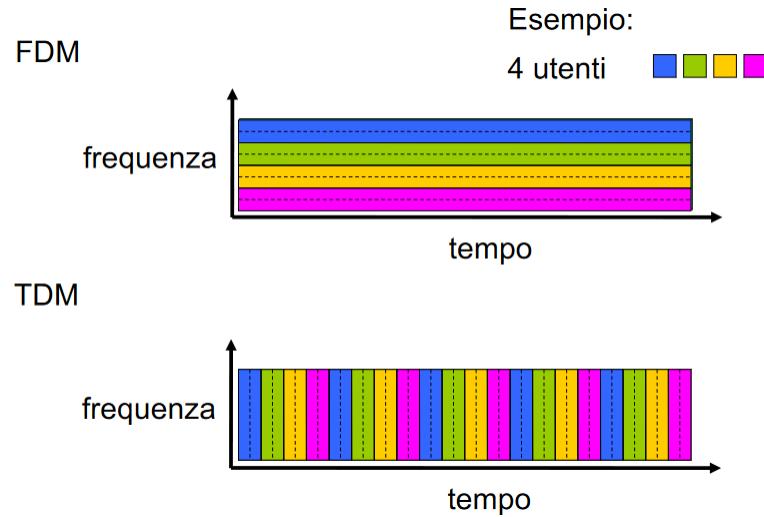


Figure 1: Esempio

- **Commutazione di Pacchetto:** I messaggi di una sessione utilizzano le risorse su richiesta e di conseguenza potrebbero dover attendere per accedere a un collegamento. In particolare il flusso di dati punto-punto viene suddiviso in pacchetti, e ciascun pacchetto utilizza completamente il canale, ma quando il pacchetto è stato trasmesso lungo il canale, quest'ultimo si libera istantaneamente, e diventa utilizzabile per la trasmissione di altri pacchetti (non necessariamente provenienti dallo stesso interlocutore).
Quindi i pacchetti degli utenti condividono le risorse, e queste vengono usate a seconda delle necessità.

Nasce però il **problema della contesa per le risorse:**

- La richiesta di risorse può eccedere il quantitativo disponibile
- **Congestione:** accodamento dei pacchetti, attesa per l'utilizzo del collegamento
- **store and forward:** il commutatore deve ricevere l'intero pacchetto prima di poter cominciare a trasmettere sul collegamento in uscita.

2 Lezione n2 04 03 2022

2.1 Struttura di internet

La struttura di internet, sia da un punto di vista fisico e sia da un punto di vista gestionale è fondamentalmente gerarchica. Costituita da un architettura che comprende un insieme di ISP(Internet Service Provider) di livello 1 che hanno una copertura nazionale/internazionale, ognuno dei quali è direttamente connesso a gli altri ISP di livello 1 (Comunicano tra di loro come "pari").

Attorno agli ISP di livello 1 vi sono gli ISP di livello 2 che coprono un area geografica più ristretta (nazionali o distrettuali) e possono connettersi solo ad alcuni ISP di livello 1 e possibilmente ad altri ISP di livello 2.

Si osserva che un ISP di livello 2 è cliente di un ISP di livello 1. Inoltre quando due ISP sono direttamente interconnessi vengono detti pari grado (peer).

Continuando con tale ragionamento, si introduce l'ISP di livello 3 e ISP locali che sono ISP di accesso, sono gli Internet Service Provider più vicini ai sistemi terminali (last hop network o meglio "ultimo salto"). Gli ISP di tale livello sono client degli ISP di livello superiore, che li collegano all'intera internet.

I pacchetti, nella commutazione di pacchetto, nel raggiungere il destinatario a partire da un determinato mittente, attraversa i vari provider.

2.2 Panoramica del ritardo e delle perdite nelle reti a commutazione di pacchetto

Il pacchetto parte da un host (sorgente), passa attraverso una serie di router, qui i pacchetti si accodano nei buffer degli stessi e possono accadere due situazioni possibili:

1. Il pacchetto resta in attesa del proprio turno per essere trasmesso (ritardo)
2. Il tasso di arrivo dei pacchetti sul collegamento eccede la capacità (perdita)

Il viaggio si conclude all'arrivo nell'host destinatario. Quindi si è visto che a ogni tappa il pacchetto subisce vari tipi di ritardo, tra i principali si hanno:

- **Ritardo di elaborazione:** Dato dal controllo di errori sui bit e sulla determinazione del canale di uscita
- **Ritardo di accomodamento:** Dato dall'attesa di trasmissione e dal livello di congestione del router.
- **Ritardo di trasmissione:** Il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento.
- **Ritardo di propagazione:** Un bit, una volta immesso sul collegamento, deve propagarsi fino al router di destinazione (di quel collegamento). Il tempo impiegato è detto ritardo di propagazione.

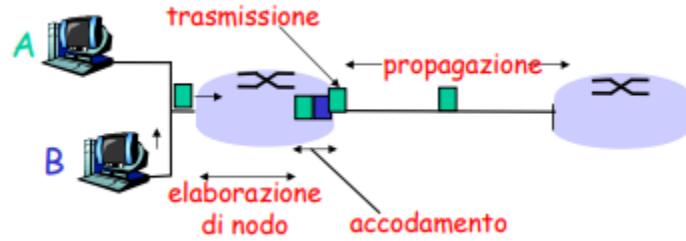


Figure 2: Ritardi

Per ottenere una misura molto vicino al reale, dei ritardi in una rete di calcolatori lungo i percorsi Internet punto-punto dalla sorgente alla destinazione, può essere utilizzato il programma diagnostico **Traceroute**. Si tratta di un semplice programma eseguibile su qualsiasi host di Internet che svolge le seguenti operazioni:

1. Invia tre pacchetti che raggiungeranno il router i sul percorso verso la destinazione.
2. il router i restituirà i pacchetti al mittente.
3. il mittente calcola l'intervallo tra trasmissione e risposta.

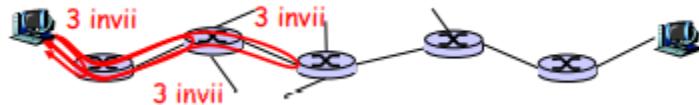


Figure 3: Esempio Traceroute

A ogni tappa (del viaggio) oltre al ritardo, potrebbe occorrere la perdita dei pacchetti in arrivo. Ciò accade perché la coda (Detta anche buffer) ha capacità finita, quando un pacchetto trova la coda piena, viene scartato (Viene perso). E il pacchetto perso può essere ritrasmesso dal nodo precedente, dal sistema terminale che lo ha generato, o non essere ritrasmesso affatto.

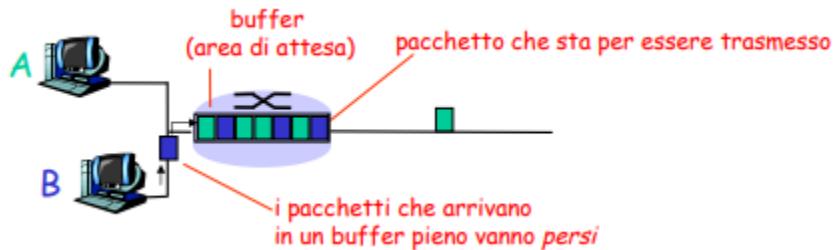


Figure 4: Perdita

2.2.1 Throughput

Oltre al ritardo e alla perdita di pacchetti, un'altra misura critica delle prestazioni in una rete di calcolatori il throughput end-to-end. Il **Throughput** è la frequenza (bit/unità di tempo) alla quale i bit sono trasferiti tra mittente e ricevente e distinguiamo tra throughput istantaneo e throughput medio.

- Istantaneo: in un determinato istantaneo
- Medio: in un periodo di tempo più lungo

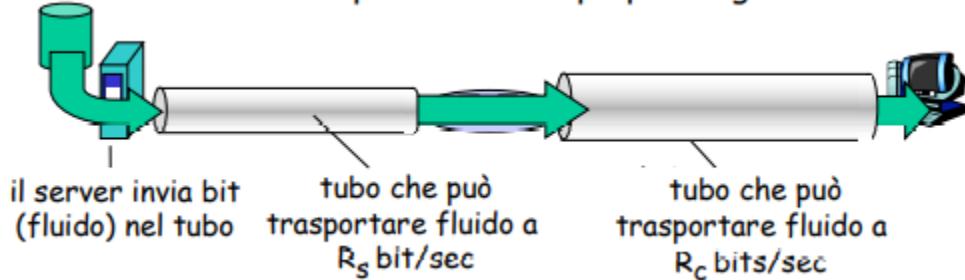


Figure 5: Esempio throughput

2.3 Introduzione organizzazione delle reti

Le reti da come si è potuto costatare, sono caratterizzate da molti pezzi (host, router, svariate tipologie di mezzi trasmissivi). Per ridurre la complessità e per ragioni di modularità le reti sono organizzate come una serie di strati o livelli. Ogni livello usa i servizi di livello inferiore. Entità remote dello stesso livello possono comunicare tra loro.

Per integrare al meglio, gli argomenti successivamente affrontati è bene tenere in mente che il **Protocollo** indica le convenzioni e le regole usate nelle comunicazioni tra due entità (processi) di uno stesso livello. Invece l'**Interfaccia** indica un insieme di servizi offerto da un livello alle entità del livello superiore (il servizio implementato dal protocollo di livello K viene fornito al livello $k+1$).

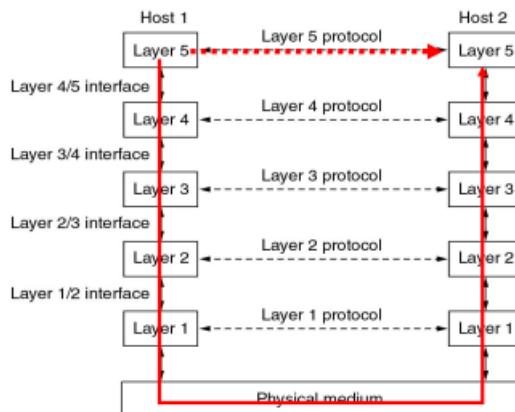


Figure 6: Architettura di rete

Logicamente il livello n di una macchina comunica con il livello n di un'altra macchina, ma in realtà non vi è una trasmissione diretta dei dati, in particolare i dati passano da un livello a quello sottostante fino al livello fisico che trasmette i dati sulla rete fino alla macchina ricevente; qui i dati effettuano il passaggio tra i diversi livelli tramite le interfacce sino ad arrivare al livello n . Un insieme di protocolli, uno per livello è detto pila di protocolli. Un insieme di livello, protocolli e interfacce è chiamato **Architettura di rete**. Le **Architetture di Rete** sono alla base di tutte le reti di calcolatori.

2.4 Modello ISO/OSI

Il Modello **International Standards Organization-Open System Interconnection** (ok fra) è composto da 7 livelli.



Figure 7: Livelli ISO/OSI

Progettato attraverso i seguenti principi guida:

- L'organizzazione a livelli definisce un grado di astrazione.
- I livelli devono corrispondere a funzioni ben definite e tra loro omogenee.
- I confini tra i livelli devono minimizzare il flusso delle informazioni tra livello e livello.
- il numero di livelli deve essere ottimale
- Le funzioni devono considerare l'insieme degli standard internazionali.

Procediamo con la descrizione dei vari livelli:

1. **Livello Fisico:** Riguarda la trasmissione dei bit sul canale fisico di trasmissione e coinvolge quindi aspetti del tipo elettrico (propagazione onde, linee comuni, comunicazione ecc) comunicazione (simplex, half-duplex, full-duplex,...) e meccanico (standard dei connettori).
2. **Livello Data-Link:** Divide le informazioni in **frame** e li trasmette attraverso il mezzo fisico. Per ogni frame inviato, il mittente attende un segnale di avvenuta ricezione (Ack) e gestisce l'eventuale duplicazione dei frame ricevuti, causata dalla possibile perdita dell'ack. Inoltre si occupa di sincronizzare un mittente veloce con un ricevente lento (**controllo di flusso**) e gestisce l'accesso al canale di trasmissione condiviso, ad esempio nelle reti locali (sottolivello MAC = Medium Access Control).

3. **Livello Rete:** Fornisce le seguenti funzionalità:

- Definisce un sistema uniforme d'indirizzamento.
- Controlla il cammino ed il flusso di pacchetti (Algoritmo di routing).
- Gestisce e controlla la congestione.
- Gestisce l'accounting dei pacchetti sulle reti a pagamento.
- Implementa l'interfaccia necessaria alla comunicazione tra reti di tipo diverso (internet-working).
- Gestisce la diffusione di messaggi a più destinazioni (multicast).

4. **Livello Transport:** Primo livello "end-to-end", gestito solo dagli host e non dai router. Ha l'obiettivo di sopperire, eventualmente, alla mancanza di affidabilità del livello rete (pacchetti persi, duplicati invertiti) e si occupa principalmente del **Servizio orientato alla connessione** e **Servizio senza connessione**.

- **Servizio orientato alla connessione:** Accetta dati dal livello superiore, li spezza in segmenti e li trasmette al destinatario, assicurando per lo meno l'ordine corretto di ricomposizione, e se possibile un servizio privo di errori, e il controllo di flusso.
- **Servizio senza connessione:** Fornisce il recapito dei messaggi senza garanzie.

5. **Livello Session:** Ha lo scopo di controllare il dialogo tra le due macchine, infatti la comunicazione è full-duplex; se non lo è questo livello tiene traccia di chi è il turno attuale. Gestisce il controllo del token nel dialogo tra più entità. Gestisce la sincronizzazione nel trasferimento dei dati (Es. checkpoint).

6. **Livello Presentation:** Permette la traduzione dei dati che viaggiano sulla rete in un formato astratto. Queste informazioni vengono poi riconvertite nel formato proprietario della macchina destinataria.

7. **Livello Application:** Lo strato applicazione comprende una varietà di protocolli utilizzati dagli utenti o dalle applicazioni.

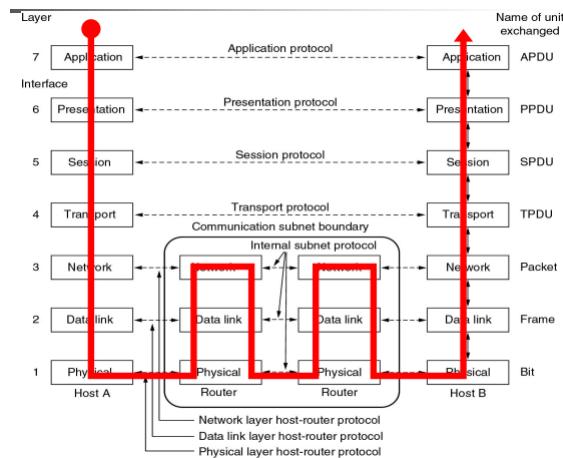


Figure 8: Modello di comunicazione ISO-OSI

3 Lezione n3 10/03/2022

3.1 Modello TCP/IP

Ancora una volta vi è l'applicazione della stratificazione dei protocolli, i progettisti organizzano i protocolli, l'hardware e il software. Il protocollo è implementato via software, hardware o con una combinazione dei due in livello o strati (layer). Ciascun protocollo appartiene a uno dei livelli ed è bene osservare che i **servizi** di un livello sono offerti a quello superiore: si tratta del **modello di servizio** (service model) di un livello, ovvero ogni livello fornisce un suo servizio (1) effettuando determinate azioni all'interno del livello stesso e (2) utilizzando i servizi del livello immediatamente inferiore.

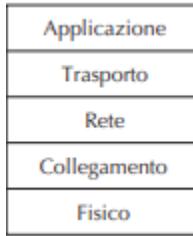


Figure 9: Pila di protocolli internet

Considerati assieme, i protocolli dei vari livelli sono detti pila di protocolli. **La pila di protocolli di Internet** consiste di cinque livelli e descriviamoli adottando un approccio top-down:

- **Applicazione:** Sede delle applicazioni di rete e dei relativi protocolli. I protocolli in tale livello forniscono supporto alle applicazioni di rete (Es FTP, SMTP, HTTP).
- **Trasporto:** Trasferisce i messaggi del livello applicazione tra il modulo client e server di un'applicazione (punti periferici gestiti dalle applicazioni). In Internet si trovano due protocolli di trasporto: TCP (servizio orientato alla connessione), UDP (servizio non orientato alla connessione). I pacchetti a livello trasporto sono detti **Segmenti**.
- **Rete:** Si occupa di trasferire i pacchetti a livello rete, detti **datagrammi**, da un host a un altro. Comprende il protocollo IP, che definisce i campi dei datagrammi e come router e sistemi periferici agiscono su tali campi e vari protocolli di instradamento che determinano i percorsi che i datagrammi devono seguire tra la sorgente e la destinazione.
- **Collegamento (Link):** Si occupa d'instradare un datagramma attraverso un serie di commutatori di pacchetto (serie di router tra la sorgente e la destinazione). Difatti per trasferire un pacchetto da un nodo a quello successivo, il livello Rete si affida ai servizi del livello collegamento.
I pacchetti in tale livello sono detti **frame** ed esempi di livello di collegamento includono Ethernet, Wi-Fi e PPP.
- **Fisico:** Il ruolo del livello fisico è trasferire i singoli bit del frame da un nodo a quello successivo. I protocolli di tale livello sono dipendenti dal tipo di collegamento e dall'effettivo mezzo trasmissivo utilizzato.

3.2 Incapsulamento

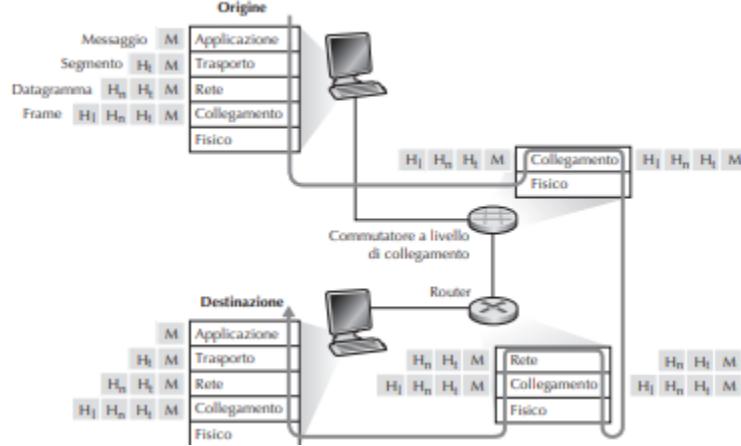


Figure 10: Incapsulamento

La figura mostra il percorso seguito dai dati scendendo lungo la pila di protocolli del sistema mittente, risalendo e scendendo lungo le pile di protocolli dei commutatori e dei router che intervengono a livello di collegamento e, infine, risalendo la pila nel sistema ricevente. Ciò mette in evidenza che a ciascun livello, il pacchetto ha due tipi di campi: quello d'intestazione e quello di **payload** (Carico utile trasportato) che tipicamente è il pacchetto proveniente dal livello superiore. Il campo d'intestazione contiene informazioni aggiuntive, utilizzate poi dalla parte ricevente (Es. bit per il rilevamento di errori).

3.3 Servizi con connessione

I servizi **connection oriented**, che permettono la consegna garantita dei messaggi e il controllo di flusso, vengono offerti tramite tre fasi:

1. Apertura della connessione tra due punti della rete
2. Utilizzo della connessione per inviare dati.
3. Chiusura della connessione

I dati sono ricevuti nello stesso ordine in cui vengono inviati e si hanno due varianti: **Stream di messaggi** e **Stream di byte**.

3.4 Servizi senza connessione

I servizi **connection-less**, ovvero un servizio senza affidabilità, né controllo di flusso né controllo di congestione in cui:

- Non c'è alcuna connessione
- I dati sono inviati impacchettati in messaggi, ognuno dei quali contiene l'indirizzo completo del destinatario.
- I messaggi non arrivano necessariamente nell'ordine in cui sono arrivati.

3.5 Qualità del servizio

QoS indica, intuitivamente, quant'è buono il servizio ottenuto da un applicazione. Ma la qualità è concetto che alle volte può risultare strettamente soggettivo, ecco perché vi è stata la necessità d'introdurre tre parametri oggettivi:

- Ordine di consegna dei dati: garantito solo dai servizi con connessione
- Ricezione garantita del messaggio: In particolare il ricevitore manda un **ack** (Ricevuta) per ogni messaggio. Se il trasmittente non riceve l'ack, può decidere d'inviare di nuovo il messaggio, successivamente allo scadere di un intervallo di tempo finito (timer). Si osservi che gli ack possono essere utilizzati sia per servizi con connessione che senza connessione.
- Consegnna corretta dei messaggi (uso di CheckSum).
- Tempi di consegna: ritardo e jitter (es. per servizi real-time multimediali).

3.6 Confronto TCP/IP e ISO-OSI

I vantaggi del modello TCP/IP sull'OSI sono fondamentalmente due:

- Quando nacque OSI, TCP/IP era già presente nel mondo accademico come supporto ad alcune applicazioni molto diffuse. (ftp,telnet)
- I protocolli del modello TCP/IP sono molto più semplici da implementare; inoltre sono open-source.

Il modello Osi è stato ed è utile per discutere e progettare le reti di computer, ma i protocolli non si sono mai affermati, al contrario di quelli del TCP/IP. I due livelli di presentazione e sessione che non erano fondamentali e le relative funzioni vengono implementate dal livello applicazione del TCP/IP. Il livello sessione gestisce la turnazione attraverso i token funzionalità che può essere implementata nel livello applicazione. C'è una corrispondenza uno a uno poi per gli altri livelli più bassi anche se Data-Link e Fisico in TCP non sono proprio distinti e sono racchiusi nel livello Host-to-network.

3.7 Livello Applicazione

3.7.1 Creare un'applicazione di rete

Un'applicazione di rete, deve necessariamente essere un software in grado di funzionare su più termini diversi caratterizzato dalla capacità di comunicare su rete. In particolare l'applicazione è costituita da una moltitudine di moduli, ognuno dei quali è in esecuzione su una macchina diversa e scopo dell'applicazione è fare in modo che questi moduli comunichino bene tra di loro.

La parte più complicata di un'applicazione di rete è gestire la comunicazione tra i moduli. Vanno quindi, definiti gli scenari e gestiti i fallimenti che possono essere di varia natura.

3.7.2 Processi Comunicanti

Il **Processo** viene definito come il programma in esecuzione su di un host. I diversi processi cooperano in diversi modi, ma in generale all'interno di uno stesso host, due processi comunicano tramite **schemi interprocesso** definiti dal S.O; Processi su host differenti, comunicano invece attraverso lo scambio di messaggi.

L'unico modo per far comunicare le varie applicazioni è tramite i socket, quindi utilizzando il protocollo TCP o UDP. Si osservi però che nonostante la possibilità di comunicazione, ogni applicazione è **indipendente** e ha la propria memoria.

Negli scenari di applicazione di rete con processi comunicanti si distinguono:

- **Processi client:** Processo che chiede il servizio ad un server (Inizio della comunicazione).
- **Processo server:** Riceve la richiesta di comunicazione e fornisce i mezzi necessari per eseguirla.

3.7.3 Indirizzamento

L'insieme degli standard e dei meccanismi per dare un indirizzo è un concetto di livello rete però in realtà anche a livello applicazione sono importanti il concetto di:

- Indirizzo IP;
- Numero di porta associato al processo in esecuzione (16 bit).

Su ogni macchina sono in esecuzione centinaia di applicazioni di rete e affinché un processo host invii un messaggio a un processo su un altro host, il mittente deve identificare il processo destinatario. L'identificazione avviene tramite la coppia precedentemente esplicitata.

I servizi standard solitamente ricevono un numero di porta compreso tra 0 e 1023 e, tale intervallo è detto **porte ben note**. I numeri di porta delle applicazioni implementate da noi vanno da 1024 in su.

3.7.4 Socket

Un **Socket** è un oggetto di trasmissione bidirezionale e può essere visto come una porta fisica attraverso cui un processo invia o riceve dati e implementano comunicazioni punto-punto. Ciò che c'è oltre la porta deve disinteressare il programmatore.

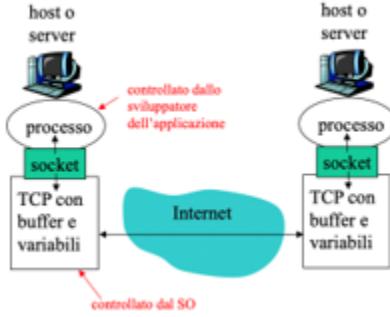


Figure 11: Esempio

Il socket quando viene creato alloca le seguenti risorse:

- Buffer di ricezione
- Item
- Risorse associate al socket

Il numero effettivo di socket dipende dal server o dall'host.

3.7.5 Programmazione dei socket

Il **Socket API**, introdotta in BSD4.1 UNIX (1981), fornisce un costrutto di programmazione denominato socket, e il processo che desidera comunicare con un'altro processo effettua operazioni fornite dall'API per inviare o ricevere dati.

Sono forniti due tipi di servizio trasporto, **orientato alla connessione e senza connessione** quindi viene utilizzato il protocollo UDP o TCP.



Figure 12: Definizione

3.7.6 Programmazione Socket con TCP

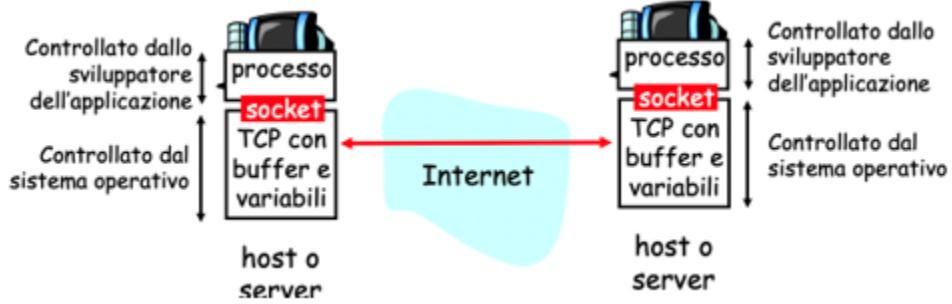


Figure 13: Schema

Innanzitutto client e server devono effettuare un'operazione di handshake e stabilire una connessione TCP. Si hanno quindi due socket, uno lato client, l'altro lato server. Una volta stabilita la connessione TCP, quando un lato vuole inviare dati all'altro deve solo mettere i dati nella connessione TCP attraverso la propria socket. Ma diamo uno sguardo più da vicino all'interazione tra i programmi client e server in TCP.



Figure 14: Diagramma di flusso della comunicazione TCP

Affinchè il server sia in grado di reagire al contatto iniziale con il client, non può essere dormiente, esso deve essere sempre pronto. Inoltre il server deve essere caratterizzato da un "Socket Speciale" detta **Server Socket** che è il punto di accesso iniziale (porta di benvenuto) per la comunicazione con tutti i client (uno per volta).

L'instaurazione della connessione TCP, dal lato client, avviene tramite la specifica della coppia indirizzo Server Socket- numero di porta processo e la creazione di una socket.

Successivamente alla creazione della socket nel client, TCP avvia un handshake a tre fasi, che ha luogo nel livello trasporto, e stabilisce la connessione TCP con il server.

Il lato server durante l'handshake a tre fasi, crea una nuova socket dedicata a quel particolare client detta **porta effimera**.

Da questo momento in poi la comunicazione avviene tramite il "condotto virtuale" stabilito tra la socket del client e la socket di connessione del server.

La comunicazione con il server socket deve essere rapida per evitare colli di bottiglia. Per tale motivo viene utilizzato un numero limitato di thread, ognuno dei quali gestisce un diverso client. Inoltre quando una porta effimera non viene più utilizzata, si libera per poter essere riutilizzata successivamente a un intervallo di tempo finito (60 o 90 sec).

3.7.7 Programmazione Socket con UDP



Figure 15: Diagramma di flusso comunicazione UDP

UDP non c'è connessione tra client e server, manca la fase del server socket perché non ci sono connessioni da accettare. Quando si utilizza UDP, vi è una porta dove chiunque mi voglia mandare dati può tranquillamente farlo. Non c'è handshaking (instaurazione della connessione). Il mittente per poter inviare il messaggio deve indicare in maniera esplicita l'indirizzo IP e la porta del destinatario. Il server UDP deve estrarre l'indirizzo IP e la porta del client che l'ha contattato. Questo sistema è inaffidabile ma allo stesso tempo leggero e di facile utilizzo.

I dati trasmessi possono perdere o arrivare a destinazione in un ordine diverso da quello dell'invio.

3.8 Stream

Dal punto di vista della macchina se questa vuole inviare dati al server remoto, quest'ultima scrivere il flusso di dati sul socket. Il flusso (stream) è una sequenza di caratteri che fluisce verso/da un processo e si distingue tra:

- Flusso in ingresso: collegato a un'origine d'input per il processo, ad esempio la tastiera o il socket.
- Flusso d'uscita: è collegato a un'uscita per il processo, ad esempio il monitor o il socket.

Si osservi il seguente esempio di applicazione client-server e la diversa soluzione con TCP e UDP.

1. Il client legge una riga dall'input standard (flusso inFromUser) e la invia al server tramite il socket (flusso outofServer).
2. Il server legge la riga del socket
3. Il server converte la riga in lettere maiuscole e la invia al client.
4. Il client legge nel suo socket la riga modificata e la visualizza (flusso inFromServer).

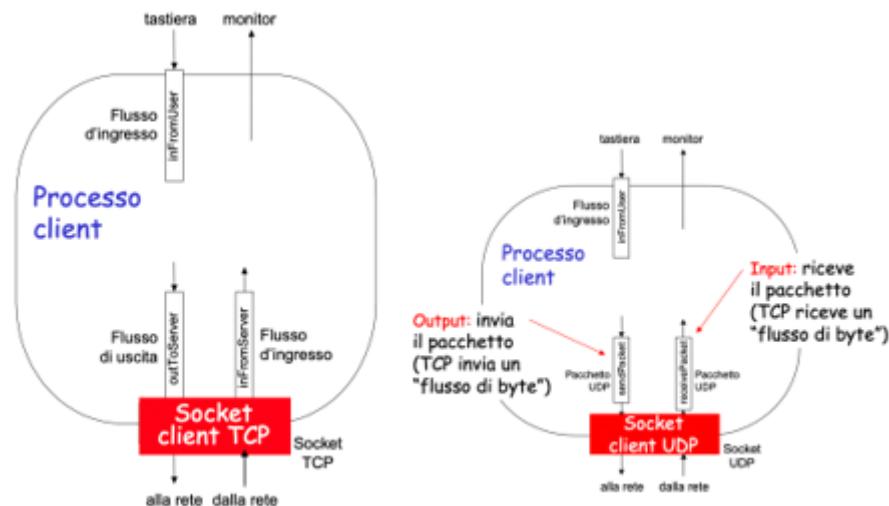


Figure 16: Programmazione dei socket con TCP/UDP

Si osservi che la comunicazione con il socket UDP si basa sull'invio/ricezioni di messaggi, ognuno dei quali indipendente dall'altro e non sugli stream.

3.9 Conclusioni

Quando abbiamo parlato di qualità del servizio abbiamo visto che questa può essere quantificata attraverso dei parametri. Se non vogliamo perdita dei dati scegliamo TCP. Se vogliamo garantire un throughput dovremmo in alcuni casi scegliere tra TCP e UDP a seconda della situazione. Ci sono problemi laterali che vanno presi in considerazione per il funzionamento dell'applicazione.

3.10 HTTP

Una pagina Web è formata da un **file base HTML** che include diversi **oggetti** referenziati, i quali possono essere file HTML, immagini JPEG o applet Java (ecc..). Ogni oggetto è costituito da un URL caratterizzato da:

- Nome dell'host
- Nome del percorso

3.10.1 Panoramica HTTP

Il protocollo è basato su un modello client-server in cui il client è il browser che richiede, riceve e visualizza gli oggetti del Web. Il server aspetta sulla porta 80 che arrivino richieste per poi inviare oggetti in risposta. HTTP usa un protocollo di trasporto di tipo TCP. Il client apre quindi una connessione TCP verso una porta 80. Il server accetta la connessione del client e poi avviene lo scambio di messaggi a livello applicazione. Una volta finita la comunicazione viene chiusa la connessione.

HTTP è un protocollo stateless (senza stato). Un protocollo si dice stateless quando il server non mantiene informazioni riguardanti le richieste precedenti fatte dal client. Generalmente utilizzato per gestire un grande numero di utenti in maniera semplice. Può adoperare un modello di connessione **persistente** e **non persistente**.

- **Non Persistenti:** Almeno un oggetto viene trasmesso su una connessione TCP
- **Connessioni persistenti:** Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server.

Si osservi che i protocolli che mantengono lo "stato" sono molto complessi, poiché necessitano di un grande quantitativo di memoria da poter allocare e perché nel caso di "guasti" deve essere garantita la riconciliazione delle visite di "stato".

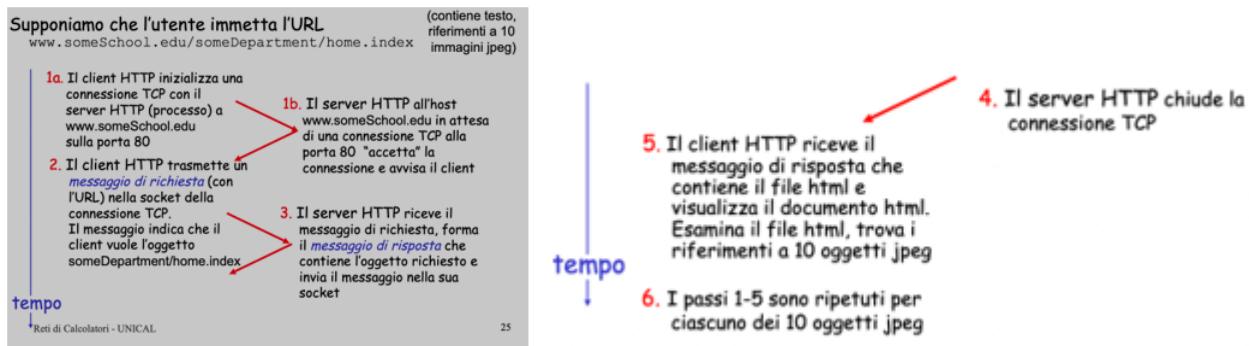


Figure 17: Esempio di connessioni non persistenti

3.10.2 Schema del tempo di risposta

Se volessimo quantificare il tempo che serve per scambiare i dati bisogna tenere conto dell'**RTT** (round trip time), ovvero il tempo impiegato da un piccolo pacchetto per viaggiare dal client al

server e ritornare al client, che è indipendente dalla dimensione del dato.

Nel caso di una connessione non persistente si ha lo svantaggio della necessità di 2RTT per oggetto:

$$TempoTotale = 2RTT + TempoTrasmissione$$

Un RTT per inizializzare la connessione TCP, un RTT perchè ritornino la richiesta HTTP e i primi byte della risposta HTTP e il tempo di trasmissione del file.

Ciò porta a un Overhead del sistema operativo per ogni connessione TCP. A tal proposito i browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati.

Al giorno d'oggi lo scaricamento di più file avviene in maniera parallela. L'alternativa è usare connessioni persistenti dove utilizzare però il parallelismo risulta più complicata. La soluzione migliore ovviamente è usare un mix delle due

4 Lezione n4 11/03/2022

4.1 Messaggi HTTP

I messaggi si dividono in due categorie:

- **Messaggio di richiesta:** sono in formato leggibile ascii leggibile dall'utente. Costituito da una prima riga, detta **riga di richiesta** formata dal comando (GET,POST,HEAD) dall'URL e dal protocollo utilizzato. Ci sono poi le **Righe di intestazione** e infine un **carriage return** e un **line feed** che indicano la fine del messaggio ("Due ritorna a capo"). La struttura generalizzata è la seguente:

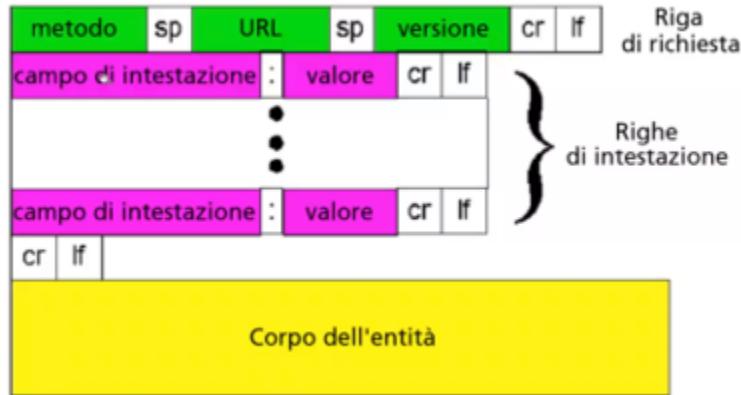


Figure 18: Struttura Messaggio di richiesta

- **Messaggio di risposta:** La logica è simile ai messaggi di richiesta. La prima riga è la **Riga di stato** formata dal protocollo utilizzato, un codice e un'espressione di stato. Poi ci sono le varie righe d'intestazione che sono duali rispetto alle righe d'intestazione della richiesta. Tra i codici di stato più utilizzati ritroviamo i seguenti:

- 200 OK: La richiesta ha avuto successo.
- 301 Move Permanently: L'oggetto richiesto è stato trasferito e la nuova locazione è specificata nell'intestazione LOCATION
- 400 Bad Request: Il messaggio di richiesta non è stato compreso dal server
- 404 Not Found: Il documento richiesto non si trova sul server
- 505 HTTP Version Not Supported: Il server non ha la versione del protocollo HTTP.

4.1.1 Upload dell'input di un form

Si hanno due metodi:

- **Post:** La pagina web spesso include un form per l'input dell'utente. L'input arriva al server nel corpo dell'entità.
- **URL:** non è un metodo inteso come comando ma perché invia i dati al server appendendo i dati all'URL della riga di richiesta. Il punto interrogativo negli URL separa l'URL vero e proprio seguito da ciò che appendiamo all'URL

Il metodo Post non pone limiti specifici sull'invio dei dati, a differenza del metodo URL, il quale si può utilizzare quando deve essere inviata un piccola quantità di dati.

4.1.2 Tipi di metodi

I metodi sono diversi a seconda della versione del protocollo.

- HTTP 1.0:
 - Get : riceve dati da un server
 - Head : chiede al server di escludere l'oggetto richiesta dalla risposta
 - Post: posta i dati sulla pagina tramite l'input da parte dell'utente
- HTTP 1.1:
 - Tutti i precedenti
 - Put: Include i file nel corpo dell'entità e lo invia al percorso specificato nel campo URL
 - DELETE: Cancella il file modificato dal campo dell'URL.

4.1.3 Interazione Utente-Server: i Cookie

Ci sono dei casi in cui l'interazione utente server richiede un concetto di **memoria** per fare in modo che il server riconosca che il client sia passato da lì per offrire contenuti adatti a quest'ultimo. La tecnologia dei cookie è basata su 4 componenti:

1. Una riga d'intestazione nel messaggio HTTP
2. Una riga d'intestazione nel messaggio di richiesta HTTP
3. Un file cookie mantenuto sul sistema terminale dell'utente e gestito dal browser dell'utente
4. Un database sul sito

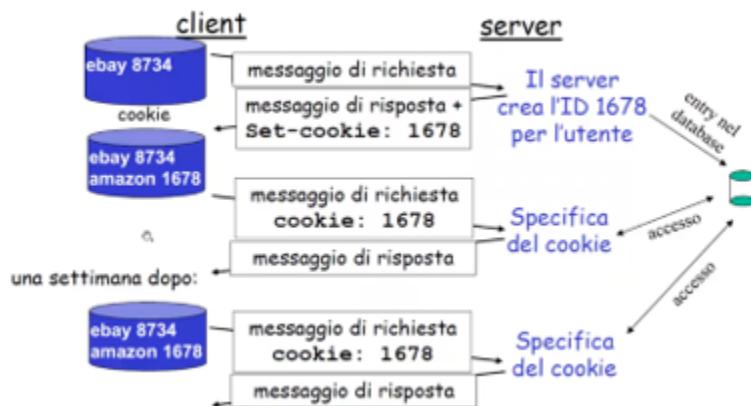


Figure 19: Esempio

I cookie possono includere una serie d'informazioni utili per la navigazione successiva come **Authorizzazioni**, **Carta per acquisti**, memorizzazione dello **Stato del mittente/ricevente** e per tale motivo sono molte volte strettamente legati a problemi di privacy.

4.1.4 Cache Web (Server Proxy)

Innanzitutto è importante sapere che la cache è una memoria in grado di mantenere dati di uso frequente. Una **cache web**, quindi, è un file system ove vengono memorizzati oggetti del web più vicini al client per evitare di riscaricarli continuamente dal server di origine. Da ciò ne consegue che le richieste HTTP non siano inviate direttamente ai vari server ma passino dal server proxy (Sempre vicino al client) che è un entità che agisce per conto di un'altra entità (Funzione di Intermediario). La **cache web** opera sia come client che come server. In particolar modo quest'ultima è installata presso la sala server dell'internet service provider o all'interno della rete aziendale.

Le motivazioni che ci spingono a usare i caching web sono:

- La riduzione dei tempi di risposta
- La riduzione del traffico sul collegamento ad accesso a internet
- Internet permette anche ai provider scadenti di fornire dati con efficacia
- Il sistema proxy determina una distribuzione dei contenuti.

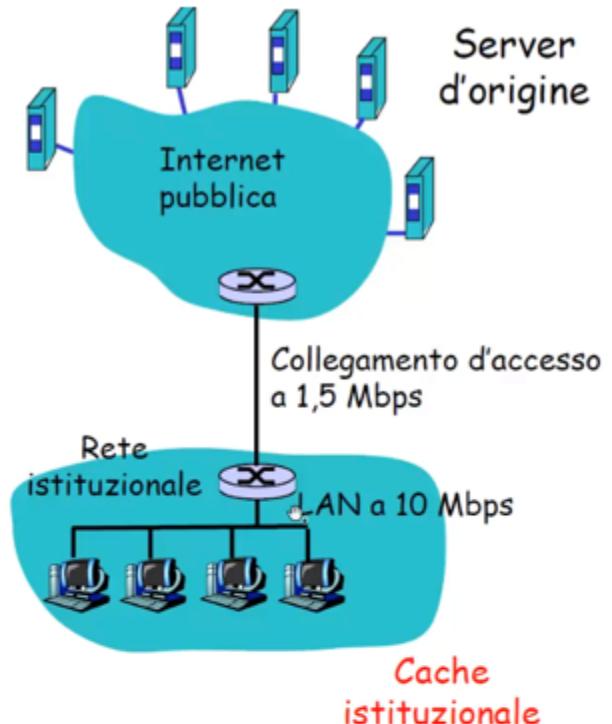
Ovviamente è bene non sottovalutare che gli oggetti presenti sul server di origine, possono cambiare e che troppe richieste potrebbero portare a un eventuale saturazione del server proxy. Tutta via il proxy può tenere traccia degli aggiornamenti dei dati memorizzati ma comunque sia presenta sempre limiti riguardante la memoria.

Ipotesi

- Dimensione media di un oggetto = 100.000 bit
- Frequenza media di richieste dai browser istituzionali ai server d'origine = 15/sec
- Ritardo dal router istituzionale a qualsiasi server d'origine e ritorno al router = 2 sec

Conseguenze

- utilizzazione sulla LAN = 15%
- utilizzazione sul collegamento d'accesso = 100%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
= 2 sec + minuti + millisecondi



Soluzione possibile

- aumentare l'ampiezza di banda del collegamento d'accesso a 10 Mbps, per esempio

Conseguenze

- utilizzo sulla LAN = 15%
- utilizzo sul collegamento d'accesso = 15%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
= 2 sec + msec + msec
- l'aggiornamento spesso è molto costoso

Soluzione possibile: installare la cache

- supponiamo una percentuale di successo (*hit rate*) pari a 0,4

Conseguenze

- il 40% delle richieste sarà soddisfatto quasi immediatamente
- il 60% delle richieste sarà soddisfatto dal server d'origine
- l'utilizzazione del collegamento d'accesso si è ridotta al 60%, determinando ritardi trascurabili (circa 10 msec)
- ritardo totale medio = ritardo di Internet + ritardo di accesso + ritardo della LAN =

$$0,6 * (2,01) \text{ sec} + \text{millisecondi} < 1,4 \text{ sec}$$

Figure 20: Esempio di Caching

4.1.5 Get Condizionale

Viene implementata dai proxy per evitare di fornire ai client versioni non aggiornate degli oggetti richiesti. L'obiettivo è non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto.

Quando il proxy riceve una richiesta dal client controlla la propria cache. La cache ha quell'oggetto che per vedere se l'oggetto è aggiornato invia una GET al server con un campo d'intestazione: *if-modified-since <data>*. Le risposte possibili sono:

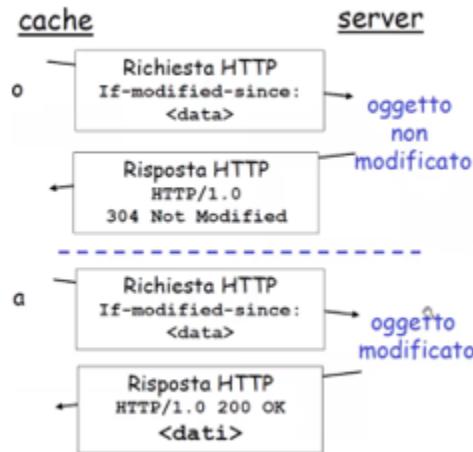


Figure 21: Esempio

In caso di modifica viene ricevuta una normale risposta HTTP dal server. In caso non ci fosse stata alcuna modifica viene inviato il codice 304 Not Modified. Questo schema funziona perché, nonostante io mi debba collegare al server, il codice 304 è ovviamente più veloce da trasferire rispetto all'intero oggetto in caso di modifica.

4.2 FTP: File transfer Protocol

File Transfer Protocol è un protocollo di trasferimento file basato su client-server, bidirezionale che effettua l'uso di TCP. In particolare si ha che il client inizia il trasferimento mentre il server è l'host remoto.

Viene effettuato l'uso di TCP e non di UDP perchè si necessita che i file arrivino a destinazione in maniera non corrotta. Inoltre viene effettuato l'uso di un UserName e di Password, per essere autorizzati all'accesso al file system remoto per poter scaricare o caricare file. Si osservi che il trasferimento dei dati e la ricezione dei comandi avviene su due socket diversi (Porta 20 per la connessione dati TCP e Porta 21 per la connessione di controllo TCP).

L'approccio utilizzato è **il controllo di connessione: fuori banda** ed è bene sottolineare che il sever FTP è **statefull**: tiene memoria delle azioni eseguite precedentemente dal client.

Comandi comuni:

- Inviati come testo ASCII sulla connessione di controllo
- **USER** *username*
- **PASS** *password*
- **LIST**
elenca i file della directory corrente
- **RETR** *filename*
recupera (*get*) un file dalla directory corrente
- **STOR** *filename*
memorizza (*put*) un file nell'host remoto

Codici di ritorno comuni:

- Codice di stato ed espressione (come in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

Figure 22: Comandi

4.2.1 Comandi e Risposte FTP

I comandi sono inviati come testo ASCII sulla connessione di controllo.

4.3 Posta Elettronica

I componenti principali della posta elettronica sono:

- **Agente Utente:** detto anche "mail reader" è il software usato dall'utente per inviare i messaggi (Es Microsoft Outlook) e si occupa della composizione, editing, lettura dei messaggi di posta.
- **Server di posta:** I messaggi in uscita o in arrivo sono memorizzati sul server. In particolare i messaggi inviati sono sul server di posta in uscita e inseriti in una coda di messaggi da trasmettere, e una volta processati i messaggi vengono inviati al server di posta in entrata corrispondente.
- Simple mail tranfer protocol: SMTP in cui si distinguono client (server di posta trasmittente) e server (Server di posta ricevente)

5 Lezione n5 17 03 2022

Continuiamo il discorso introdotto precedentemente, argomentando in dettaglio i protocolli utilizzati per la posta elettronica in internet.

5.1 SMTP

SMTP è il protocollo che costituisce il cuore della posta elettronica su Internet, e si occupa di trasferire in modo affidabile i messaggi di posta, **direttamente** dal client al server, utilizzando TCP e la porta 25.

In particolare il trasferimento si scomponete in tre fasi:

- **handshaking:** Innanzitutto il client stabilisce una connessione TCP sulla porta 25 verso il server. Una volta stabilita la connessione il server e il client effettuano, a livello applicativo, la fase di handshaking, ossia si presentano prima di effettuare scambi d'informazioni. In particolare il client specifica l'indirizzo del mittente e del destinatario.
- **Trasferimento di messaggi:** Successivamente vi è lo scambio d'informazioni e il client può contare sul servizio dati affidabile TCP.
- **Chiusura:** Al termine del trasferimento, il client ordina a TCP di chiudere la connessione.

Il server SMTP usa CRLF.CRLF per determinare la fine del messaggio.

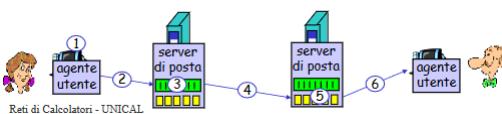
L'interazione avviene tramite una serie di comando/risposta ove i primi sono esplicitati tramite un testo ASCII e la seconda invece altro non è che un codice di stato ed espressione.

Si osservi però che SMTP è penalizzante poiché applica la restrizione all'ASCII a 7 bit non solo alle intestazioni, ma anche al corpo di tutti i messaggi di posta.

Per una migliore comprensione si allega uno scenario in cui Alice invia un messaggio a Bob.

- 1) Alice usa il suo agente utente per comporre il messaggio da inviare "a" bob@someschool.edu
- 2) L'agente utente di Alice invia un messaggio al server di posta di Alice; il messaggio è posto nella coda di messaggi
- 3) Il lato client di SMTP apre una connessione TCP con il server di posta di Bob

- 4) Il client SMTP invia il messaggio di Alice sulla connessione TCP
- 5) Il server di posta di Roberto pone il messaggio nella casella di posta di Bob
- 6) Bob invoca il suo agente utente per leggere il messaggio



Esempio di interazione SMTP

```
S: 220 someschool.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@someschool.edu>
S: 250 bob@someschool.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Bla bla
C: Bla bla bla
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 someschool.edu closing connection
```

Figure 23: Esempio

5.1.1 Confronto con HTTP

I due protocolli vengono utilizzati per trasferire file da un host a un altro. HTTP trasferisce il file (Spesso chiamati oggetti) da un web server a un web client. SMTP trasferisce file (messaggi di posta elettronica) da un mail server a un altro. Durante il trasferimento, ambedue i protocolli utilizzano connessioni persistenti e utilizzano un'interazione comando/risposta in ASCII e codici di stato (HTTP però non impone il vincolo, sul corpo, di codifica in ASCII a 7 bit), presentano quindi, caratteristiche comuni.

Tuttavia HTTP utilizza un **protocollo PULL**: Qualcuno carica informazioni su un web server e gli utenti utilizzano HTTP per scaricarle (pull) dal server, quindi TCP inizializzata dalla macchina che vuole ricevere. Al contrario SMTP utilizza un **protocollo PUSH**: il mail server invia (push) i file al mail server destinazione e la connessione TCP viene inizializzata dalla macchina che vuole inviare.

5.2 Formato dei messaggi di posta elettronica

Il formato standard dei messaggi di posta elettronica è specificato da RFC 822, che prevede le **Righe d'intestazione**, contenente informazioni di servizio, separate dal **corpo del messaggio**, costituito soltanto da caratteri ASCII, da una riga senza contenuto.

In particolare le righe d'intestazione, contengono testo leggibile, costituito da una parola chiave

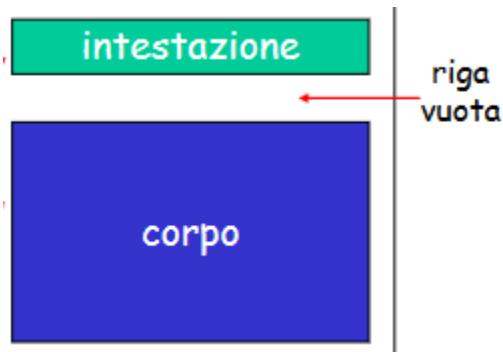


Figure 24: Esempio

seguita da due punti e un valore. Alcune parole chiavi sono obbligatorie, mentre altre sono opzionali (FROM: TO: SUBJECT: ecc) e sono differenti dai comandi SMTP, poiché fanno parte del messaggio stesso.

Si osservi che successivamente è stato introdotto MIME, ossia l'estensione di messaggi di posta multimediale, che tramite l'utilizzo di righe aggiuntive nell'intestazione permise la dichiarazione del tipo di contenuto multimediale allegato.

5.3 Protocolli di accesso alla posta

Il protocollo SMTP permette la consegna/memorizzazione sul server del destinatario. Il protocollo di accesso alla posta, invece permette di ottenere i messaggi dal server e tra questi si hanno:

- POP: Post Office Protocol.
- IMAP: Internet Mail Access Protocol
- HTTP: gmail, Hotmail, Yahoo ecc.

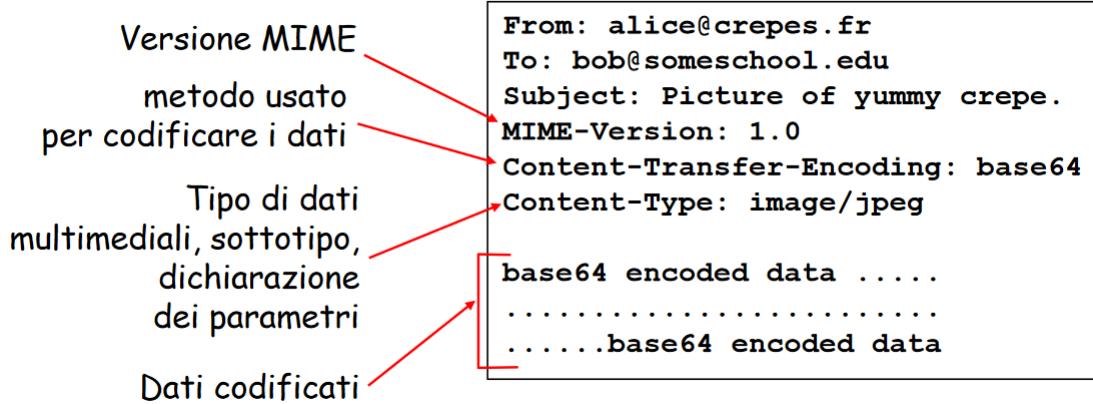


Figure 25: Esempio

5.3.1 POP3

POP3 è un protocollo di accesso alla posta estremamente semplice e di conseguenza con funzionalità piuttosto limitate. In particolare entra in funzione quando l'user agent (il client) apre una connessione TCP verso il mail server sulla porta 110. Stabilita la connessione TCP, POP3 procede in tre fasi:

1. **Autorizzazione:** L'user agent invia username e password in chiaro per autenticare l'utente e il server ha due possibili risposte **+OK** e **-ERR**. I comandi utilizzati sono i seguenti: **user** e **pass**.
2. **Transazione:** L'user agent recupera i messaggi, inoltre si ha la possibilità aggiungere/rimuovere marcatori di messaggi per la cancellazione e ottenere statistiche sulla posta. Qui lo user agent invia comandi e il server reagisce con due possibili risposte **+OK** e **-ERR**.
Uno UserAgent può spesso configurare POP3 per applicare due tipi di modalità in questa fase:
 - Scarica e Cancella: Vengono inviati i messaggi list, retr e dele, ossia elencare la dimensione dei messaggi memorizzati, recuperarli e cancellarli dal server. La problematica associata a tale modalità è che l'utente non potrà in alcun modo accedere ai medesimi messaggi da più macchine contemporaneamente.
 - Scarica e Mantieni: Lo UserAgent lascia i messaggi sul server di posta dopo averli scaricati e in questa casistica l'utente potrà accedere ai medesimi messaggi da più macchine contemporaneamente.
3. **Aggiornamento:** Tale fase si ha successivamente all'avvio da parte del client della chiusura della sessione POP3 e applica le modifiche effettuate durante la seconda fase.

Durante una sessione tra uno user agent e un mail server, POP3 mantiene alcune informazioni di stato, difatti tiene traccia dei messaggi dell'utente marcati come cancellati. Tuttavia non trasporta informazioni di stato tra le varie sessioni e questa mancanza di stato persistente semplifica di molto l'implementazione.

5.3.2 Piccola descrizione di IMAP

Per risolvere i problemi di organizzazione dei messaggi di posta, viene introdotto IMAP, un protocollo di accesso alla posta più complesso ma con maggiori potenzialità rispetto a POP3. In

particolare IMAP mantiene tutti i messaggi in unico posto, ossia il server, e consente di organizzare i messaggi in cartelle garantendo inoltre la conservazione dello stato dell’utente tra le varie sessioni: I nomi delle cartelle e l’associazione tra identificatori dei messaggi e nomi delle cartelle sono disponibili su tutte le macchine.

5.4 DNS: Il servizio di directory di Internet

Per identificare host e router di Internet può essere utilizzato l’**hostname**, ad es www.google.com, i quali risultano appropriati per l’uomo, ma forniscono poca informazione sulla loro collocazione all’Internet. Per tali motivi gli host vengono identificati anche dai cosiddetti **Indirizzi IP** (32 bit) gerarchico, perché leggendolo da sinistra verso destra si ottengono informazioni sempre più specifiche sulla collocazione dell’host in internet.

Al fine di conciliare perfettamente, i due approcci proposti è necessario un servizio che consente di **risolvere** i nomi, ossia tradurre i nomi degli host nei loro indirizzi IP. Si tratta del principale compito del **domain name system** DNS di Internet.

DNS è un database distribuito implementato in una gerarchia di DNS server e un protocollo a livello applicazione consente agli host di interrogare il database.

Si osservi che i DNS generalmente sono macchine UNIX che eseguono software BIND e il protocollo DNS utilizza UDP e la porta 53.

DNS è un protocollo a livello applicazione che viene utilizzato tra sistemi periferici che comunicano adottando il modello client-server e si affida al protocollo di trasporto end-to-end per trasferire messaggi.

5.4.1 Servizi DNS

Come abbiamo già riportato DNS permette la traduzione degli hostname in indirizzi IP, ma oltre a ciò mette a disposizione ulteriori servizi:

- **Host aliasing:** Un host dal nome complicato può avere uno o più sinonimi (alias). In tal caso il nome viene definito **Hostname canonico** e i sinonimi permettono una maggiore accessibilità maggiore essendo generalmente più facili da ricordare. (relay1.west.cost.com e www.cost.com).
- **Mail server aliasing:** Per ovvi motivi gli indirizzi di posta elettronica devono necessariamente essere facili da ricordare. Tuttavia l’hostname del server di posta è molto più complicato e assai meno facile da ricordare rispetto a un semplice gmail.com . In tal caso il server di posta invoca il DNS per ottenere il nome canonico.
- **Load Distribution:** La Distribuzione del carico di rete permette di distribuire il carico tra server replicati. In particolare i server con molto traffico, vengono replicati su più server, ognuno eseguito su un host diverso con un indirizzo IP differente e il DNS si occupa di associare l’hostname canonico a questo insieme di indirizzi.

Da ciò che è stato riportato precedentemente ne consegue che il DNS dal punto di vista dell’applicazione nell’host utente, è una scatola nera che fornisce un servizio di traduzione semplice e diretto. Tuttavia la scatola nera è costituita da un elevato numero di DNS server distribuiti e un protocollo a livello applicazione che permette la comunicazione tra DNS server e host richiedenti. Il tutto sarebbe più semplice, dal punto di vista progettuale si intende, se vi fosse un unico DNS server contenente tutte le corrispondenze. In questo schema centralizzato, i client dirigerebbero

le proprie richieste a un singolo server e quest'ultimo risponderebbe direttamente. Puttropo tale soluzione è inappropriata in Internet per i seguenti motivi:

- Se il DNS server si guasta, ne soffre l'intera Internet **Singolo punto di guasto**.
- Un singolo DNS dovrebbe gestire tutte le richieste **Volume di Traffico**.
- Un singolo DNS server non può essere contemporaneamente vicino a tutti i client **Database centralizzato distante**.
- Il singolo DNS sarebbe vasto poiché conterebbe record relativi a tutto l'internet caratterizzato da frequentissimi aggiornamenti per tracciare tali record **Manutenzione**.

In conclusione, un database centralizzato su un singolo DNS server non sarebbe in grado di gestire la crescita esponenziale della rete (NON SCALABILE).

5.4.2 DNS: struttura gerarchica

Per trattare la scalabilità, il DNS utilizza un grande numero di server, organizzati in maniera gerarchica e distribuiti nel mondo. Difatti nessun DNS server ha le corrispondenze per tutti gli host in Internet, che sono invece distribuite tra tutti i DNS server.

In maniera approssimativa quando un client DNS vuole determinare l'indirizzo IP relativo a un determinato hostname, quest'ultimo interroga il **server radice** che restituisce uno o più indirizzi IP realtivi al **Server TLD** per il dominio di primo livello. Contattato il server TLD, quest'ultimo restituisce uno o più indirizzi IP al **server autoritativo** dell'hostname specificato che una volta contattato restituisce l'indirizzo IP richiesto originariamente.

Quindi dall'esempio proposto si carpirà che esistono tre classi di DNS server:

- **Root Server**: In internet esistono 400 root server (server radice) dislocati in tutto il mondo. Forniscono gli indirizzi IP dei server TLD.
- **Top-level domain**: TLD server si occupano dei domini di primo livello quali com ,org,net,edu e gov e di tutti i domini locali di alto livello quali it,uk,fr. Forniscono gli indirizzi IP dei server autoritativi.
- **Authoritative server**: Ogni organizzazione dotata di host Internet pubblicamente accessibili (server web e i server di posta) deve fornire i record DNS di pubblico dominio che associano i nomi di tali host a indirizzi IP. Il server autoritativo dell'organizzazione ospita questi record.
Si osservi che il server autoritativo può essere implementato e gestito dall'organizzazione o può pagare un fornitore di servizi per ospitare su un suo server i record.

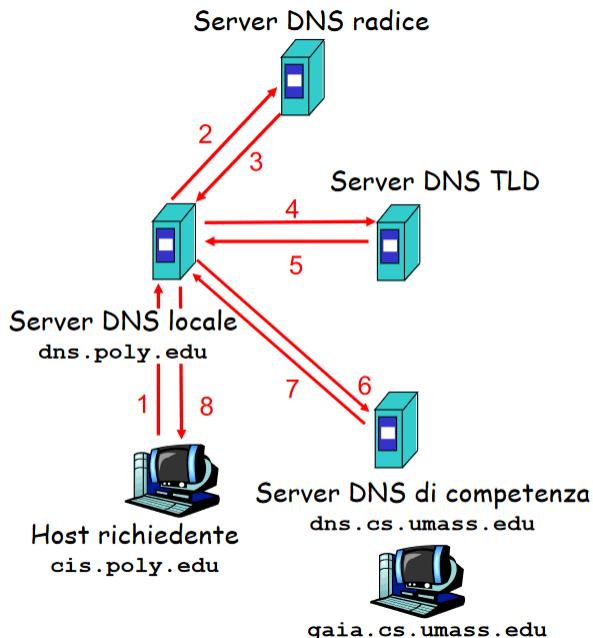
Tuttavia vi è un ulteriore tipologia di DNS server, ossia il **Server DNS locale** che non appartiene strettamente alla gerarchia dei server, ma che comunque è centrale nell'architettura DNS. Difatti ciascun ISP (università, società) ha un server DNS locale detto anche "default name server" e quando un host effettua una richiesta DNS, la query viene inviata al suo server DNS locale che opera da proxy e inoltra la query in una gerarchia di server DNS.

Esempio

- L'host cis.poly.edu vuole l'indirizzo IP di gaia.cs.umass.edu

Query iterativa:

- Il server contattato risponde con il nome del server da contattare
- "Io non conosco questo nome, ma puoi chiederlo a questo server".



Esempio

Query ricorsiva:

- Affida il compito di tradurre il nome al server DNS contattato

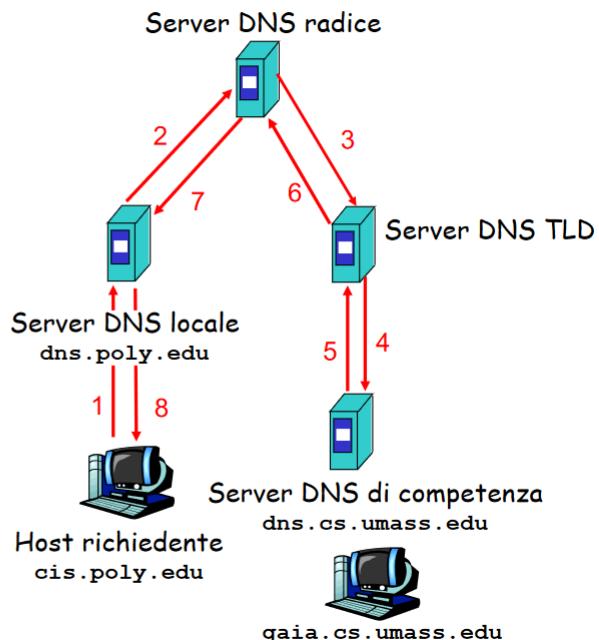


Figure 26: Esempi

5.4.3 DNS Caching

Una caratteristica di fondamentale importanza è il DNS caching che permette di migliorare le prestazioni di ritardo e il numero di messaggi DNS che "rimbalzano" in Internet. In particolare una volta che un server DNS impara una traduzione da hostname a indirizzo IP, mantiene in cache le nuove informazioni ottenute. Tuttavia quest'ultime vengono invalidate dopo un certo periodo di tempo.

Generalmente un server DNS memorizza nella cache gli indirizzi IP dei Server TLD e di conseguenza i server DNS root non vengono visitati spesso.

5.4.4 Record DNS

I server che implementano il database distribuito di DNS memorizzano i cosiddetti **record di risorsa** (RR) che forniscono le informazioni a cui si vuole accedere. Ogni messaggio di risposta DNS trasporta uno o più record di risorse che contiene i seguenti campi:

$$(Name, \ Value, \ Type, \ TTL)$$

TTL indica il time to live, ossia il tempo residuo di un record e determina quando una risorsa deve essere rimossa dalla cache. I campi di name e value dipendono dal valore di Type.

- Type=A : name indica il nome dell'host e value l'indirizzo IP.
- Type=NS : name indica il dominio e value è il nome dell'host del server di competenza di quel dominio.
- Type=CNAME : name indica l'alias di qualche nome canonico e value è il nome canonico.
- Type=MX : value è il nome del server di posta associato a name.

5.5 Applicazioni P2P

Tutte le applicazioni descritte fino ad ora (posta elettronica, web e DNS), utilizzano un architettura client-server con una significativa dipendenza da un'infrastruttura di server sempre attivi. Tuttavia nell'architettura peer-to-peer questa dipendenza è minima o del tutto assente. In particolare non vi è un server sempre attivo e le coppie arbitrarie di host (peer) comunicano direttamente tra di loro.

Si osservi inoltre che i peer non appartengono ai fornitori dei servizi, ma sono computer fissi o portatili controllati dagli utenti e per tale motivo non devono necessariamente essere sempre attivi e possono cambiare indirizzo IP.

Nelle prossime pagine affronteremo tre argomenti chiave : **Distribuzione di file, Ricerca di informazioni e Skype**.

5.5.1 Distribuzione di file

Si supponga di voler trasferire un file voluminoso da un singolo server a un gran numero di host, detti peer. In una distribuzione di file client-server, il server deve inviare una copia del file a ciascun peer, utilizzando un'elevata quantità di banda. In una distribuzione di file P2P ciascun peer, ha la possibilità di ridistribuire qualsiasi porzione del file abbia ricevuto, aiutando così il server nel processo di distribuzione.

Mettiamo a confronto i due approcci, evidenziando la quantità di tempo stimata per distribuire un file a N peer e a tale scopo ipotizziamo con abbondante ampiezza di banda e introduciamo le seguenti variabili:

- u_s : frequenza di upload del collegamento di accesso del server
- u_i : frequenza di upload del collegamento di accesso dell'i-esimo peer
- d_i : frequenza di download del collegamento di accesso dell'i-esimo peer.
- F : dim File
- N : numero di peer.

Si ottengono quindi le seguenti due casistiche:

- **Client-Server:**

Il server invia in sequenza N copie e impiega

$$Tempo_{server} = \frac{N * F}{u_s}$$

Il client i-esimo per scaricare impiega:

$$Tempo_{client} = \frac{F}{d_i}$$

Quindi il tempo totale per distribuire F a N client utilizzando l'approccio client server è di

$$D_{cs} = \max\{NF/u_s, F/\min(d_i)\}$$

- **P2P:**

Il server invia il file ciacun bit del file almeno una volta nel collegamento e impiega quindi:

$$Tempo_{server} = F/u_s$$

Il client i-esimo per scaricare impiega

$$Tempo_{client} = \frac{F}{d_i}$$

Si osservi però che la capacità totale di upload del sistema nel suo complesso è uguale alla velocità di upload del server più quella di ciacun peer:

$$u_{tot} = u_s + u_1 + \dots + u_n$$

Inoltre lo stesso sistema deve consegnare (upload) F bit a ciascuno degli N peer, consegnando quindi un totale di NF bit. E ciò non può essere fatto a una velocità più grande di u_{tot} . Quindi il tempo di distribuzione minimo è pari:

$$D_{p2p} = \max\{F/u_s, F/\min(d_i), N * F/u_{tot}\}$$

Si allega una figura in grado d'illustrare il confronto tra il tempo di distribuzione minimo per le architetture client-server e P2P, supponendo che tutti i peer abbiano la medesima velocità di upload.

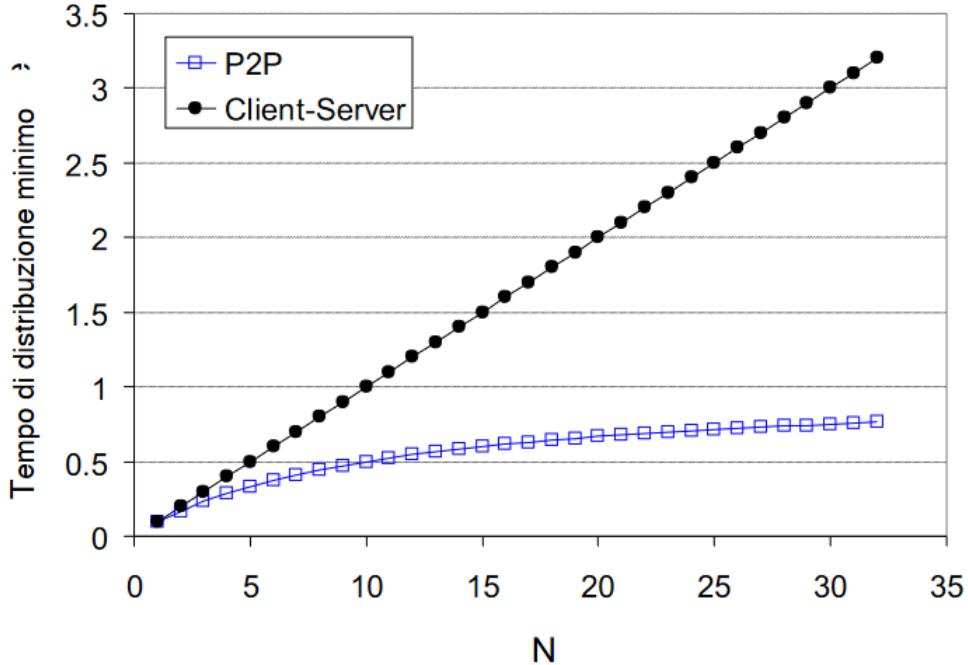


Figure 27: Grafico

Si osserva che in Client-Server il tempo di distribuzione cresce linearmente e senza limite al crescere del numero di peer. Inoltre il tempo di distribuzione dell'architettura P2P, il tempo di distribuzione minimo non è solo sempre minore di quello dell'architettura client-server, ma è anche minore di un'ora per qualsiasi numero di peer N.

Si ricava quindi che le applicazioni con architettura P2P possono essere scalabili e la scalabilità è una diretta conseguenza del fatto che i peer ri-distribuiscono i bit oltre a scaricarli.

5.5.2 Bit-torrent

Bit-torrent è un diffuso protocollo P2P per la distribuzione di file. In particolare si ha che **Torrent**(Torrente) è il gruppo di peer che partecipano alla distribuzione di un particolare file e ciascun torrent ha un nodo d'infrastruttura chiamato **Tracker** che tiene traccia di tali peer, scaricano **Chunk** (parti) del file di uguale dimensione (tipicamente di 256kbyte).

Quando un peer entra a far parte di un torrent, si registra presso il tracker, che periodicamente lo informa che è ancora nel tracker. Inoltre appena connesso non possiede nessuna parte del file, ma le accumula col passare del tempo e mentre scarica tali parti, carica gli stessi su altri peer.

Una volta ottenuto l'intero file (Può uscire ed entrare a piacimento dal torrent), il peer può lasciare il torrent (egoisticamente) o (altruisticamente) rimanere collegato.

BitTorrent (2)

- In un dato istante, peer diversi hanno differenti sottoinsiemi del file
- periodicamente, un peer (Alice) chiede a ciascun vicino la lista dei chunk che possiede
- Alice invia le richieste per le sue parti mancanti:
 - ❖ Adotta la tecnica del *rarest first*
- Alice invia le sue parti a quattro vicini, quelli che attualmente le stanno inviando i propri chunk alla frequenza più alta
 - ❖ i 4 favoriti vengono rivalutati ogni 10 secondi
- ogni 30 secondi seleziona casualmente un altro peer, e inizia a inviargli chunk
 - ❖ Il peer appena scelto può entrare a far parte dei top 4
 - ❖ A parte i "top 4" e il "nuovo entrato", gli altri peer sono "soffocati", cioè non ricevono nulla.

Figure 28: Esempio Bit-torrent

5.5.3 P2P: Ricerca d'informazioni

L'indice nei sistemi P2P permette di ottenere la corrispondenza tra le informazioni e la loro posizione negli host.

- **File Sharing:** L'indice in tale casistica tiene traccia dinamicamente della posizione dei file che i peer condividono. In particolare i peer comunicano all'indice ciò che possiedono e in caso di richiesta di accesso a un determinato file consultano sempre l'indice per determinare la sua posizione.
- **Messaggistica istantanea:** L'indice crea la corrispondenza tra utenti e posizione. Quando l'utente lancia l'applicazione, informa l'indice della sua posizione. I peer consultano l'indice per determinare IP dell'utente.

Napster fu uno dei progetti P2P di "prima generazione" che permise un trasferimento dei file distribuiti (basandosi appunto sulla tecnologia P2P), ma che necessitava per il funzionamento di un **Server directory centralizzato**, ossia si basava su un processo di localizzazione dei file fortemente centralizzato.

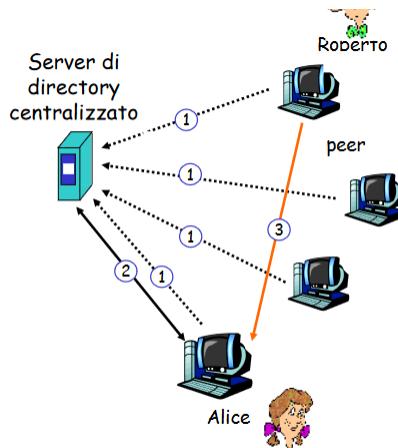


Figure 29: Esempio

La dipendenza da tal tipo di processo costituiva un enorme problematica, poiché ciò costituiva un **Collo di bottiglia per le prestazioni** e inoltre si aveva un **unico punto di guasto**.

Tuttavia Napster fallì non per tali motivi, ma per il supporto alla violazione del diritto d'autore (aiutava la gente a farlo).

Perciò, come vedremo nella prossima lezione, si è provato a elaborare un sistema privo di server centrale.

6 Lezione n6 24 03 2022

6.1 Ancora sul P2P

Prima di continuare con il discorso introdotto nella scorsa lezione è bene osservare i seguenti concetti riguardanti una **Rete di copertura: grafo**

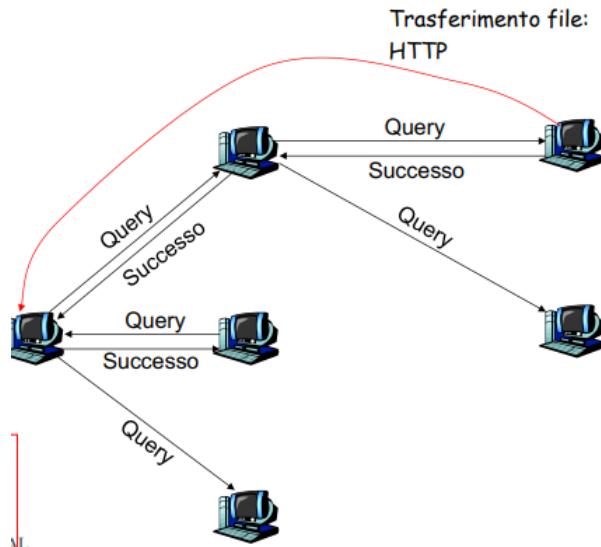
1. Una rete di copertura è formata da archi e peer attivi.
2. Vi è un arco tra i peer X e Y se c'è connessione TCP.
3. Un arco è un collegamento virtuale e non fisico, da ciò ne consegue che il significato di vicinanza si basa non sulla distanza ma dal numero di hop.
4. Un dato peer è solitamente connesso con meno di 10 peer vicini nella rete di copertura.

6.1.1 Query Flooding

Query Flooding è un protocollo di pubblico dominio utilizzato da Gnunetella che permise di risolvere il problema precedentemente proposto, ossia lo sviluppo di un sistema completamente distribuito che non necessitava quindi di un server centrale.

In particolare, ciascun peer indica i file che vuole rendere disponibili per la condivisione e le ricerche, come suggerisce il nome stesso (innondazione di query), vengono inoltrate sulla rete da ogni nodo ai propri vicini, che a loro volta la inoltrano ai propri vicini e così via.

Si osservi il seguente esempio: Il messaggio di richiesta viene trasmesso sulle connessioni TCP



84

Figure 30: Esempio

esistenti. Se i peer, non possiedono la risorsa richiesta, inoltrano il messaggio verso ulteriori vicini. Il messaggio di successo, (non la risorsa, che viene scaricata direttamente) viene poi trasmessa sul percorso inverso ed eventualmente aggregata ad altri messaggi di successo incontrati durante il percorso.

L'aggregazione dei messaggi di successo, permette di ricevere un'unica risposta invece di una moltitudine di risposte evitando così problematiche da eventuali sovraccaricamenti.

Ma non esistendo un server dedicato, come avviene l'accesso alla rete da parte di un peer? Il peer entrante deve necessariamente conoscere l'identità di almeno un peer già connesso e a tale scopo in Gnunetella viene utilizzata la lista dei peer candidati. Successivamente avviene **L'esplorazione del vicinato** effettuata, in modo non controllato e instradando messaggi "PING" sulle connessioni TCP (esistenti grazie alla lista precedentemente citata). Tutti i peer che ricevono il messaggio PING rispondono al "peer entrante" con un messaggio Pong e danno la possibilità a quest'ultimo di impostare connessioni TCP addizionali.

Tuttavia tale approccio, essendo applicato a una rete totalmente decentralizzata e non strutturata, presenta delle criticità:

- La rete di copertura, generalmente è un grafo ciclico da ciò ne consegue che uno stesso peer potenzialmente potrebbe ricevere più volte la medesima richiesta (Costo quadratico)
- Potrebbe occorrere un problema di circolazione infinita di pacchetto (Messaggio di richiesta), tuttavia prevenuto con il campo TTL. Ma il valore di TTL non sempre è facile da determinare e dipende dalla dimensione della rete e dal numero di vicini.

6.1.2 Copertura Gerarchica

Per risolvere le criticità viste si è proposta una **Copertura Gerarchica**, che combina le migliori caratteristiche di indice centralizzato e query flooding.

- Ogni peer è un **leader di gruppo** o è **assegnato a un leader di gruppo**
 - Connessione TCP tra un peer e il suo leader di gruppo.
 - Connessione TCP tra qualche leader di gruppo.
- Il leader di gruppo tiene traccia del contenuto di tutti i suoi figli

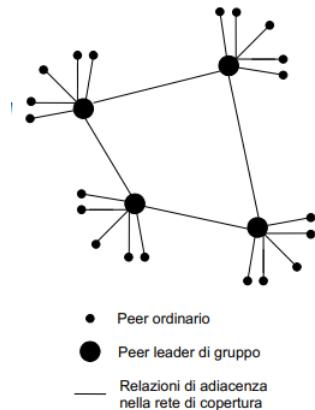


Figure 31: Rete di copertura

6.1.3 Skype

Skype è un'applicazione intrinsecamente P2P, poiché permette la comunicazione tra coppie utenti, ed è un protocollo proprietario (Non di pubblico dominio) dedotto mediante reverse engineering. Inoltre è caratterizzato da una copertura gerarchica con i supernodi e da un login server che indicizza username e indirizzi IP degli utente.

Tuttavia si ha un problematica: quando viene utilizzato il NAT, questo impedisce ad un host al di fuori della rete domestica di creare una connessione con un host all'interno di questa. Per ovviare a tale problematica si usa un relay che permette la comunicazione con NAT.

6.2 Sicurezza nelle Reti

Quando parliamo di sicurezza nelle reti è importante identificare le proprietà auspicabili per la **Comunicazione Sicura**:

- **Riservatezza:** Solo mittente e destinatario devono comprendere il contenuto del messaggio. Quindi si ha la necessità dell'invio di messaggi cifrati e dell'utilizzo del codice di decifratura, il quale scambio non è così semplice.
- **Autenticazione:** Mittente e destinatario devono essere sicuri della loro identità.
- **Integrità del messaggio:** Mittente e destinatario devono essere sicuri che il contenuto non subisca alterazioni durante la trasmissione (per cause fortuite o per manipolazioni).
- **Disponibilità e controllo dell'accesso:** Un servizio deve essere accessibile a chi è legittimamente autorizzato.

Se non si adottano tali proprietà, un malintenzionato potrebbe procedere con le seguenti azioni:

- **Spiare:** Intercettare i messaggi.
- **Aggiungere** messaggi e sovraccaricare il sistema.
- **Impersonare** un altro soggetto.
- **Dirottare** una sessione in corso e sostituirsi al mittente o al destinatario
- **Negare il servizio**

Tuttavia tali reali minacce possono essere contrastate grazie alla crittografia, il quale utilizzo non solo fornisce riservatezza, ma è un importante strumento per il servizio di autenticazione e per l'integrità dei messaggi.

6.2.1 Princìpi di crittografia

I princìpi di crittografia permettono al trasmittente di mascherare i dati in modo che un intruso non possa comprendere il contenuto. Il ricevente, naturalmente, deve invece essere in grado di recuperare i dati originali.

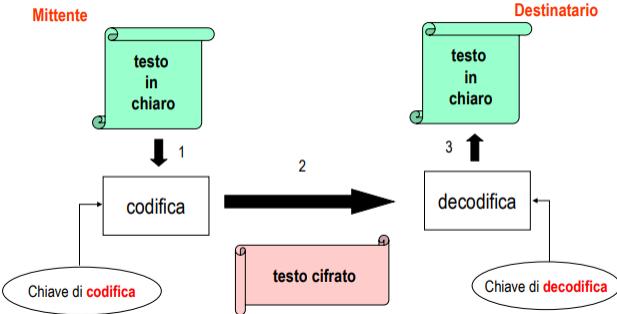


Figure 32: Esempio

Dall'esempio proposto si carpisce che la **Codifica** ha l'obiettivo di ricevere il testo in chiaro e restituire il testo codificato. Invece la **Decodifica** riceve il teso codificato e restituisce il testo in chiaro. Tuttavia ambedue sono basate sull'utilizzo di un algoritmo (pubblico) e la chiave e la sicurezza è data proprio dalla robustezza del primo e la segretezza della seconda.

Si hanno due approcci:

- Crittografia a chiave simmetrica:

Viene utilizzata la medesima chiave per codifica e decodifica. Di conseguenza Segretezza, autenticazione, integrità dipendono strettamente dalla segretezza della chiave, che generalmente sono di 64/128 bit.

Quindi ad ogni coppia di utenti deve essere distribuita una chiave

$$n \text{ utenti } O(n^2) \text{ chiavi}$$

Il problema principale riguarda lo scambio della chiave.

Crittografia simmetrica

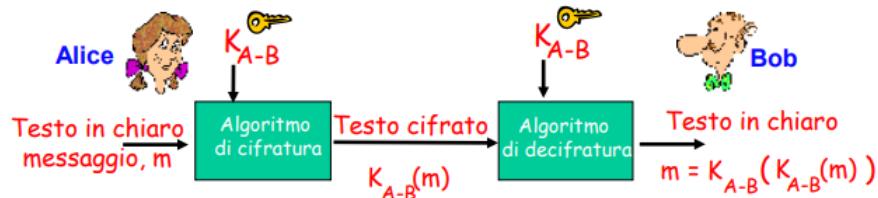


Figure 33: Chiave Simmetrica

- Crittografia a chiave asimmetrica:

Viene utilizzata una chiave per la codifica e un'altra per la decodifica. In particolare ogni utente ha due chiavi:

- **Chiave privata:** deve rimanere segreta
- **Chiave pubblica:** Informazione da diffondere

Le due tipologie sono utilizzate sia per codificare che per decodificare e generalmente si utilizzano chiavi di 1024-2048 bit. Tale approccio permette secretezza autenticazione e integrità.

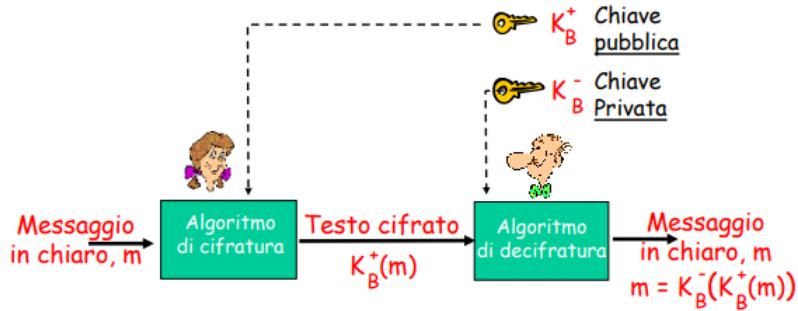


Figure 34: Crittografia asimmetrica

Si osservi che devono essere soddisfatti requisiti sulle chiavi:

1. $K_B^+(\cdot)$ e $K_B^-(\cdot)$ tale che:

$$K_B^-(K_B^+(m)) = m$$

2. Data la chiave pubblica K_B^+ , deve essere impossibile calcolare la chiave privata K_B^-

6.2.2 Algoritmi a chiave segreta

Soffermiamoci sui seguenti algoritmi a chiave segreta:

- **Cifrario di Cesare:**

Il suo funzionamento è molto semplice, si ha che ogni lettere del testo in chiaro è sostituita da quella che la segue di k posizioni nell'alfabeto. In questo caso dunque k è la chiave.

- **Sostituzione mono-alfabetica:**

Tale cifrario è una generalizzazione del cifrario di cesare, in cui ogni lettera del testo in chiaro è sostituita da un'altra, secondo uno schema libero. In particolare sostituendo tutte le occorrenze di una data lettere con un'altra scelta arbitrariamente.

Si ha quindi che il cifrario mono-alfabetico appare più efficiente di quello di Cesare, in quanto usando un alfabeto di 26 lettere, ci si trova a ben 26 fattoriali differenti possibilità di accoppiamento (un valore dell'ordine di 10^{26}). Tuttavia basandosi su informazioni statistiche è possibile rendere più agevole la violazione del cifrario (Partendo dalla frequenza delle lettere e delle parole in una lingua). Per esempio potrebbe essere effettuata un **Analisi delle frequenze** in tal modo:

1. Trovare un altro testo in chiaro nella stessa lingua dell'originale.

2. Calcolare la frequenza di ciascuna lettera
3. Ordinare le lettere in maniera decrescente in base alla frequenza
4. Fare lo stesso con il testo cifrato.
5. Provare a sostituire la lettera più frequente del testo cifrato con quella più frequente del testo in chiaro scelto, fino a quando la frase risultante non abbia un senso.

- **Sostituzione poli-alfabetica:**

Rappresenta un’ulteriore evoluzione delle precedenti forme di crittografia attraverso l’impiego di molteplici sostituzioni monoalfabetiche. In sostanza, le varie occorrenze di una stessa lettera vengono codificate in modo diverso a seconda della posizione in cui appaiono nel messaggio in chiaro, tramite l’utilizzo di uno schema predeterminato.

Esempio : C_1, C_2, C_1, C_2

La prima lettera del testo in chiaro utilizzerà il primo cifrario, la seconda il secondo cifrario, la terza lettera il primo, la quarta il secondo e così via dicendo (Lo schema si ripete in modo ciclico).

Più è lunga la chiave più è robusto l’algoritmo, sempre per ragioni statistiche.

- **Cifrari a trasposizione:**

Le lettere sono trasposte in altre posizioni, secondo una **chiave di trasposizione**. Ad esempio, la prima lettera è sostituita con la quarta, la seconda con la terza ecc. Tuttavia rimane attaccabile statisticamente.

- **One-Time Pad:**

Il testo chiaro viene codificato con una chiave delle medesima lunghezza. Risulta inattaccabile ma complesso da implementare (Chiavi molto lunghe, non efficiente).

6.2.3 Algoritmi moderni a chiave segreta

Gli algoritmi moderni, sempre a chiave segreta, rispetto a quelli precedentemente visti privilegiano l’utilizzo di chiavi piccole con l’applicazione di più metodologie in cascata (trasposizione, sostituzione, XOR). Troviamo dunque:

- **Algoritmo DES a chiave segreta:**

Data Encryption Standard, codificato e aggiornato dal U.S National Bureau of Standards, codifica il testo in chiaro in blocchi di 64 bit, utilizzando generalmente una chiave di 56 bit. In particolare vengono effettuate le seguenti operazioni (data una chiave di 56 bit):

- Permutazione iniziale
- 16 iterazioni intermedie identiche, ciascuna con 48 bit differenti come chiave.
- Permutazione finale.

(Dalle slide si legge che è sicuro ma, durante un concorso, DES fu ”bucato” in meno di 4 mesi).

- **Algoritmo AES a chiave segreta:**

Advanced Encryption Standard fu proposto nel 2001 come sostituto da NIST. AES a differenza di DES, codifica il testo in chiaro a blocchi di 128 bit, utilizzando chiavi generalmente di 128, 192 o 256 bit.

Si stima che se un calcolatore individuasse una chiave DES a 56 bit in 1 secondo, per violare una chiave AES a 128 bit ci impiegherebbe 149 miliardi di anni.

7 Lezione n7 01 04 2022

7.1 Algoritmi RSA e chiave pubblica

l'**algoritmo RSA**, acronimo derivato dal nome dei suoi autori: Rives, Shamir e Adleman, è diventato praticamente sinonimo di crittografia a chiave pubblica.

Rsa garantisce l'impossibilità di ricavare la chiave privata da quella pubblica, sfruttando le proprietà dei numeri primi e la difficoltà nel fattorizzare numeri molto grandi. Inoltre si presenta come un algoritmo molto lento rispetto ai precedenti a chiave segreta, e per tal motivo è applicato a piccole quantità di dati (Ad esempio per la trasmissione delle chiavi di una sessione DES).

Le chiavi possono essere lunghe 512,1024,2048 bit.

Due sono i punti principali su cui si basa RSA:

- **La scelta della chiave pubblica e di quella privata:**

1. Scegliere due numeri primi p e q : tanto più grande sarà il loro valore tanto più difficile risulterà violare RSA anche se, cifratura e decifratura richiederanno più tempo. Si consiglia di scegliere due numeri primi il cui prodotto è dell'ordine di 1024 bit).
2. Calcolare $n = pq$ $z = (p - 1)(q - 1)$
3. Scegliere e (encryption) minore di n , diverso da 1 e relativamente primo a z (ossia non devono avere divisori in comune).
4. Scegliere d (decryption) tale che $ed - 1$ sia divisibile (nessun resto) da z . In altri termini, dato e , d deve essere scelto in modo tale che:

$$ed \bmod z = 1$$

5. La chiave pubblica K_B^+ è la sequenza di bit concatenati della coppia n, e ; quella privata K_B^- è la coppia n, d .

- **Gli algoritmi di cifratura e di decifratura:**

1. Dati (n, e) e (n, d) calcolati come visto precedentemente.
2. Per la codifica di un messaggio m si determina m^e e poi il resto intero di m^e/n . Allora il messaggio cifrato c risulterà:

$$c = m^e \bmod n$$

3. Per decifrare il messaggio ricevuto si determina

$$m = c^d \bmod n$$

che richiede l'utilizzo della sua chiave segreta (n, d)

Si dimostra che: $m = \underbrace{(m^e \bmod n)^d}_{c} \bmod n$

- **IMPORTANISSIMA PROPRIETA':**

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

Se si usa prima la chiave pubblica, e poi quella privata o viceversa il risultato non cambia.

7.2 Confronto

- Gli Algoritmi a chiave segreta sono più veloci (di ordini di grandezza) rispetto agli algoritmi a chiave pubblica.
- Gli Algoritmi a chiave pubblica permettono l'eliminazione del problema della chiave segreta, le chiavi pubbliche sono invece diffuse e accessibili a tutti. Inoltre vi è l'introduzione della firma digitale e dei certificati digitali.
- Uso congiunto dei due tipi di algoritmo, generalmente infatti si usa un algoritmo a chiave pubblica per scambiare la chiave segreta di "sessione" con cui poi crittografare i successivi dati.
- Crittografia perfetta:

Def. Nessun testo codificato rilascia informazione alcuna né sulla chiave usata per la codifica, né sul testo in chiaro, il quale può essere recuperato se e solo se la chiave è disponibile.

Vi è probabilità **nulla** di ricavare informazioni supplementari da un testo codificato e perciò non vi è nessun tipo di attacco crittografico (crittoanalisi) possibile. Tuttavia rimane un approccio ideale poiché in pratica la crittografia quasi mai è perfetta

7.3 Integrità del messaggio

Rivolgiamo ora l'attenzione all'uso della crittografia per garantire l'**integrità dei messaggi**, tema noto anche come **autenticazione dei messaggi**.

Definiamo il problema dell'integrità dei messaggi usando Alice e Bob. Si supponga che Bob riceva un messaggio, cifrato o in chiaro, da Alice e voglia essere sicuro che:

- Il messaggio provenga effettivamente da Alice.
- Il messaggio non sia stato alterato lungo il cammino.

A tale scopo si utilizzano le **Funzioni hash crittografiche**

7.3.1 Funzioni hash crittografiche

Una **Funzione hash** prende in ingresso m e calcola una stringa di lunghezza fissata $H(m)$ detta hash. (Un esempio sono i checksum Internet e CRC che soddisfano tale proprietà). Tuttavia una **Funzione Hash crittografica** deve soddisfare un'ulteriore proprietà:

- Deve essere computazionalmente impossibile trovare due messaggi x e y diversi, tali che $H(x) = H(y)$.
- o anche: dato $m = H(x)$ con x sconosciuto è impossibile determinare x . (Ciò non accade con la checksum di Internet)

Vale a dire che, dal punto di vista computazionale, che data la coppia messaggio-hash $(m, H(m))$, creata dal trasmittente, un intruso non può falsificare il contenuto di un altro messaggio, y , che abbia lo stesso valore hash dell'originale.

Il checksum semplice, come quello di Internet, è un algoritmo che presenta alcune proprietà di una funzione hash:

- Crea sintesi di messaggi di lunghezza fissa (16bit)
- È multi-a-uno

Tutta via è poco efficace, difatti è relativamente semplice trovare altri dati con il medesimo checksum del messaggio originale. Si osservi l'esempio:

Messaggio	Rappresentazione				Checksum
	ASCII				
I O U 1	49	4F	55	31	
0 0 . 9	30	30	2E	39	
9 B O B	39	42	4F	42	
	B2	C1	D2	AC	

Messaggio	Rappresentazione				Checksum
	ASCII				
I O U 9	49	4F	55	39	
0 0 . 1	30	30	2E	31	
9 B O B	39	42	4F	42	
	B2	C1	D2	AC	

Figure 35: Esempio

In Figura 35 i messaggi sono diversi ma la checksum è identica, di conseguenza viola i requisiti di autenticazione precedentemente indicati. Quindi risulta ovvio che i requisiti di sicurezza richiedono l'impiego di una funzione hash più efficiente.

7.3.2 MD5 e SHA1

Un algoritmo molto utilizzato per l'hash dei messaggi è MD5, ideato da Ron Rivest (RFC 1321), in grado di calcolare una hash di 128 bit con un processo a 4 fasi (il prof non le ha spiegate io le riporto per completezza perché mi piace perdere tempo):

- Si inizia con la normalizzazione attraverso l'aggiunta del valore 1 seguito da una serie di 0, in un numero tale da soddisfare determinate condizioni.
- Si effettua l'aggiunta in coda di una rappresentazione a 64 bit della lunghezza del messaggio originale.
- Viene inizializzato un accumulatore
- In questa fase i blocchi composti da 16 gruppi di 32 bit (word) del messaggio vengono "triturati" attraverso un processo che prevede quattro cicli di elaborazione.
- Se volete perdere altro tempo, la descrizione dettagliata è in RFC 1321.

Ritornando a noi, con una stringa x di 128 bit arbitrari, appare difficile costruire un messaggio m il cui hash MD5 sia uguale a x . Tuttavia recentemente (2005)⁹ sono stati condotti attachhi contro MD5.

Un altro importante algoritmo hash, attualmente in uso, è **Hash sicuro** (SHA-1, secure hash algorithm). Tale algoritmo è uno standard utilizzato nelle applicazioni federali USA e produce una sintesi del messaggio di 150 bit. La maggior lunghezza del risultato rende SHA-1 più sicuro.

7.3.3 Codice di autenticazione dei messaggi (MAC)

Ritornando all'integrità dei messaggi, per garantire quest'utlme, oltre alla funzioni hash crittografiche, si ha bisogno di un segreto condiviso, ossia una stringa di bit chiamata formalmente **chiave di autenticazione**. Usando il segreto condiviso, l'integrità del messaggio è realizzata come segue:

1. Utente 1 crea un messaggio m , concatena s con m ottenendo così $m + s$, calcola la stringa hash $H(m + s)$ con algoritmo visto precedentemente (SHA-1). $H(M + S)$ è chiamato **codice di autenticazione del messaggio** (MAC, message authentication code).
2. Utente 1 aggiunge il *MAC* al messaggio m , creando il messaggio esteso $(m, H(m + s))$ e lo manda all'utente 2.
3. Utente 2 riceve il messaggio esteso, e conoscendo s , calcola il *MAC*. Se il *MAC* calcolato coincide con il *MAC* ricevuto, conclude che va tutto bene.

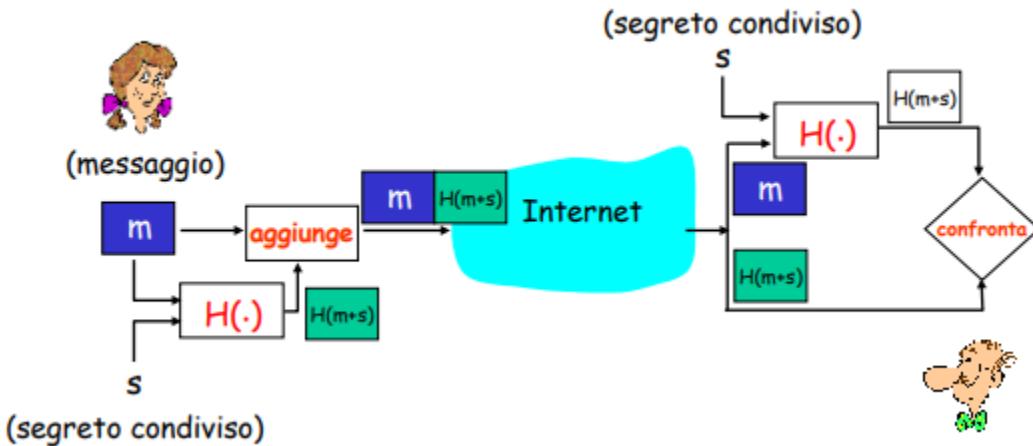


Figure 36: Esempio

7.4 Firme Digitali

Nel mondo digitale per indicare il titolare o il creatore di un documento, o dichiarare di essere d'accordo con il suo contenuto, si ricorre alla **firma digitale**, una tecnica crittografica analoga alla firma tradizionale scritta che consente di raggiungere svariati obiettivi.

La firma digitale, come la firma tradizionale deve essere verificabile e non falsificabile. Ossia deve consentire di dimostrare che un certo documento sia davvero stato firmato proprio da quella data persona (verificabile) e che solo lei poteva realizzarlo (non falsificabile). Inoltre deve valere anche il non ripudio. Cioè non è possibile rinnegare di essere il proprietario di un documento firmato digitalmente.

Il professore non vuole la versione naive sulle slide all'interrogazione quindi riporto solo la versione efficiente. Comunque in generale quella è troppo eccessiva poiché utilizzo della crittografia a chiave pubblica per le firme digitali presenta il problema per cui cifratura e decifratura dei dati

sono onerose dal punto di vista computazionale. Allora invece di firmare l'intero messaggio, potrebbe limitarsi a firmare solo l'hash così lo sforzo computazionale risulterebbe sostanzialmente diminuito.

Il processo di firma digitale si presenta in tal modo:

1. Utente 1 applica una funzione hash all'intero messaggio e ottiene una hash, che cifra con la propria chiave privata. Successivamente il messaggio intero originale (testo in chiaro) concatenata a la sua sintesi firmata digitalmente (firma digitale) sono quindi inviati all'Utente 2.
2. Utente 2 applica la chiave pubblica del trasmittente al messaggio per ottenerne una hash e poi impiegare la funzione hash al messaggio con il testo in chiaro, in modo da procurarsi la seconda hash. Se i due risultati coincidono allora Utente 2 può stare tranquillo sull'integrità del messaggio e sull'identità del suo autore.

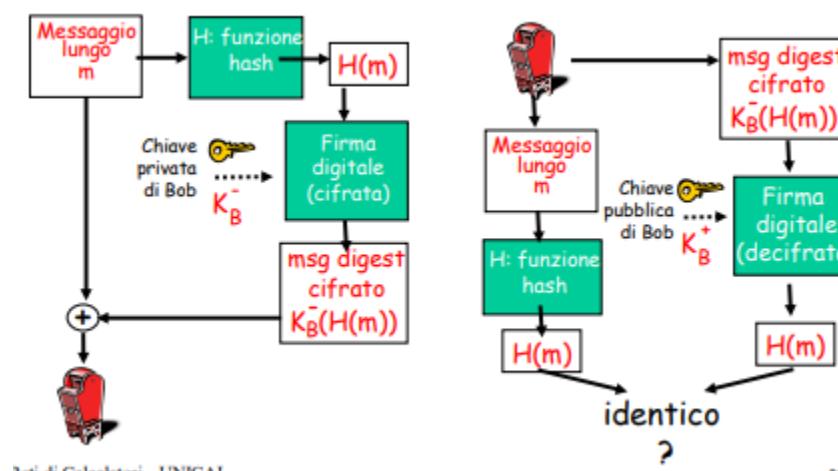


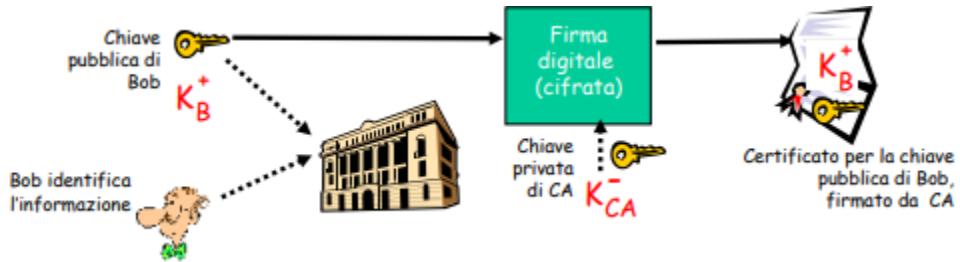
Figure 37: Firma digitale versione efficiente

Non si confonda firma digitale con mac, poiché la prima è una tecnica più gravosa in quanto richiede l'infrastruttura della chiave pubblica.

7.4.1 Certificazione della chiave pubblica

Un'importante applicazione della firma digitale è la certificazione della chiave pubblica, cioè la certificazione che una chiave pubblica appartenga a una specifica entità (Importantissima per il caso precedentemente esposto, così che l'utente 2 possa stare sicuro che la chiave pubblica con cui sta operando appartenga esattamente all'utente 1). Generalmente, la relazione tra una data chiave pubblica e una determinata entità è stabilita da un'autorità di certificazione (CA, certification authority), il quale ha i seguenti compiti:

1. Verifica che un'entità sia veramente chi afferma di essere. Difatti quando si stabilisce un accordo con CA si deve fornire una "prova d'identità".
2. Una volta verificata l'identità, la CA rilascia il certificato che autentica la corrispondenza tra chiave pubblica ed entità.
3. Il certificato contiene la chiave pubblica e le informazioni d'identificazione globali e uniche del suo proprietario ed è firmato digitalmente dalla CA.



Successivamente a ciò quando un Utente 2 viola la chiave pubblica di un Utente 1:

- Utente 2 prende il certificato di Utente 1.
- Utente 2 applica la chiave pubblica di CA al certificato pubblico di Utente 1 e ottiene la chiave pubblica di Utente 1.

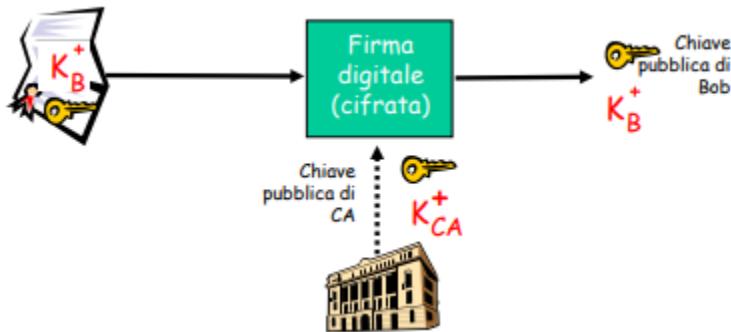


Figure 38: Esempio

Tra i più significativi campi dei certificati si ha:

- Numero di serie
- Informazioni sul titolare, compreso l'algoritmo e il valore della chiave
- Informazioni su chi ha emesso il certificato
- Data di validità
- Firma digitale di chi lo ha emesso.

8 Lezione n8 08 04 2022

8.1 Autenticazione end-to-end

Con autenticazione end-to-end si intende il processo attraverso il quale una entità prova la sua identità a un'altra entità su una rete di calcolatori, come nel caso di un utente che prova la sua identità a un server e-mail.

È importante tenere a mente la sostanziale differenza di quest'ultima dalla problematica della firma digitale ossia dimostrare che un messaggio ricevuto è stato veramente inviato da un dato mittente.

Nell'autenticazione in rete, devono necessariamente basarsi unicamente sullo scambio di messaggi o di dati come parte di un protocollo di autenticazione, che interviene prima che le due parti eseguano qualsiasi altro protocollo.

Il professore prima di entrare nello specifico, ha introdotto l'argomento presentando dei protocolli di autenticazione **ap** inventati, elencando pregi e difetti di ciascuna.

- **Protocollo di autenticazione ap1.0:**

Protocollo di autenticazione molto semplice, in cui un Utente1 invia un messaggio a un Utente2 di essere effettivamente Utente1. Ovviamente tale approccio non permette all'Utente2 di essere sicuro che chi ha inviato il messaggio sia effettivamente chi dice di essere.

- **Protocollo di autenticazione ap2.0:**

Se l'Utente1 avesse un indirizzo di rete conosciuto (Indirizzo IP) che utilizza abitualmente, Utente2 potrebbe autenticarlo verificando la corrispondenza tra indirizzo sorgente sul datagrammaIP che trasporta il messaggio e quello di Utente1. Questo dissuaderebbe un intruso molto ingenuo, ma non fermerebbe sicuramente un Utente3 (l'intruso) in grado di elaborare un datagramma IP contenente un indirizzo sorgente artefatto (che imita quello di Utente1) e poi inviarlo attraverso il protocollo a livello di collegamento al primo router.

- **Protocollo di autenticazione ap3.0:**

Un classico approccio all'autenticazione è l'utilizzo di una password, un segreto condiviso dall'entità che svolge il processo di riconoscimento e da quella che deve essere autenticata. Dato che l'impiego di password è ampiamente diffuso, potremmo aspettarci che il protocollo *ap3.0* sia piuttosto sicuro. E invece no, perché se l'Utente3 (l'intruso) intercetta la comunicazione (Con un semplice attacco di replica) dell'Utente1, può scoprirla la password che viene inviata in chiaro.

- **Protocollo di autenticazione ap3.1** Per superare i limiti di *ap3.0* potremmo cifrare la password in modo da impedire all'intruso di appropriarsene. In particolare l'Utente1 condivide una chiave simmetrica segreta K_{A-B} con l'Utente2, e utilizza quest'ultima per cifrare la password segreta. L'Utente2 provvederebbe poi a decodificarla e a controllarne la veridicità. In caso di risponso positivo l'Utente2 si sente sicuro in quanto il mittente, oltre la password conosce anche la chiave segreta condivisa con cui è stata cifrata.

Tuttavia l'Utente3 pur non conoscendo la password, può utilizzare ancora una volta l'**attacco di replica**, ossia inserirsi nella comunicazione, registrare la chiave cifrata e successivamente riprodurla per inviarla all'Utente2 sostenendo di essere l'Utente1.

In sostanza, l'uso della password cifrata non ha reso la situazione differente dalla precedente.

- **Protocollo di autenticazione ap4.0** Il fallimento dello scenario precedente deriva dal fatto che l'Utente2 non riesce a distinguere fra l'autenticazione originale di Utente1 e la sua successiva riproduzione. Uno studente attento (non tu chiaramente e neanche io) sa sicuramente (ma manco per il cazzo aggiungerei) che il protocollo dell'handshake a tre vie di TCP potrebbe risolvere il problema. Adottiamo quindi la medesima idea introducendo un **nonce** un numero che il protocollo userà soltanto una volta nel seguente modo:

1. Utente1 invia il messaggio "Sono Utente1" a Utente2.
2. Utente2 sceglie un *nonce* R e lo trasmette.
3. Utente1 utilizza K_{A-B} , la chiave simmetrica segreta che condivide con Utente2 per codificare il nonce e gli re-invia il valore risultante. $K_{A-B}(R)$. Il punto cruciale è che l'Utente conosca K_{A-B} .
4. Utente2 decifra il messaggio ricevuto: se il **nonce** è quello da lui inviato, Alice è autenticata.

Attraverso l'utilizzo del nonce, R, e con il controllo del valore di ritorno $K_{A-B}(R)$, Bob può essere sicuro che si tratta veramente di Alice (perché conosce la chiave segreta necessaria per cifrare R) e che si trova in quel momento all'altro capo del collegamento (in quanto ha codificato il *nonce* R , che Bob ha appena creato).

- **Protocollo di autenticazione ap5.0** Il protocollo ap4.0 risolve il problema tramite l'impiego di nonce e la crittografia a chiave simmetrica. Tuttavia il medesimo risultato può essere ottenuto utilizzando il nonce e la crittografia asimmetrica, ovviando però il problema del primo scambio della chiave segreta condivisa.

8.1.1 Autenticazione con uso di un KDC

In un sistema distribuito composto da N host, ogni host condivide $N - 1$ chiavi e globalmente sono necessarie $N(N - 1)/2$ chiavi segrete. Ciò ci mette difronte al problema di come distribuire le chiavi e non solo, difatti anche la complessità è un parametro da non trascurare.

Tuttavia per risolvere tali problematiche può essere utilizzato un approccio centralizzato, ossia l'utilizzo di un **Key Distribution Center** (KDC). Il **KDC** è un server che condivide un'unica chiave simmetrica segreta con ciascun utente registrato.

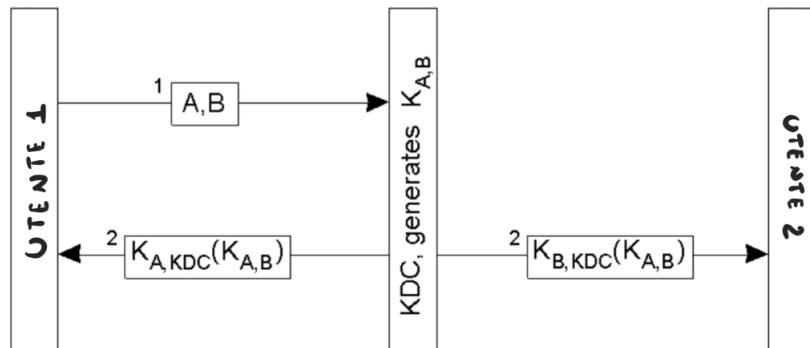


Figure 39: Principio di uso di un KDC

Per Utente1 e Utente2, indichiamo come chiavi K_{A-KDC} e K_{B-KDC}

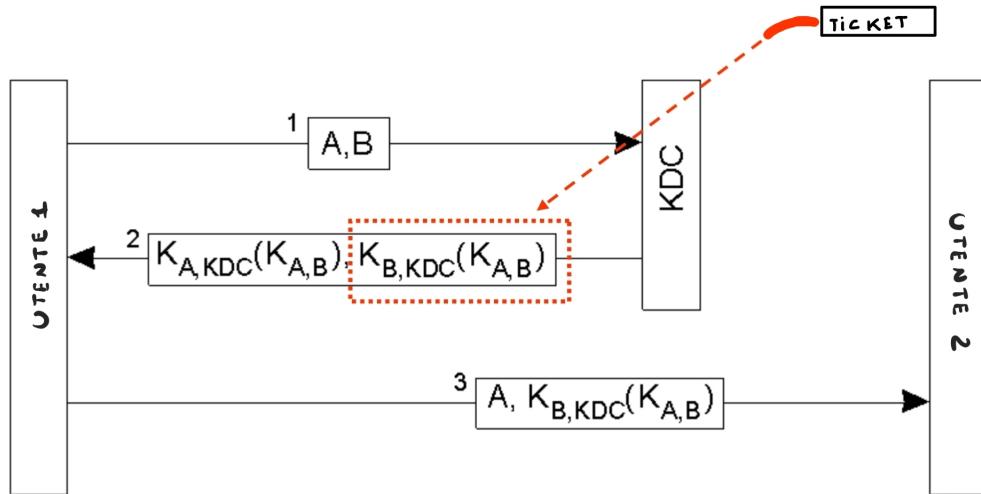


Figure 40: Uso di un ticket per permettere ad Utente1 di connettersi ad Utente2

Nell'esempio proposto Il KDC trasmette $K_{B-KDC}K(A-B)$ ad *Utente1* ed affida a quest'ultimo il compito di connettersi a *Utente2*. Il messaggio è chiamato *Ticket*.

8.2 Rendere sicura la posta elettronica

Abbiamo affrontato tutte le questioni fondamentali relative alla sicurezza di rete, compresa la crittografia simmetrica e a chiave pubblica, l'autenticazione, la distribuzione delle chiavi, l'integrità dei messaggi e la firma digitale. Esaminiamo ora come questi meccanismi possono essere utilizzati per fornire la sicurezza sulla posta elettronica.

Si osservi che le funzionalità di sicurezza deve essere implementata non solo a livello Applicazione (Come nel caso che vedremo) ma anche nei livelli bassi della pila (Trasporto e Rete).

8.2.1 E-mail sicure

Realizziamo dunque l'esposizione incrementale di un progetto ad alto livello per la sicurezza della posta elettronica.

Ipotizziamo che un Utente1 voglia inviare un e-mail a un Utente2 e che un Utente3 voglia intrrompersi. Definiamo le caratteristiche di sicurezza che il progetto deve soddisfare:

- **Riservatezza:** L'Utente3 non deve poter leggere le email tra Utente1 e Utente2
 - **Autenticazione del mittente:** L'Utente2 vuole essere sicuro che sia proprio L'Utente1 l'interlocutore con cui sta scambiando messaggi.
 - **Integrità:** Ovvero la certezza che i messaggi non siano stati alterati durante il trasporto.
 - **Autenticazione del ricevente:** L'Utente1 deve essere certo che i messaggi siano stati inviati esattamente all'Utente2.

Partiamo dalla prima caratteristica la **Riservatezza**. Il modo più semplice per ottenerla è cifrare il messaggio con una chiave simmetrica (DES o AES), tuttavia si pone il problema della distribuzione in modo che solo mittente e destinatario ne abbiano una copia. Un approccio alternativo consiste nell'utilizzare RSA e assumendo che l'Utente1 riceva esattamente la chiave pubblica

dell'Utente2 si ottiene un ottimo strumento di segretezza. Purtroppo però tale approccio è relativamente inefficiente, soprattutto per messaggi troppo lunghi.

Per risolvere il problema dell'efficienza utilizziamo una chiave di sessione, si ottiene dunque:

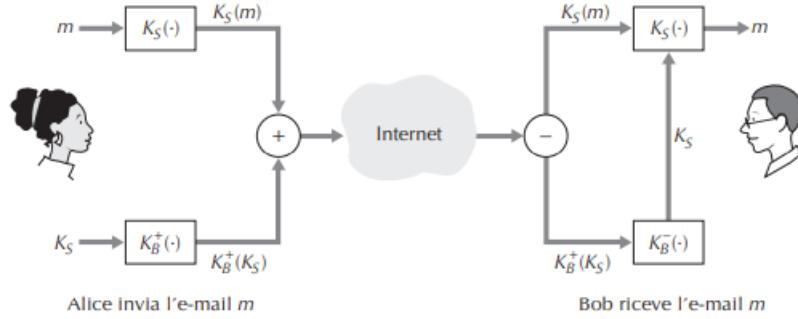


Figure 41: Caso d'uso

- **Utente1 (Alice):**

1. Sceglie arbitrariamente una chiave simmetrica K_s
2. Codifica il messaggio m con K_s
3. Cifra la chiave simmetrica con la chiave pubblica dell'Utente2 $K_B^+(K_s)$.
4. Invia $K_s(m)$ e $K_B^+(K_s)$ all'Utente2.

- **Utente2 (Bob):**

1. Utilizza la sua chiave privata per ottenere la chiave simmetrica K_s
2. Utilizza K_s per decodificare $K_s(m)$ e ottiene m .

Supponiamo ora che la principale preoccupazione dei due utenti siano **L'autenticazione del mittente** e **L'integrità dei messaggi**, e nessun interesse verso la sicurezza. Per raggiungere tale obiettivo si utilizzano le firme digitali e una funzione di hash. In particolare:

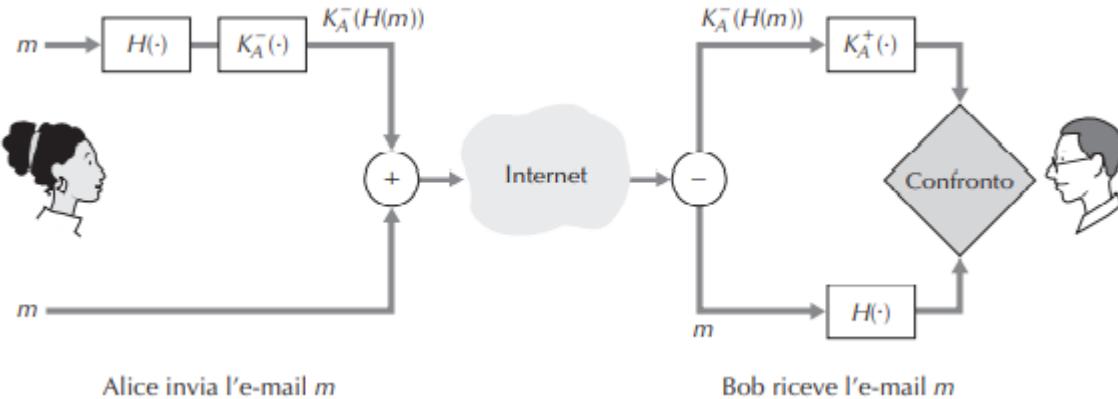


Figure 42: Caso d'uso

- **Utente1:**

1. Applica una funzione hash H (Come MD5) al proprio messaggio m .
2. Cifra il risultato con la sua chiave privata K_A^- per creare la firma digitale.
3. Concatena messaggio in chiaro e firma
4. Invia all'Utente2 il risultato

- **Utente2:**

1. Applica la chiave pubblica dell'Utente2 K_A^+ , all'hash del messaggio che è stata firmata.
2. Confronta il risultato con quanto ottenuto con la sua funzione di hash H .
3. Se coincidono allora si è sicuri che il messaggio è dell'Utente1 e che non è stato alterato durante il trasferimento.

Affrontiamo ora il progetto di un sistema di posta elettronica che risponda alle richieste di segretezza, autenticazione del mittente e integrità dei messaggi.

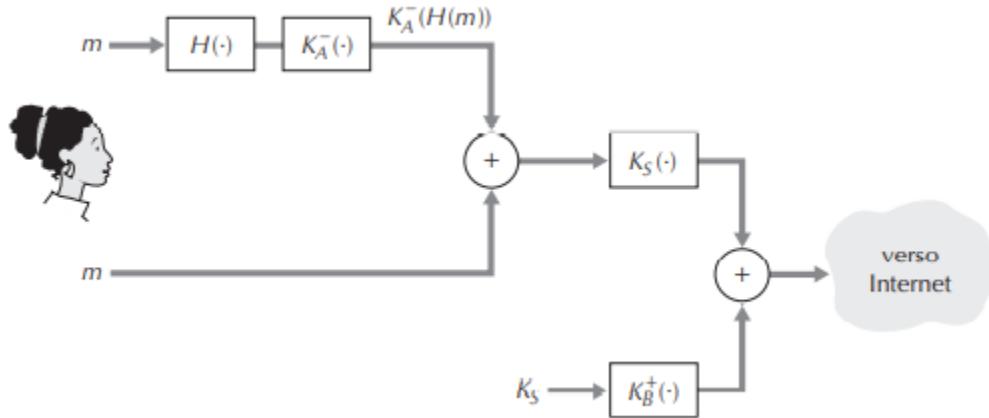


Figure 43: Caso d'uso che risponde a tutte le richieste di segretezza

- Utente1: Confeziona un "pacco" preliminare che contiene il messaggio originale m e hash del messaggio firmato digitalmente $K_A^-(H(m))$. Successivamente cifra il "pacco" con la propria chiave privata e poi con la chiave pubblica dell'Utente2.
- Utente2: Riceve il "pacco" applica due volta la tecnica a chiave pubblica, ossia con la sua chiave privata e con la chiave pubblica dell'Utente1.

Tale schema raggiunge un grado di sicurezza soddisfacente. Tuttavia rimane ancora un problema la distribuzione delle chiavi pubbliche tra i due Utenti. Difatti l'Utente3 potrebbe spacciarsi per l'Utente2 e inviare all'Utente1 la propria chiave pubblica.

8.3 PGP

Pretty Good Privacy, ideato da Phil Zimmerman (indagato per tre anni dai servizi federali), è uno schema di cifratura per la posta elettronica diventato uno standard. Utilizza chiavi simmetriche di crittografia, chiavi pubbliche, funzioni hash e firme digitali. Inoltre il software assicura sicurezza, integrità del messaggio e autenticazione del mittente.

```

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Possiamo vederci stasera?
Appassionatamente tua, Alice
----- BEGIN PGP SIGNATURE -----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yHJRHHGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----

```

Figure 44: Messaggio PGP firmato

8.4 Posta elettronica certificata (PEC)

La posta elettronica standard, pur essendo uno strumento facilmente accessibile da qualsiasi utilizzatore, non fornisce né garanzie certe di effettivo invio e consegna di un messaggio, né certezza sulla sua paternità, né regole standardizzate che consentano di tracciare il percorso del messaggio durante la fase di trasmissione. Invece la **PEC** è un sistema di posta elettronica in grado di superare queste "debolezze" e di poter essere utilizzata in qualsiasi contesto nel quale sia necessario avere prova opponibile dell'invio e della consegna di un documento elettronico.

La posta elettronica sta alla lettera ordinaria come la PEC sta alla raccomandata

Tra le caratteristiche principali si ha:

- Conoscenza certa della casella mittente e del suo titolare.
- Possibilità di legare in maniera certa la trasmissione con il documento trasmesso.

Inoltre la posta elettronica certificata è adatta anche allo scambio d'informazioni tra applicazioni. In particolare un'organizzazione può ricevere attraverso messaggi di posta elettronica certificata documenti che possono essere immediatamente tratti da un'applicazione informatica.

8.4.1 PEC: Componenti

- **Utente mittente:**

Cioè chi ha l'esigenza di inviare un documento informatico

- **Utente Destinatario:**

Il soggetto al quale sarà destinato l'oggetto dell'invio

- **Gestore del mittente:**

Il soggetto con il quale il mittente mantiene un rapporto finalizzato alla disponibilità del servizio di PEC

- **Gestore del destinatario:**

Il soggetto con il quale il destinatario mantiene un rapporto finalizzato alla disponibilità del servizio di PEC

- **Rete di comunicazione:**

Tipicamente può essere considerata Internet.

- **Documento informatico:**

Realizzato dal mittente e oggetto dell'invio verso il destinatario.

8.4.2 PEC: Processo

Il punto di partenza del processo coinvolge il mittente che con i propri strumenti predisponde uno o più documenti informatici. Successivamente predisposto l'oggetto dell'invio, il mittente si deve far riconoscere dal sistema di PEC del proprio gestore secondo le modalità da questi previste.

- Una modalità diffusa per accedere al servizio PEC è ad esempio la classica coppia user-id/password.
- Possibilità di adottare modalità diverse e con maggiori livelli di sicurezza quali, ad esempio le smart card.

Superata la fase di riconoscimento, il mittente, utilizzando l'interfaccia disponibile, che verosimilmente sarà il classico client di posta elettronica o in alternativa, un Web browser, predisponde il messaggio di PEC e lo invia.

Si osservi che il mittente opererà secondo le abituali modalità previste per l'invio di un messaggio di posta elettronica convenzionale.

A seguito dell'invio, il sistema di PEC del mittente effettua una serie di controlli finalizzati a verificare la correttezza formale del messaggio e l'assenza di virus. Si hanno due scenari possibili:

- a Nel caso in cui i controlli evidenziassero criticità, il messaggio non verrebbe inoltrato verso il destinatario e il mittente riceverebbe una ricevuta, firmata elettronicamente dal proprio gestore, contenente l'informazione che l'invio non ha avuto luogo e le relative motivazioni.
- b Qualora i controlli, realizzati in fase d'invio, non rilevino criticità, il gestore mittente provvede a inserire, come allegato, il messaggio preparato dal mittente e a firmarlo digitalmente. Quest'ultima operazione è finalizzata a garantire l'inalterabilità del messaggio che il mittente ha predisposto per l'invio.

A questo punto il gestore mittente provvede ad inoltrare, tramite la rete, il messaggio verso il gestore destinatario. Quest'ultimo, ricevendo ciò che è stato inoltrato dal Gestore mittente, provvede ad effettuare una serie di verifiche, destinate a controllare la provenienza (da un gestore PEC iscritto nell'apposito elenco) e l'integrità del messaggio ricevuto.

Questi ultimi controlli sono finalizzati ad avere tutte le garanzie in merito alla non alterazioni del messaggio nel suo transito tra un gestore e un altro.

Fra i controlli effettuati, anche in questo caso si rileva la presenza di virus che bloccerebbero l'inoltro del messaggio verso il destinatario. In particolare il verificarsi di questa situazione comporta una notifica, al mittente, di mancata consegna del messaggio inviato per problemi di sicurezza.

Giunti a tale punto il destinatario ha disponibile nella sua casella il messaggio ricevuto, e può quindi consultarlo come in un sistema di posta elettronica tradizionale. Il sistema di PEC, essendo un sistema di trasporto, non considera la lettura del messaggio come prova dell'effettiva consegna. Il messaggio si intende consegnato quando il gestore lo rende disponibile nella casella di posta elettronica certificata del destinatario.

Nelle due situazioni d'invio e ricezione, laddove il gestore rilevi la presenza di virus nel messaggio, non deve trasmetterlo e deve mantenere il messaggio in un apposito archivio per una durata di trenta mesi, così come previsto dalle norme, alla fine di poter effettuare successive verifiche circa l'evento certificato.

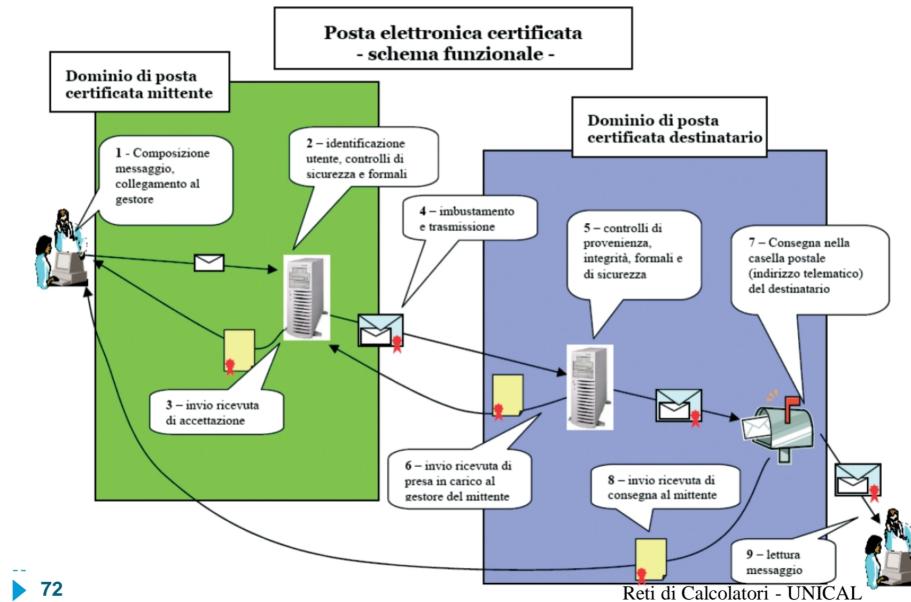


Figure 45: Posta elettronica certificata - schema funzionale

8.5 Cloud computing

Il **Cloud Computing** è un modello di elaborazione in cui le risorse **dinamicamente scalabili e virtualizzate** sono fornite come **servizio** tramite **Internet** ("Il cloud").

Gli utenti non devono avere conoscenza, esperienza o controllo dell'infrastruttura tecnologica nel "Cloud" che li supporta e agisce dietro le quinte.

Se usi un servizio online per inviare posta elettronica, modificare documenti, guardare film è probabile che tutto questo sia possibile grazie al cloud computing.

- **Definizione del NIST** (National Institute of Standards Technology):
Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.

8.5.1 Caratteristiche essenziali

1. Scalabile ed elastico:

L'utente finale ha l'impressione di avere accesso ad una quantità di risorse potenzialmente illimitate. Il fornitore di un servizio basato su cloud non deve occuparsi del dimensionamento dell'infrastruttura la quale si adatta dinamicamente (Aumentando o diminuendo la sua consistenza) al carico di lavoro corrente.

2. Service-based:

Gli utenti possono disporre di applicazioni, tempo di CPU, archiviazione di dati o processi complessi, tutti erogati sotto forma di servizio.

3. Internet-based:

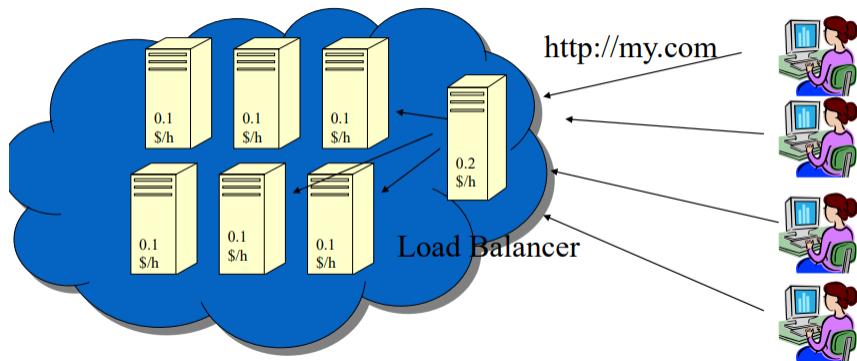
Internet viene utilizzato come mezzo di erogazione dei servizi offerti su Cloud, attraverso Web Service con interfacce standardizzate.

4. Costi a consumo:

Nel paradigma del cloud, è previsto il pagamento per la quantità di servizio erogato. L'utente percepisce l'accesso ad una potenza di elaborazione illimitata, ma paga soltanto ciò che ha effettivamente utilizzato.

5. Risorse condivise:

I servizi cloud sono multi-tenant, ovvero sono installati in un ambiente di lavoro condiviso da più entità, sia a livello hardware, che a livello software. Ciò può significare che una singola istanza di un software, e la piattaforma hardware dove viene eseguita, serve più client.



9 Lezione n9 21 04 2022

9.1 Ancora sul Cloud Computing

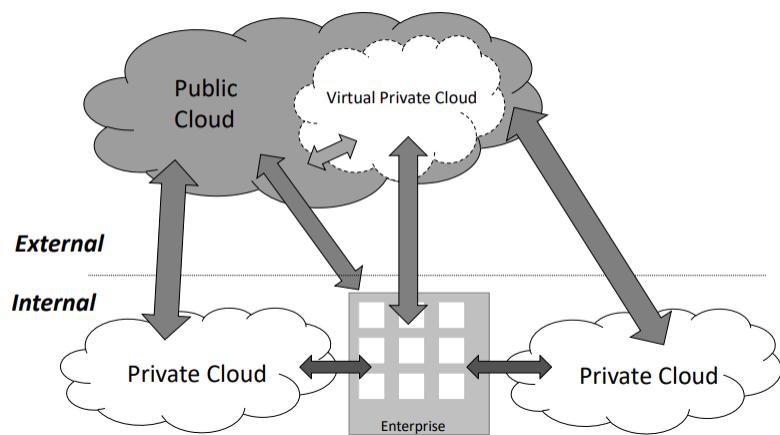
9.1.1 Motivazioni economiche

Devo riascoltare sta parte

9.1.2 Modelli di deployment

Non tutti i cloud sono uguali e non sempre lo stesso tipo di cloud computing è adatto a tutte le esigenze. Sono disponibili numerosi modelli, tipi e servizi diversi per offrire la soluzione più conforme alle proprie esigenze. Per prima cosa si deve determinare il tipo di distribuzione di cloud, ossia l'architettura in cui verranno implementati i servizi. Vi sono tre modalità diverse:

- **Public Cloud:** Il cloud pubblico è reso disponibile su Internet per utenti singoli o aziende, ed è tipicamente gestito da un provider di terze parti (Amazon, Google, Microsoft...) I servizi dei cloud pubblici sono offerti attraverso Internet, ed esposti in maniera standardizzata, auto-configure e predisposti per il pagamento a consumo.
- **Private Cloud:** Un cloud privato si riferisce alle risorse di cloud computing usate esclusivamente da una singola azienda o organizzazione. Un cloud privato può trovarsi fisicamente nel data center locale della società o le stesse società pagano provider di servizi di terze parti per ospitare il proprio cloud privato. Al di là del substrato tecnologico, un'importante differenza fra i grandi data center delle aziende e i cloud privati risiede nel modello di gestione.
- **Hybrid Cloud:** Il cloud ibrido è un'infrastruttura ottenuta dalla combinazione di cloud pubblici e privati, i quali rimangono entità distinte, ma collegate e amalgamate da tecnologie che garantiscono la condivisione di applicazioni e dati. Grazie alla possibilità di spostare dati e applicazioni tra cloud pubblici e privati, un cloud ibrido offre all'azienda maggiore flessibilità e più opzioni di distribuzione e aiuta a ottimizzare l'infrastruttura esistente, la sicurezza e la conformità. Inoltre combinando elementi di cloud pubblico e privato, inclusa qualsiasi combinazione di provider e consumer, può contenere più livelli di servizio.



Si osservi che la Virtual Private Cloud è un modello tramite la quale l'azienda ha la possibilità di limitare un determinato quantitativo di risorse a uno specifico cliente. Simile al private cloud ma le risorse risiedono sempre dal fornitore.

9.1.3 Modelli di servizio

Il cloud prevede tre modelli di servizio, organizzati in una struttura piramidale a tre livelli. Tale piramide è detta di Sheehan, dal nome del suo ideatore. La conoscenza di queste soluzioni e delle loro differenze semplifica il raggiungimento degli obiettivi aziendali.

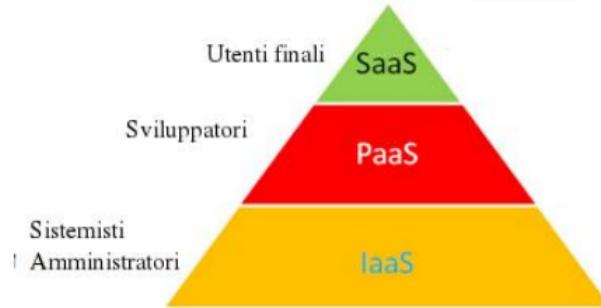


Figure 46: Piramide di Sheehan

- **Software as a Service, SaaS:**

In questo livello il software è presentato agli utenti finali come un servizio on-demand tipicamente come un servizio disponibile sul web.

L'utente che usufruisce di tale servizio non si deve curare di problemi quali installazione e manutenzione del software, ma ottiene in maniera semplice e intuitiva il servizio stesso, il quale viene gestito in maniera condivisa nel cloud, con un meccanismo di scaling che permette l'aggiunta immediata di funzionalità addizionali all'occorrenza (on-demand).

Esempi sono le Google Apps Wordpress.com sebbene ormai la quasi totalità dei servizi offerti in rete ricada in questa definizione.



Figure 47: SaaS

- **Platform as a Service, PaaS:**

Fornisce una piattaforma di sviluppo software con una serie di servizi e funzionalità che supportano e assistono le applicazioni realizzate nel cloud durante tutto il ciclo di vita dello sviluppo.

Le fasi di progettazione, sviluppo, testing, deployment, monitoraggio e hosting sono tutte effettuate nel cloud, ottenendo così un ambiente di sviluppo accessibile e utilizzabile on-

line, anche in maniera collaborativa per team distribuiti geograficamente in parti diverse. L'applicazione che viene eseguita nel cloud, è generalmente in grado di scalare automaticamente in base alla mole di utenti che interagiscono con essa.

Alcuni esempi sono Heroku, Google App Engine, Openshift e Joyent.

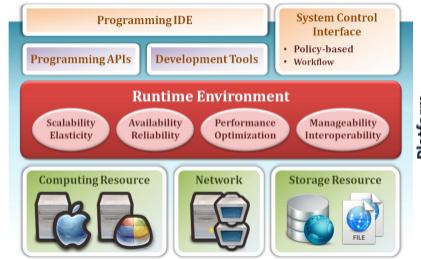


Figure 48: PaaS

- **Infrastructure as a service, IaaS:**

Questo livello si fa carico della virtualizzazione della potenza di calcolo, dello storage e della connettività di rete di più data center, offrendo all'utente la possibilità di gestire un'infrastruttura di elaborazione completa in rete.

È noto anche come HaaS (Hardware as a Service), ed è il tipo di cloud più potente e più flessibile poiché permette a sviluppatori e amministratori di sistema la massima libertà di scelta dei componenti hardware (Virtualizzati) e di scaling di risorse, il tutto fornito sempre come un servizio offerto nel cloud, pertanto on-demand.

Esempi sono Amazon Elastic Compute Cloud (EC2), OpenStack, VMware vCloud Hybrid Service e Microsoft Azure.

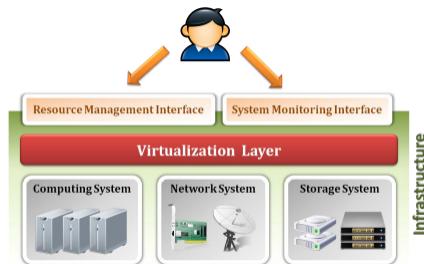


Figure 49: IaaS

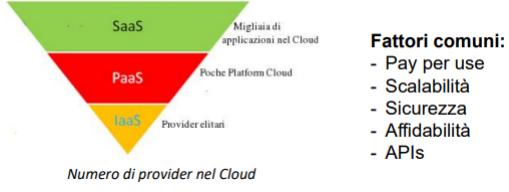


Figure 50: Piramide rovesciata per enfatizzare l'utilizzo in termini di utenti

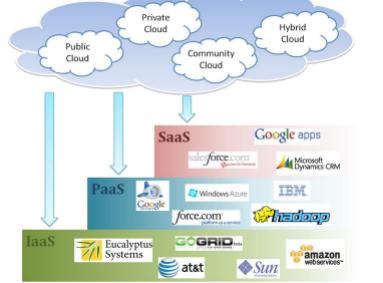


Figure 51: Cloud Ecosystem

9.1.4 Ruoli

Il sistema del cloud prevede tipicamente tre entità:

- **Fornitore di servizi** (cloud provider): Offre servizi (Ad es. server virtuali, storage, applicazioni complete) generalmente secondo un modello "pay per-use";
- **Cliente amministratore** Sceglie e configura i servizi offerti dal fornitore, generalmente offrendo un valore aggiunto come ad esempio applicazioni software;
- **Cliente finale**: Utilizza i servizi opportunamente configurati dal cliente amministratore.

In determinati casi , il cliente amministratore e il cliente finale possono coincidere. Ad esempio, un cliente potrebbe utilizzare un servizio di storage per effettuare il backup dei propri dati: in questo caso il cliente finale provvede a configurare e utilizzare il servizio.

9.1.5 Accesso ai servizi

Gli utenti accedono ai servizi di cloud computing utilizzando dispositivi client, come computer desktop, laptop tablet e smartphone. Attraverso questi dispositivi, gli utenti accedono e interagiscono con i **servizi cloud-based** utilizzando un browser Web o un'app desktop/mobile. Il software aziendale e i dati dell'utente vengono eseguiti e archiviati su server ospitati in **data center** cloud che forniscono risorse di **storage** e di **elaborazione**. Le risorse includono migliaia di server e dispositivi di archiviazione collegati tra loro attraverso una rete **intra-cloud**. Il trasferimento di dati tra data center e utenti avviene su una rete geografica.

9.1.6 Architetture

L'architettura informatica del cloud computing prevede uno o più **server** reali (**cluster di server**), generalmente in architettura ad alta affidabilità e fisicamente collocati presso uno o più **data center** del fornitore del servizio.

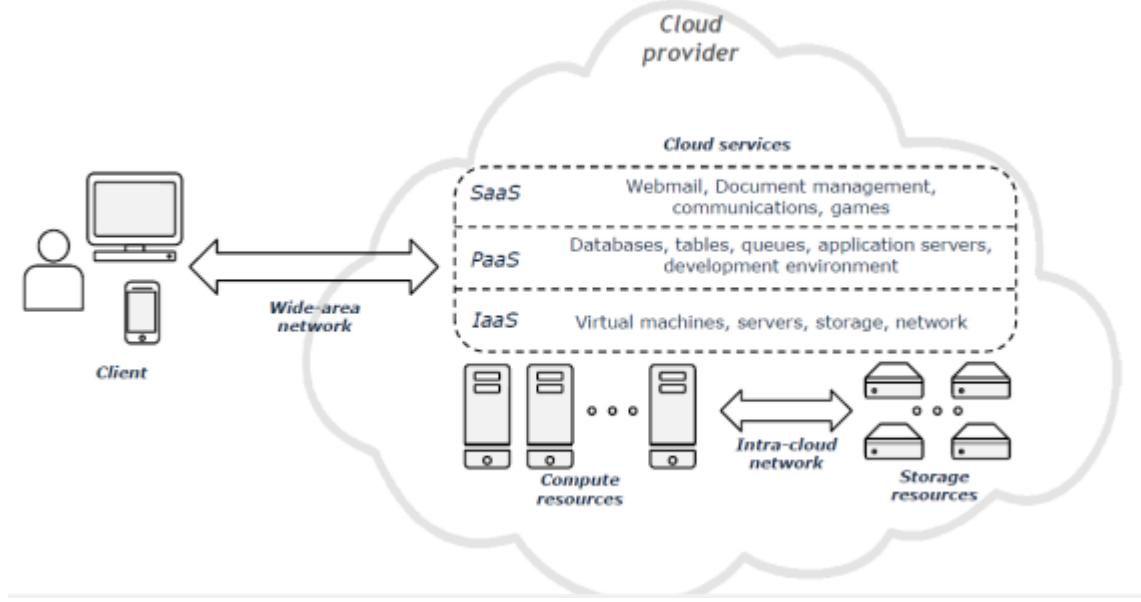


Figure 52: Esempio

Il fornitore del servizio espone delle **interfacce** per elencare e gestire i propri servizi (Ad es **APIs**). Il **client amministratore** utilizza tali interfacce per selezionare il servizio richiesto e per amministrarlo.

Il client finale utilizza il servizio configurato dal **client amministratore**. Le caratteristiche fisiche dell’implementazione (Server reale, localizzazione del data center) sono irrilevanti.

9.1.7 Virtualizzazione

La **Virtualizzazione** è uno degli aspetti essenziali nel cloud computing; infatti, è principalmente grazie ad essa che gli utenti hanno l’illusione di un ambiente dedicato esclusivamente per le loro applicazioni, ed è sempre essa che svolge un ruolo critico nel concetto di scalabilità.

La virtualizzazione fornisce la necessaria astrazione affinché le unità fisiche appartenenti al livello <<infrastruttura>>, possano essere aggregate come un insieme di risorse, potenzialmente infinite.

La virtualizzazione ha introdotto un considerevole aumento di efficienza all’interno dei **data center**.

Hypervisor corrisponde ad un livello intermedio tra sistemi fisici e logici, il quale inganna il sistema operativo soprastante, facendogli credere di avere accesso diretto a delle risorse hardware in realtà virtuali. Nello specifico un Hypervisor permette di eseguire più sistemi operativi contemporaneamente sullo stesso insieme di risorse hardware.

9.1.8 Interoperabilità

È importante poter spostare le proprie applicazioni su più cloud, oppure poter usare diverse infrastrutture cloud sul quale offrire i propri servizi.

L’interoperabilità nel cloud è un concetto chiave, sul quale si stanno concentrando molti sforzi

verso un processo di standardizzazione delle operazioni e delle metodologie di deployment, al fine di garantire alcuni dei concetti basilari analizzati precedentemente quali affidabilità e durata del servizio nel tempo, anche in caso di disastro.

9.1.9 Cloud vs Grid computing

Il cloud computing è un'infrastruttura che virtualizza le risorse hardware e software. Il grid computing comprende schemi, strumenti e framework per distribuire il calcolo o i dati.

Grid e Cloud computing condividono intenzioni, architetture e tecnologie ma differiscono nei modelli di programmazione e di business, modelli di computazione, applicazioni e evitualizzazione.

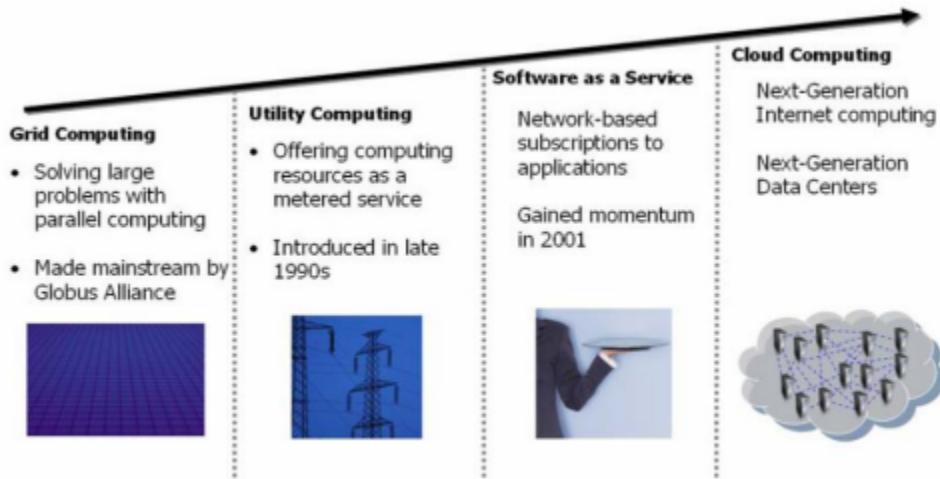


Figure 53: Cloud vs Grid Computing

9.2 Sistemi P2P strutturati : Introduzione alle Distributed Hash Tables

9.2.1 Introduzione

Nei sistemi non strutturati analizzati nei paragrafi precedenti, le risorse messe a disposizione di un peer sono memorizzate dal peer stesso. Il problema principale, dovuto alla mancanza di strutturazione della rete è quello della ricerca dell'informazione con le caratteristiche desiderate.

Si osservi, invece, che i sistemi strutturati sono basati su assunzione ben diversa, ossia una informazione condivisa può essere memorizzata su qualsiasi nodo della rete e l'associazione delle informazioni ai nodi avviene secondo un criterio ben preciso che consente poi un routing 'intelligente' delle query verso i nodi che possono soddisfarle.

Si supponga di memorizzare un'informazione I all'interno di un sistema distribuito, e che B voglia reperire I senza conoscere la sua effettiva locazione.

Come organizziamo il sistema distribuito? Quali sono i meccanismi da utilizzare per decidere ove memorizzare l'informazione e come reperirla?

Qualsiasi soluzione deve tener particolarmente in considerazione:

- **Scalabilità del sistema:** Occorre controllare l'overhead di comunicazione e la memoria utilizzata da ogni nodo, in funzione del numero dei nodi del sistema.
- **Robustezza e adattabilità:** In caso di faults e frequenti cambiamenti.

9.2.2 Analisi delle soluzioni e Confronti

- **Approccio Centralizzato:**

- Ricerca $O(1)$ poiché l'informazione viene memorizzata su un server centralizzato.
- Quantità di memoria richiesta sul server pari a $O(N)$ ove N è il numero di informazioni disponibili sul sistema.
- Banda richiesta $O(N)$ (Connessione server/rete).
- Possibilità di sottomettere al sistema queries complesse

- **Approccio Completamente Distribuito:**

- Ricerca caso pessimo $O(N^2)$ poiché ogni nodo chiede a tutti i vicini. Tuttavia vi sono possibili ottimizzazioni (TTL, permette di evitare cammini ciclici)
- Quantità di memoria richiesta $O(1)$ poiché l'informazione è condivisa e non dipende dal numero di nodi del sistema, inoltre non si utilizzano strutture dati per ottimizzare il routing della query (flooding).

Si osserva quindi che gli svantaggi del Flooding sono gli overhead di comunicazione e i falsi negativi. Per quanto riguarda l'approccio centralizzato vi è una richiesta non irrisiona di Memoria,Cpu e Banda e inoltre Fault Tolerance.

La domanda che sorge spontanea è : Esiste una soluzione che realizza un compromesso tra le due proposte?

- **Distributed Hash Table:**

- Scalabilità: $O(\log n)$
- Falsi negativi eliminati
- Auto Organizzazione: Il sistema gestisce automaticamente entrate di nuovi nodi del sistema e uscite volontarie/fallimenti.

Gli obiettivi di **Distributed Hash Tables** sono **Scalabilità, Flessibilità e Affidabilità**. Parametri più che raggiunti, infatti vi è affidabilità ai fallimenti tramite l'inserimento ed eliminazione di nodi.

Vi è la possibilità di assegnare informazioni a nuovi nodi o Re-assegnamento e re-distribuzione delle informazioni in caso di fallimento o disconnessione volontaria dei nodi dalla rete=

Inoltre vi è il bilanciamento delle informazioni tra i nodi, fondamentale per l'efficienza della ricerca. Difatti si ha che si necessitano $O(\log(N))$ hops per la ricerca del nodo che memorizza l'informazione con $O(\log(N))$ come dimensione della tabella di routing di ogni nodo.

Approccio	Memoria per Nodo	Overhead di Comunicazione	Queries Complesse	Falsi Negativi	Robustezza
Server Centrale	$O(N)$	$O(1)$	✓	✓	✗
P2P puro (flooding)	$O(1)$	$O(N^2)$	✓	✗	✓
DHT	$O(\log N)$	$O(\log N)$	✗	✓	✓

Figure 54: Riepilogo

Come avviene la gestione distribuita dei dati?

- Mapping dei nodi e dei dati nello stesso spazio di indirizzamento:

Ai peers sono associati degli identificatori unici *ID* che li individuano univocamente all'interno del sistema. Anche i dati sono associati degli identificatori unici che gli identificano univocamente nel sistema. Inoltre esiste uno **spazio logico** comune degli indirizzi per i dati e per i peer.

Si osservi che i nodi sono responsabili della gestione di una porzione dello spazio logico degli indirizzi e che la corrispondenza tra i dati ed i nodi può variare per l'inserimento/cancellazione dei nodi.

- Memorizzazione/Ricerca dei dati:

La ricerca di un dato avviene attraverso il routing verso il nodo responsabile. Ogni nodo, infatti, mantiene una tabella di routing, che fornisce una visibilità parziale del sistema.

Il routing è **Key based Routing** ossia si basa sulla conoscenza dell'ID del dato ricercato. I falsi negativi eliminati.

9.2.3 DHT: Indirizzamento

Vengono utilizzate tecniche d'indicizzazione distribuita, ossia vengono mappati sia i dati che i nodi sullo **Spazio degli indirizzi** e i nodi intermedi mantengono delle tabelle di routing e ciò permette un instradamento efficiente verso il nodo "destinazione" che si basa sull'ID dei dati (Content routing).

Tuttavia la problematica di tale approccio è che ne consegue una gestione dinamica delle tabelle di Routing (inserimenti, eliminazioni) e non vi è la possibilità di supportare queries complesse (Es. contenente wildcard).

Come avviene nello specifico?

1. Passo: Definizione dello **spazio degli indirizzi logici**, generalmente strutturato secondo un **anello logico**.

- Lo spazio lineare degli indirizzi logici $0, \dots, 2^m - 1$ è molto più grande del numero di oggetti da memorizzare ($m=N$).
- Sullo spazio è definito un ordinamento totale (operazioni in modulo)

L'associazione nodi-indirizzi logici avviene mediante una **funzione hash** e la topologia reale e logica (overlay network) non sono in genere correlate tra loro.

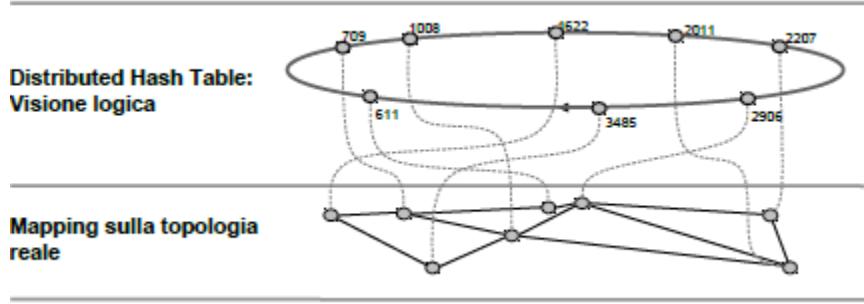


Figure 55: Topologia reale e Visione logica

- Passo: Ogni nodo è responsabile di una parte dei dati memorizzati nella DHT. In generale ad ognuno di essi viene assegnata una **porzione contingua** ove vengono mappati anche i dati da memorizzare mediante funzioni hash. Inoltre vi è la memorizzazione d'informazioni relative ai dati mappati sulla propria porzione di indirizzi. Per tali motivi spesso si introduce una certa ridondanza (overlapping).

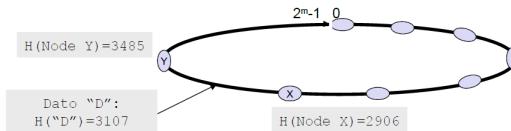


Figure 56: Visione Logica

9.2.4 DHT: Bilanciamento del carico

Vi è un problema del DHT, ossia vi è una distribuzione uniforme degli indirizzi tra i peers che definiscono la DHT stessa e per tale motivo possono occorrere sbilanciamenti del carico, difatti:

- Un nodo deve gestire una grossa porzione dello spazio degli indirizzi
- Gli spazi degli indirizzi sono distribuiti in modo uniforme tra i nodi, ma gli indirizzi gestiti da un nodo corrispondono a molti dati
- Un nodo deve gestire diverse queries, perché i dati corrispondenti agli indirizzi gestiti sono molto richiesti.

Il sbilanciamento del carico comporta minor robustezza del sistema e minore scalabilità, ossia la complessità $O(\log(N))$ non è garantita. Tuttavia la soluzione a tale problematica è l'utilizzo di algoritmi di bilanciamento del carico.

9.2.5 DHT: Routing

Dato un dato D per determinare il nodo che gestisce $key = H(D)$ si eseguono i seguenti passi:

1. La ricerca inizia in un nodo arbitrario della DHT
2. si è guidati da $H(D)$
3. Come detto in precedenza ogni nodo ha in genere una visione limitata della DHT

- Il next hop è determinato dall'algoritmo di routing utilizzato (Solitamente routing contenent based, ossia basato sulla vicinanza tra ID del dato e ID del nodo).

9.2.6 DHT: Memorizzazione Diretta e Indiretta

La DHT memorizza coppie del tipo $(key, valore)$. Si osservi che Il dato viene copiato, al momento del suo inserimento nella DHT, nel nodo che ne è responsabile secondo il mapping dati-nodi. (Tale nodo non è generalmente il nodo che ha inserito il dato nella DHT) Vi sono due tecniche di memorizzazione: dei dati:

- Diretta: Il valore è il valore del dato ricercato. Il dato viene memorizzato sul nodo responsabile dell'indirizzo restituito da $H(D)$ (key).
- Indiretta: Il valore è il riferimento al dato ricercato, ad esempio può essere l'indirizzo fisico del nodo che memorizza il contenuto. Più flessibile rispetto al precedente anche se richiede un passo in più per l'accesso al dato. Difatti per il **trasferimento dei dati** il nodo responsabile determinato invia al richiedente la coppia $(H(D), (ip/port))$, il quale dopo aver ricevuto l'esatta locazione del dato può finalmente scaricarlo $GetData(ip, port)$.

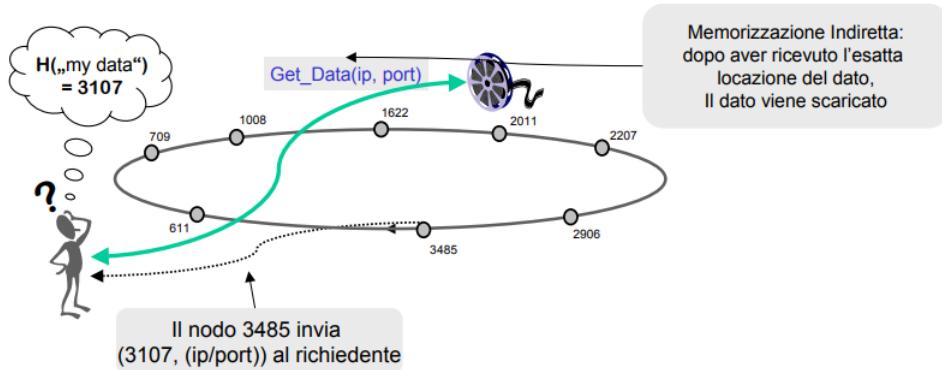


Figure 57: Memorizzazione Indiretta

9.2.7 DHT: Inserzione di nuovi nodi

Per l'inserimento di un nuovo nodo si eseguono i seguenti passi:

1. Calcolo dell'identificatore del nodo (ID).
2. Il nuovo Nodo contatta un nodo arbitrario della DHT (bootstrap).
3. Avviene l'assegnamento di una porzione dello spazio degli indirizzi ad ID
4. Copia delle chiavi K/V assegnate (in genere si utilizza ridondanza).
5. Inserzione nella DHT (collegamento con nodi vicini).

9.2.8 DHT: Ritiro/Fallimento di nodi

In caso di **Ritiro volontario di un nodo**, avviene il partizionamento della propria porzione degli indirizzi sui nodi vicini, la copia delle coppie $(Key, Valore)$ su i nodi corrispondenti e l'eliminazione

del nodo dalletabelle di routing. Invece in caso di **Fallimento di un nodo**, tutti i dati memorizzati vengono persi a meno che non siano memorizzati su altri nodi (Memorizzazione d'informazioni ridondanti anche detta replicazione). Inoltre vi è l'utilizzo di percorsi di routing alternativi/ridondanti, difatti periodicamente viene effettuato il Probing dei nodi vicini per verificarne le operatività ed in caso di fault, vi è l'aggiornamento delle routing tables.

9.2.9 DHT: Applicazioni

Un interfaccia (API) per l'accesso alla DHT necessita dei seguenti elementi:

- Inserimento di Informazione condivisa tramite $PUT(key, value)$
- Richiesta di Informazione (Content search) tramite $Get(key)$.
- Risposte, ossia $Value$.

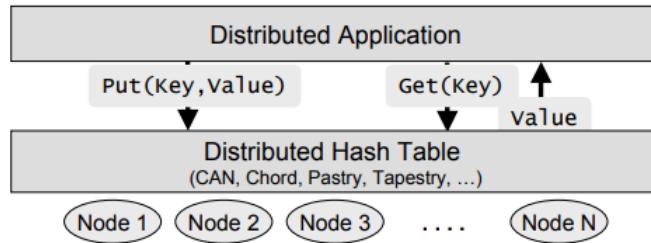


Figure 58: Interfaccia comune a molti sistemi basati su DHT

Riassumendo le DHT offrono un servizio generico distribuito per la memorizzazione e l'indicizzazione di informazioni e il valore memorizzato in corrispondenza di una chiave può essere un file, un indirizzo IP o qualsiasi altro dato. Applicazioni che possono utilizzare le DHT sono realizzazioni di DNS o P2P storage system (Freenet).

9.2.10 Conclusioni

- Il contenuto è basato sul contenuto della query
- Le chiavi sono equamente distribuite tra i nodi appartenenti alla DHT e così facendo si evitano colli di bottiglia, si supporta l'inserzione incrementale di chiavi nel sistema e si è tolleranti ai guasti.
- Sistemi auto-organizzanti
- Realizzazione semplice ed efficiente.
- Supportano un ampio spettro di applicazioni in cui i valori associati alle chiavi dipendono dalle medesime applicazioni.

Esempio di sistemi esistenti Chord, Pastry, Tapestry, CAN, P-Grid Ed altri.

10 Lezione n10 22 04 2022

10.1 Sistemi P2P strutturati: Chord

10.1.1 Introduzione

Chord è un sistema DHT, basato su pochi e semplici concetti **Facilità di comprensione e di implementazione, Eleganza e Possibilità di definire ottimizzazione**. Le cui caratteristiche principali sono le seguenti:

- **Routing:** Lo spazio logico degli indirizzi è piatto, ossia gli indirizzi sono identificatori di m-bits invece che indirizzi IP. Inoltre è caratterizzato da un Routing efficiente con alta probabilità di complessità pari $\log(N)$ hops ove N è il numero totale di nodi del sistema. Anche la dimensione delle tabelle di routing con alta probabilità è pari ad $\log(N)$.
- **Auto-organizzazione:** Gestisce inserzione di nuovi nodi, ritiri volontari del sistema e fallimenti.

10.1.2 Applicazioni

Può essere applicato per:

- **Cooperative Mirroring:** Distribuire l'informazione gestita da un insieme di server sui nodi di una rete CHORD.
- **Shared Storage:** Mantenere online il contenuto fornito da un nodo che si connette in modo intermittente alla rete, memorizzando il contenuto su altri nodi della medesima rete. Tale concetto sta alla base della **Cooperazione**, ossia utilizzo lo spazio su disco di altri peer per memorizzare il mio contenuto e offro il mio spazio per memorizzare il contenuto di altri.
- **Indici Distribuiti:** Definire un indice distribuito per Gnutella ove le chiavi sono parole che identificano un file e i valori sono indirizzi delle macchine che memorizzano quel contenuto.

10.1.3 Topologia

Chord si basa sull'hash-table storage, ovvero per inserire dati viene utilizzata la $put(key, value)$ e per ricercare i dati la $get(key)$. In particolare ad ogni nodo (host) viene associato un **Identificatore ID**, generato mediante Hash (Si utilizza SHA-1).

$$ID = sha - 1(indirizzoIP, porta)$$

E ad ogni dato, invece, viene associata una chiave **key**

$$key = sha - 1(dato)$$

Tuttavia **SHA-1** (Secure Hash Standard) è utilizzabile per assegnare identificatori unici a chiavi e nodi, solo se la **probabilità**, che due nodi diversi o due chiavi diverse generino lo stesso identificatore, è trascurabile. A tale scopo è necessario utilizzare identificatori sufficientemente lunghi.

$$160 - bit \quad - > \quad 0 <= identificatore <= 2^{160}$$

Si osservi che chiavi e identificatori sono mappati **nello stesso spazio degli indirizzi**. Per quanto riguarda l'associazione delle chiavi ai nodi, quest'ultima avviene mediante **consistent hashing** che garantisce:

- Bilanciamento del carico: le chiavi vengono distribuite uniformemente tra i nodi
- L'inserimento/eliminazione di un nodo comporta lo spostamento di un **numero limitato** di chiavi. Se inserisco l'N-esimo nodo sposto $1/N$ chiavi.
- **ESEMPIO:**
Si considerino identificatore di m bits

$$[0, 2^m - 1]$$

e che vi sia un ordinamento tra gli identificatori, in base al loro valore numerico (Il successore di $2^m - 1$ è 0. Possiamo quindi rappresentare l'ordinamento mediante un semplice **anello** che prende il nome di **Chord Ring**. Descriviamo ora l'algoritmo di Mapping:

Si assegna una chiave di valore K al primo nodo n dell'anello il cui identificatore risulti maggiore o uguale a K

$$n(key) = successor(key)$$

Quindi n è il primo nodo individuato partendo da K e proseguendo sull'anello Chord in senso orario.

Tale nodo è chiamato "successor node" della chiave k , denotato da $successor(key)$.

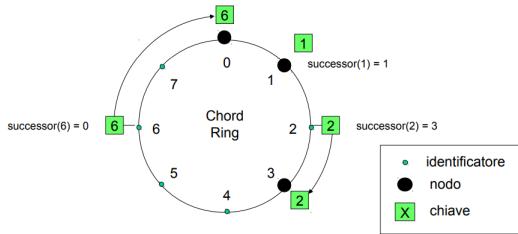


Figure 59: Identifier circle con $m=3$

La Figura 59 presenta 3 nodi e si ottiene quindi che il successore dell'identificativo 1 è 1, così la chiave 1 dovrebbe essere allocata al nodo 1. Similmente la chiave 2 dovrebbe essere allocata al nodo 3 e la chiave 6 al nodo 0. In tal maniera il **consistent hashing** permette ai nodi di entrare e lasciare la rete con il minimo disordine.

Quando un nodo lascia la rete, ognuna delle sue chiavi viene riassegnata al successore di n e quando un nodo "n" entra nella rete inevitabilmente alcune chiavi assegnate al "successore di n" vengono assegnate al nodo n entrato.

10.1.4 Overlay

L'overlay è determinato dai links stabiliti tra i nodi dell'anello. Memorizzati nella **Routing Table** di ogni nodo, rappresentano la conoscenza, che un nodo possiede degli altri nodi facente parte dell'anello.

La topologia più semplice è la **lista circolare**, ossia ogni nodo possiede un link verso il nodo successivo in senso orario.

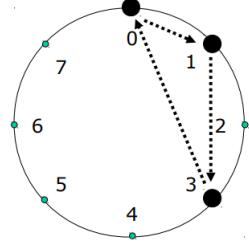


Figure 60: Esempio

10.1.5 Routing

In un ambiente distribuito è necessario un numero veramente piccolo d'informazioni di routing per implementare "consistent hashing". Difatti ogni nodo ha la necessità di possedere solo un link verso il suo successore e inoltrare la query per una **chiave** x , lungo l'anello finché non viene individuato

$$n = \text{successor}(x)$$

Ottenendo così un algoritmo semplice e una dimensione delle Tabelle di routing pari a $O(1)$. Tuttavia questo schema di risoluzione è inefficiente, poiché potrebbe richiedere di attraversare tutti gli N nodi per trovare l'assegnazione adeguata (Lineare, NON va bene) e inoltre il fallimento di un nodo interromperebbe i collegamenti sull'anello.

Per accelerare questo processo (e risolvere le problematiche legate), Chord memorizza informazioni di routing addizionali e utilizza strutture dati ad hoc.

- **Informazioni memorizzate:**

Puntatore al nodo successore e al nodo predecessore sull'anello. L'elenco delle chiavi mappate sul nodo e un insieme di puntatori a nodi successivi necessari per garantire la consistenza della rete in caso di join/leave dinamici dei nodi.

- **Finger table (Struttura dati):**

Ogni nodo n mantiene una tabella di routing con al più m elementi (in realtà sono meno) chiamata **"finger table"**.

L' i -esimo elemento della tabella del nodo n contiene l'**identità del primo nodo** s che si trovano a una distanza di almeno 2^{i-1} ove:

$$1 \leq i \leq m \quad s = \text{successor}(n + 2^{i-1})$$

Si osservi che un elemento della finger table include insieme al **Chord Identifier**, l'indirizzo IP e numero di porta del nodo rilevante.

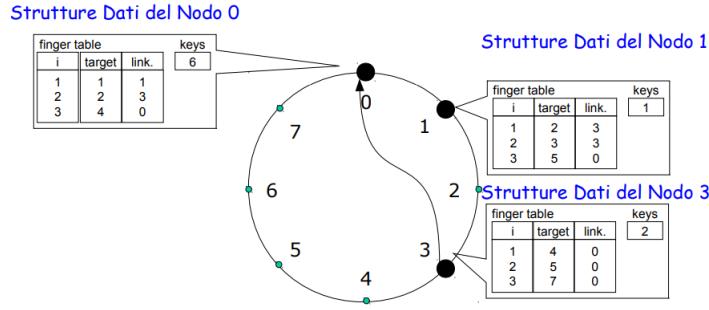


Figure 61: Esempio

Nell'esempio proposto la finger table del nodo 1 punta ai nodi successori degli identifiers (target):

$$(n + 2^0) \bmod 2^3 = 2$$

$$(n + 2^1) \bmod 2^3 = 3$$

$$(n + 2^2) \bmod 2^3 = 5$$

Il successore di 2 è il nodo 3, il successore di 3 è il nodo 3 e il successore di 5 è il nodo 0.

Tale schema ha due importanti caratteristiche:

1. Ogni nodo mantiene diversi links verso alcuni nodi vicini dell'anello e solo un piccolo numero di altri nodi lontani. Da ciò ne consegue un routing accurato in prossimità di un nodo e più approssimato verso nodi lontani.
2. Una finger table di un nodo generalmente contiene un numero limitato di links, non abbastanza per determinare il successore di un'arbitraria chiave k . Quindi cosa succede quando un nodo n non conosce il successore di una chiave k ? Speghiamo l'algoritmo di Routing di Chord.

Ogni nodo n propaga la query per la chiave k al **finger più distante che precede k**, in senso orario e la propagazione continua fino al nodo n tale che:

$$n = predecessore(k) \quad AND \quad successore(n) = successore(k)$$

Se valgono tali proprietà allora il successore di n possiede la chiave k .

10.1.6 Auto Organizzazione

Chord è in grado di gestire in modo dinamico i cambiamenti della rete:

1. Fallimento di nodi.
2. Fallimenti nella rete.
3. Arrivo di nuovi nodi.
4. Ritiro volontario dei nodi dalla rete.

Ossia mantiene consistente lo stato del sistema in presenza di cambiamenti dinamici e per soddisfare tale scopo Chord preserva le seguenti caratteristiche:

- L'aggiornamento dell'informazione necessaria per il routing.
 - **Correttezza del Routing:** ogni nodo mantiene aggiornato il suo effettivo successore sull'anello
 - **Efficienza del Routing:** strettamente legata all'aggiornamento tempestivo delle finger table.
- Tolleranza ai guasti

Ma come fa a preservare le suddette caratteristiche quando avviene l'inserimento di nodi all'interno della rete:

- **Inserimento di nodi:**

1. Il nuovo nodo n sceglie un identificatore ID . La scelta dell'identificatore può avvenire nei seguenti modi:
 - Nel caso la distribuzione sia uniforme l' ID può essere calcolato in maniera randomica.
 - L' ID può essere ottenuto dall'hash dell'indirizzo IP + porta del nodo.
 - Una terza possibilità è utilizzare specifici algoritmi di generazione di ID basati sulla conoscenza del carico dei nodi attivi sull'anello e la locazione geografica.
2. n contatta un nodo esistente sulla rete, in particolare un **bootstrap node** (Riferimento al nodo ottenuto mediante un bootstrap servers, anche tramite DNS aliases).
3. Individua il suo successore sull'anello e vi si aggancia in maniera **immediata**. Ma come fa a trovare il proprio successore?
 - Il nuovo nodo invia al nodo di bootstrap una query con chiave uguale al proprio identificatore.
 - Il risultato della query sarà il successore della chiave che coincide con il successore del nuovo nodo (chiave e id sono uguali).
4. Costruisce la propria finger table (può essere lazy). In particolare:
 - (a) Viene definita una struttura di m entrate (Ossia $\log(n)$ entrate)
 - (b) Si itera sulle righe della finger table e si determinano gli *identifiers* (target).

$$finger[i] \quad successor(n + 2^{i-1})$$

$$1 \leq i \leq m$$

- (c) Per ogni riga viene inviata una query al nodo di bootstrap (entry point) per trovare il successore del target, ossia il link che deve essere inserito nella finger table. (Dall'entry point viene utilizzato l'algoritmo di routing standard di Chord).

Dopo i passi illustrati in precedenza, il nuovo nodo n conosce il suo successore sull'anello e possiede una finger table valida, ma gli altri nodi dell'anello non sono consapevoli della sua presenza. A tale scopo i nodi **eseguono periodicamente** alcune procedure per acquisire conoscenza sui nuovi nodi che si sono inseriti nella rete.

5. Aggiornamento dei finger dei nodi esistenti: Il nodo n necessita di essere inserito nelle finger table di alcuni nodi esistenti. In particolare le finger table degli altri nodi dell'anello (non di quello che si è inserito) vengono aggiornate in modo lazy, ossia scelto casualmente e periodicamente un finger, si sceglie un entrata nella sua finger table (Sempre in modo casuale) e si sovrascrive il suo valore con:

$$\text{finger}[j] = \text{successor}(n + 2^{j-1})$$

j scelto casualmente

6. Trasferimento delle chiavi: l'ultima operazione che deve essere eseguita quando un nodo entra nella rete è spostare tutta la responsabilità sulle chiavi di cui ora è il nuovo successore al fine di rispettare la strategia di allocazione chiavi-nodi. Tale procedura deve essere effettuata dalla applicazione e tipicamente si basa sul copiare tutte le chiavi minori o uguali a n dal proprio successore.

PseudoCodice:

```

1 #Creazione di un nuovo nodo n
2 n.create(){
3     predecessor=null;
4     successor=n;
5 }
6
7 #Inserzione del nodo n nell'anello contenente il nodo n'
8 n.join(n'){
9     predecessor=null;
10    successor= n'.find_successor(n);
11 }
12 #Fino a che la rete non si stabilizza, solo il successore di
13 #e' consapevole della presenza di n sulla rete.
14
15 #Aggiorna il puntatore al successore
16 n.stabilize(){
17     x=successor.predecessor;
18     if( x in (n,successor))
19         successor=x;
20     successor.notify(n);
21 }
22 n.notify(n'){
23     if(predecessor==null or n' in (predecessor,n))
24         predecessor=n';
25 }
26 #Aggiorna la finger table di ogni nodo, inserendovi eventuali
27 #riferimenti a nuovi nodi
28 n.fixFingers(){
29     next=Scelta casuale di un entrata nella finger table
30     finger[next]= find_successor(n+2^(j-1))
31 }
```

Si osservi che tutti i link di un nodo sono correttamente aggiornati solo quando:

- a È stato aggiornato il successore del nodo considerando eventuali nodi inseriti (stabilize()).
- b È stato eseguito l'aggiornamento delle finger table sia del nodo che ha effettuato la join() sia degli altri nodi (fixfingers()).

- **Concetti Chiave da ricordare:**

- **Lazy Join:** Quando un nodo si inserisce sull’anello può inizializzare solo il suo successore e periodicamente rinfresca il contenuto della propria finger table.
- La correttezza della ricerca dipende dalla corretta impostazione del nodo successore di ogni nodo appartenente all’anello Chord
- La migrazione delle risorse al nuovo nodo non è gestita da Chord, ma deve essere eseguita dalla applicazione.
- nel caso si effettui una ricerca prima che il sistema sia ritornato in uno stato stabile successivamente all’inserimento di un nodo accade che:
 - * Non tutti i puntatori ai nodi successivi sono corretti oppure le chiavi non sono completamente trasferite. La ricerca può fallire ed occorre rientrare in seguito.
 - * Ogni nodo ha aggiornato il suo successore reale sull’anello e le chiavi sono state correttamente trasferite tra nodi, le fingers table possono contenere informazioni non completamente aggiornate. La ricerca ha esito positivo, ma può essere rallentata.
 - * Tutte le finger table sono in uno stato ”ragionevolmente” aggiornato, allora il routing viene garantito in $O(\log N)$ passi.
- Chord è **Key-Based Routing:** ogni nodo riceve la chiave e la instrada ad un altro nodo della rete scelto nella propria finger table. Tale scelta è guidata dal valore della chiave. L’Instradamento continua fino a che non si è individuato il successore della chiave sull’anello.
- Le operazioni che possono essere richieste da un qualsiasi nodo della rete sono la put(key,value) e la get(key). In entrambi i casi il nodo fornisce la chiave dell’informazione che vuole memorizzare/ricercare.

10.1.7 Conclusioni sull’inserimento/Caduta dinamica dei nodi

In generale potremo dire che tutte le operazioni relative all’inserimento/caduta di un nodo sono corrette se avvengono quando l’anello di trova in uno stato stabile. Tuttavia l’anello necessita di un intervallo di tempo tra due operazioni successive, per stabilizzarsi e per tal motivo quest’ultimo potrebbe non essersi stabilizzato prima di un nuovo inserimento/cancellazione.

Da ciò si ottiene un **risultato generale:** Se i protocolli di stabilizzazione dell’anello vengono eseguiti con una frequenza opportuna, dipendente dalla frequenza di aggiornamenti (inserimenti/-fallimenti) allora l’anello rimane costantemente in uno stato stabile, in cui il routing rimane corretto e veloce (Si mantiene il limite di complessità $\log(N)$).

10.1.8 Fallimento di Nodi

1. Caso:

Ogni comunicazione con i finger è controllata tramite **time-outs**. Se il time-out scade il mittente prende atto che il finger non è più presente e sceglie un nuovo destinatario anzichè far fallire la query, ossia:

- La query viene inviata al **finger precedente**, per evitare di oltrepassare il nodo destinazione.
- Il finger caduto viene rimpiazzato con il suo successore nella finger table (Trigger Repair). (Ricordare **finger[i]** punta a $successor(n + 2^i - 1)$, $1 \leq i \leq m$.

2. Caso:

Analizziamo ora una **situazione inconsistente**, ossia un nodo perde il riferimento al suo vero successore sull'anello.

Si osserva che la correttezza dell'algoritmo di routing è garantita se ogni nodo mantiene aggiornato il **riferimento al nodo successivo** anche in caso di fallimenti multipli, ma tuttavia in caso di fallimenti simultanei questo invariante non può essere garantito.

Per migliorare, infatti la robustezza del sistema si ha:

- Ogni nodo n mantiene una lista dei suoi r immediati successori sull'anello.

$$r = \log(n)$$

r è logaritmico rispetto al numero di nodi.

- Se n rileva la caduta del suo successore, il successore viene sostituito con il secondo elemento della lista e così e via.
- La lista viene mantenuta consistente mediante la procedura di stabilizzazione e inoltre valori crescenti di r rendono il sistema maggiormente robusto.

Approfondiamo però la struttura dati introdotta, ossia **La lista dei successori**: Come detto in precedenza la lista dei successori di n include i primi r successori di n sull'anello, in senso orario. Inoltre il nodo n richiede la lista dei successori al suo immediato successore s , rimuove l'ultimo elemento e inserisce s in testa. Se il successore cade, viene sostituito con l'elemento successivo della lista

Look up: Ricerca nella finger table + lista successori

Si osservi che avendo una lista di r nodi successivi il caso precedentemente visualizzato risulta risolto, tranne nel caso in cui gli r nodi falliscano contemporaneamente (Altamente Improbabile).

Si ha quindi che per ottenere la correttezza della finger table, viene svolto un controllo periodico della caduta dei finger e la sostituzione con nodi attivi. Ciò provoca un traffico aggiuntivo per la gestione dei fallimenti ma una maggiore correttezza e velocità nel reperimento delle informazioni (tradeoff)

10.1.9 Uscita volontaria di un nodo

L'uscita volontaria di un nodo può essere trattata come il caso precedente (Fault) per semplicità. Tuttavia possono essere effettuate delle ottimizzazioni:

- Il nodo notifica l'intenzione di abbandonare l'anello al successore, al predecessore ed ai fingers.
- Il predecessore può rimuovere n dalla lista dei successori.
- Il predecessore può aggiungere alla sua lista di successori il primo nodo della lista di successori di n .
- Trasferisce le chiavi di cui è responsabile al suo successore.

10.1.10 Replicazione dei dati

Chord non garantisce l'affidabilità dei dati in presenza di fallimenti, ma fornisce dei meccanismi per tale scopo. In particolare ogni applicazione che utilizza il livello Chord, può utilizzare la lista dei successori di un nodo per garantire che un dato venga **replicato** negli r successori. Nel caso in cui un nodo fallisca, il sistema può utilizzare le repliche per individuarne i dati in modo corretto.

10.1.11 Scelte di progetto

- **Approccio a livelli:**

Chord è responsabile del routing e la gestione dei dati è demandata alle applicazioni.

- **Approccio Soft:**

I nodi hanno il compito di cancellare le **coppie (key,value)** successivamente allo scadere di un intervallo di tempo detto **periodo di refresh** (Parte dall'ultimo inserimento). Le applicazioni, invece, effettuano il **refresh periodico** delle coppie (key,value). Procedendo in tal modo si attribuiscono le informazioni ai nuovi nodi arrivati.

Si osserva quindi che se un nodo fallisce occorre aspettare il periodo di refresh per ottenere le informazioni nuovamente disponibili.

10.1.12 Simulazione

- rete = 2^k nodi, 100×2^k chiavi, k variabile tra 3 e 14.
- per ogni valore di k , si considera un insieme casuale di chiavi e si effettua un esperimento separato
- Per ogni chiave si valuta il numero di hops per la ricerca
- $\text{Lookup Path Length} \sim \frac{1}{2} \log_2(N)$
- Conferma dei risultati teorici
- Il grafico mostra il 1 il 99 percentile e la media

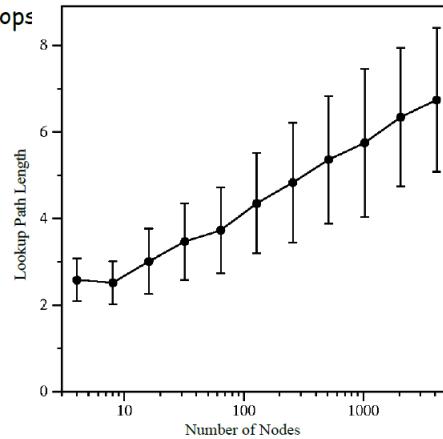


Figure 62: Simulazione

10.1.13 Conclusioni

- **Complessità:**

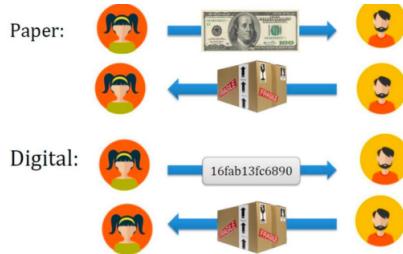
- **Messaggi di lookup:** $O(\log N)$ hops
- **Memoria per node:** $O(\log N)$ entrate nella tabella di routing
- **Messaggi per auto-organizzazione** (join/leave/fail):

- **Vantaggi:** Semplice, flessibile, Modelli teorici e prove di complessità.
- **Svantaggi:** Manca una nozione di prossimità fisica e limite sulle possibili situazioni.
- **Ottimizzazioni:** Prossimità, links bi-direzionali, load balancing.

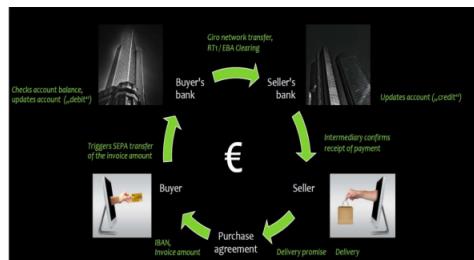
11 Lezione n11 06 05 2022

11.1 Il caso dei Bitcoin

Le **Criptovalute** sono valute "virtuali" che possono essere utilizzate per pagamenti digitali. Generalmente i pagamenti digitali possiedono le seguenti caratteristiche:



- Necessitano di un **Trusted Server**, (Financial institution) poiché non avviene circolazione di "denaro digitale" ma la trascrizione di transazioni.
- Nessuna anonimità
- Commissioni per transazioni elevate.



11.1.1 Payment transaction: Who trust whom?

In un ordine online durante il **Pre-pagamento** l'acquirente effettua un atto di fiducia verso il venditore il quale potrebbe anche non mantenere la promessa di effettuare la spedizione. Certamente si ha che tutte e due le parti hanno fiducia nelle loro banche e le banche si fidano tra di loro. In particolare vi è un registro **centralizzato** dell'acquirente che previene il fenomeno del **double-spending**

Il denaro può essere speso una volta sola, ossia una banconota da 100euro non può essere utilizzata per due acquisti diversi (in un sistema centralizzato facile da gestire ma in uno distribuito??).

Un'azienda in particolare ha avuto molto successo per quanto riguarda la garanzia nei pagamenti digitali, ossia **Paypal**. Paypal si presenta come un **intermediary service**, ossia una compagnia tra l'acquirente e il venditore in modo tale che l'acquirente dia la propria carta di credito all'intermediario invece che al venditore e che mette a disposizione protocolli di crittografia standardizzati. Il funzionamento si basa su un sistema centralizzato che richiede un intermediario per l'iscrizione ed il collegamento del servizio all'account bancario del fruitore.

Sulle slide viene descritta in breve che i bitcoin vennero rilasciati nell'ottobre del 2008 come Peer to Peer Electronic Cash System. Che iniziarono ad esserci i primi mining di Bitcoin e la prima transazione tra Hal Finney and Satoshi. Inoltre viene raccontata una curiosa vicenda di come "laszlo" (Il nickname di una persona) chiese a domicilio due pizze per 10000 bitcoins. Inoltre il professore afferma che il 20% di bitcoin in circolazione sono persi, poiché ogni utente per accedere al proprio wallet necessita di una chiave privata, se si perde tale chiave i soldi rimangono nel blockchain ma non possono essere utilizzati, anche se oggi i sistemi odierni non soffrono più di tali problematiche.

11.1.2 Bitcoin main goal

I bitcoin permette un sistemi di pagamenti senza intermediari. In particolare si tratta di una versione peer-to-peer dei "soldi digitali" poiché non vi è nessuna autorità che controlla e nessun server. Difatti avendo tal tipo di approccio sono considerati "sospetti" visto l'ambiente totalmente inaffidabile.

Alla base della crescita "legit" del bitcoin vi è **Nakamoto** (lui,lei non si sa) che riuscì a risolvere il problema dei bitcoin nell'essere "copiati" oltre che a identificare intenzioni malevoli, così da far crescere il consenso generale verso questa nuova moneta. Mettiamo ora in evidenza le principali

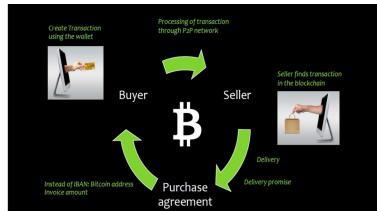


Figure 63: Payment senza intermediari

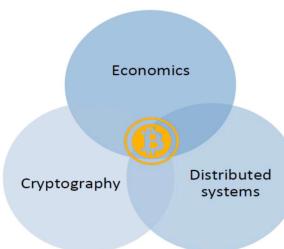
differenza tra i BitCoin e una Banca.

- Bitcoin:

Sono **openness**, ossia tutto ciò di cui hai bisogno è **Bitcoin client**. Nessun account bancario, nessuna carta di credito, nessuna entity centralizzata che controlla il denaro e nessuna informazioni sulle transazioni effettuate vengono inviate a terzi. Infatti vi è una **Pseudo anonymity**, le transazioni non sono caratterizzate da informazioni aggiuntive sul pagamento effettuato e gli indirizzi (credo si riferisca al conto) vengono utilizzati come pseudonimi. Inoltre vi sono tasse non molto elevate.

- Sistemi maggiormente utilizzati:

Necessitano di un server affidabile e di un istituzione finanziaria, non vi è anonimità e per ogni transazione vi sono tasse elevate da pagare.



11.1.3 Perchè il bitcoin divenne così popolare?

Vi sono diverse ragioni per cui il bitcoin oggi come oggi è una moneta molto popolare. **La ragione ideologica** è la **cryptoanarchy** poiché nessuno controlla il bitcoin, oltre all'avanzare del **cyberpunk movement**. Inoltre il bitcoin nacque durante la crisi finanziaria del 2008, tant'è che a volte viene additato come "il figlio della crisi". A differenza degli altri tipi di moneta, il bitcoin non necessita di essere stampato.

Vi è anche un motivo tecnologico dietro alla crescita del bitcoin, ovvero l'avanzare dei sistemi distribuiti. Per non parlare delle tasse molto basse per ogni pagamento, oppure delle possibilità in ambito "illegal" data la **pseudonimity** che caratterizza questa moneta. Inoltre negli ultimi anni sempre più aziende accettano transazioni in Bitcoin, e tutto ciò porta a una reale esistenza dell'economia del Bitcoin.

Tutta via tra i **Problemi** più grandi del bitcoin vi è che alla base di quest'ultima non c'è nessuna autorità centrale che gestisce possibili minacce di sicurezza come lo sfruttamento della criptovaluta per azioni illegali. Tant'è che in molti pensano che dietro al bitcoin vi sia lo **Schema Ponzi**, ma ciò solo per il continuo e improvviso oscillamento del valore intrinseco dell'oggetto. Quando la realtà dei fatti è che vi è un vero e proprio mercato alla base.

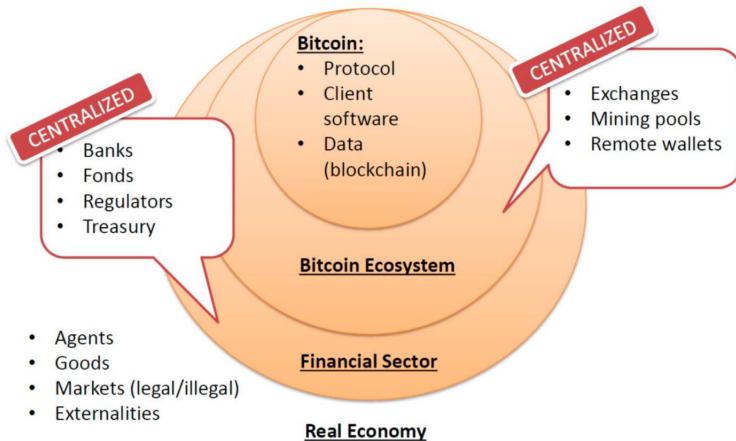


Figure 64: Bitcoin in context

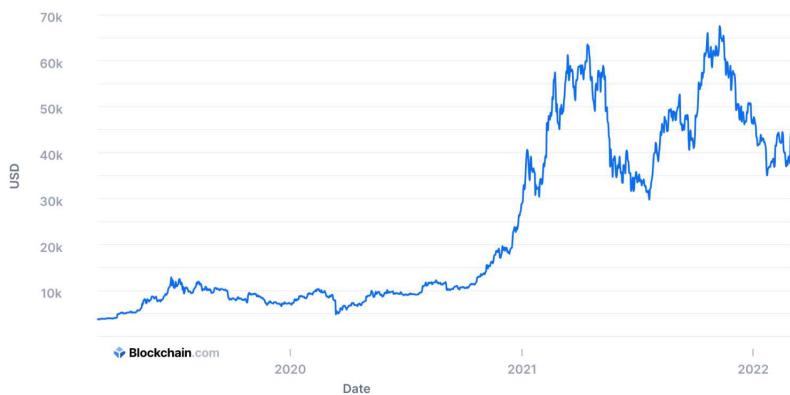


Figure 65: Price fluctuation

12 Lezione n12 13/05/2022

12.1 Il problema principale della Moneta Digitale

Il problema principale è il **double spending**. Difatti chi ha la responsabilità di gestire la moneta e prevenire che la stessa moneta digitale venga spesa più volte, se non c'è un entità centrale? Anche perché le monete digitali sono rappresentate da una sequenza di bit, e ciò li rende facilmente "duplicabili", più della carta, è sufficiente inviare a più persone contemporaneamente la medesima sequenza di bit.

Per risolvere tale problema, l'idea è quella di avere un **public trusted bulletin-board** o meglio **ledger of transaction**, ossia un registro bancario che tiene traccia degli spostamenti di denaro (entrate ed uscite).

User P_1 transfer a coin #16fab13 to user P_2

Se l'utente P_1 tenta di trasferire la stessa moneta con id #16fab13 a un Utente P_3 , questa risulterà già registrata al ledger dell'Utente P_2 .

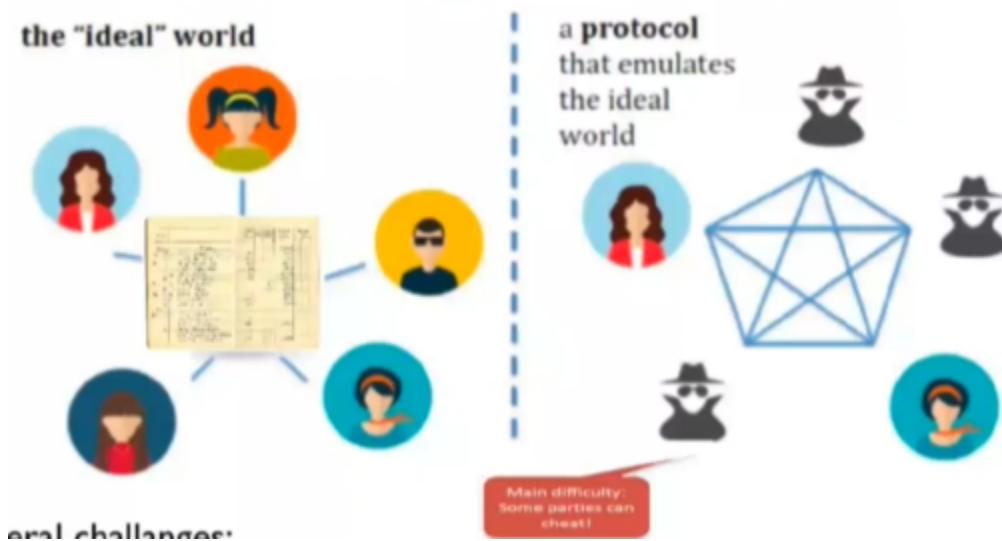


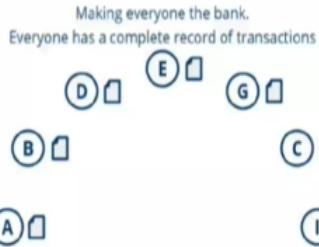
Figure 66: Esempio

Tuttavia, pur essendo una soluzione ottima nel caso in cui la maggior parte dei partecipanti siano **Utenti Onesti**, vanno affrontati anche i seguenti problemi:

1. Alcune persone potrebbero imbrogliare
2. Il sistema è distribuito, quindi va tenuto conto anche del delay,jitter che variano al variare della rete in uso.

12.1.1 Ledger of transaction distribuito

Vi è un **public ledger** condiviso che tiene conto di tutte le transazioni effettuate. In particolare tale **ledger** è replicato in ogni nodo (Ogni nodo rappresenta un utente), cosicché tutti possano tener conto dei "record" effettuati e a chi appartiene una determinata moneta (un determinato bitcoin). Data la latenza della rete, la copia che ogni nodo ha, tende prima o poi a essere consistente con tutte le altre (Non è possibile che ciò accade istantaneamente). Ne consegue quindi che la banca è costruita collettivamente da tutti i partecipanti.



In Bitcoin, il **ledger** è memorizzata nel **blockchain** e intuitivamente può essere vista come una lista di transazioni, più concretamente si parla di una lista immodificabile di blocchi contenente transazioni (Nel caso si volesse aggiungere un gruppo di transazioni, allora avviene l'aggiunta di un nuovo blocco e non la modifica di uno precedente, ecco perché immodificabile).

- **Funzionamento:**

Si ipotizzi che un Utente1 invii una certa quantità di Bitcoin a un Utente2. Allora L'utente1 registra l'ammontare inviato nel proprio ledger. Successivamente l'Utente2 si assicura che l'ammontare ricevuto sia effettivamente dell'Utente1 tramite il proprio ledger. Se tale verifica ha successo, allora la transazione effettuata viene inoltrata a tutta la rete, e tutti gli altri partecipanti (peer) effettuano il check della transazione e se valida l'aggiungono al proprio ledger.

- **Probabile Problema:**

L'Utente1 però potrebbe mandare gli stessi bitcoin a un Utente3 e vi sono due situazioni possibili:

- Il problema viene rilevato:

Viene rilevato **se** l'Utente3 riceve la notifica della transazione dall'Utente2 prima del trasferimento di denaro da parte dell'Utente1

L'utente3 dice, fra vedi che sti cazzo di bitcoin li hai mandati già all'utente2, non prendermi per il culo.

- Il problema non viene rilevato:

l'utente3 e l'Utente2 non hanno ricevuto ancora il broadcast da parte dell'altro e assumono quindi che la transazione è valida e l'accettano.

L'utente1 effettua un'analisi della rete per prendere per il culo i due partecipanti, oppure un attacco DOS (Denial of Service), ossia un sovraccarico della rete per impedire ai due utenti di comunicare tra loro.

- **La necessità di un algoritmo di consenso:**

L'**algoritmo di consenso** permette di superare la problematica affrontata precedentemente. In particolare tutti i partecipanti sono responsabili della validazione di una transazione, ossia una transazione è valida solo se è valida per la maggior parte dei nodi.

Tuttavia, pur essendo l'idea implementabile e ragionevole, alcuni partecipanti potrebbero organizzarsi e aggirare il sistema per atti illeciti, quindi l'idea risolve il problema solo in un mondo ideale di partecipanti onesti (Non va bene).

Ma assumendo che la maggior parte degli utenti siano onesti e d'implementare il **ledger** tramite voto allora si effettua il broadcast di ogni transazione sulla rete e si collezionano i voti per determinarne la validità.



Transaction Id	Value
ddbs21239864k...	0.084 BTC
edd98763hn3nr...	1.2 BTC
mkk8765g4g2j3...	0.036 BTC

Tale approccio in **cryptocurrencies** è detto **consensus protocol**. E come visto in precedenza soffre di **Sybil attack**, ossia la rete può essere acceduta da tutti, non vi è modo di verificare l'onestà di chi si collega, e lo stesso utente può registrarsi un numero arbitrario di volte in modo tale da creare una maggioranza di partecipanti disonesti.

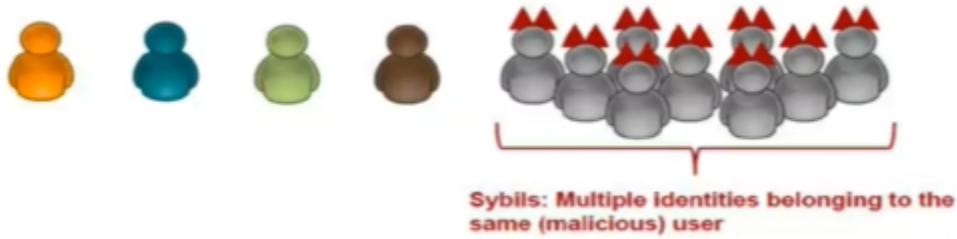


Figure 67: Sybils

Il **Sibil attack** è un problema molto discusso nel **Distributed Computing**, perché molto facile da realizzare nelle reti P2P, ove un nodo può possedere identità multiple.

Il problema potrebbe essere risolto, assegnando la possibilità a ogni nodo di votare una e una sola volta. Quindi un utente pur possedendo identità multiple, non potrà compiere atti illeciti. Tale strategia di **voting** è basata sulla **potenza di calcolo**. In altri termini non si esprime un voto per partecipante, ma **si esprime un voto per potenza computazionale disponibile**.

Così facendo creare identità multiple non funziona più, se tutte le identità sono eseguite sullo stesso nodo. (Si è fixato quindi un semplice artificio)

La potenza viene misurata tramite la **Proof of work** (POW).

- **Terminologia per entrare meglio nel contesto analizzato:**

- Miners: sono gli utenti che lavorano per mantenere il **bulletin-board** (ledger distribuito), confermare le transazioni e creare nuovi blocchi per contenere le transazioni generate. Creano e mantengono quindi la blockchain.

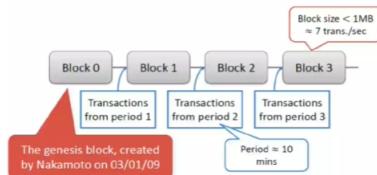


Figure 68: Blockchain

12.1.2 The Bitcoin blockchain

La **Blockchain** è un database distribuito contenente tutte le transazioni di bitcoin a partire dall'esistenza stessa del bitcoin, ossia dal 2009. Consiste in una lineare lista di blocchi, ove ogni blocco è costituito da:

- **Block Header**
- **Una lista di transazione**

La dimensione della lista non rappresenta un problema, perché la dimensione dei dischi cresce molto velocemente, inoltre la blockchain odierna potrebbe essere scaricata molto tranquillamente su qualsiasi computer.

Sulla lista possono essere effettuate solo delle aggiunte di blocchi e tali aggiunte sono eseguite **collettivamente dai nodi partecipanti alla Bitcoin P2P network**. L'azione di aggiunta viene detta **mining**, gli esecutori dell'azione vengono detti **miners**.

I miners effettuano tasks riguardanti la soluzione di un **problema computazionale difficile** e un minatore che determina la soluzione effettua l'aggiunta di un blocco alla **blockchain**, viene premiato con nuovi bitcoin che vengono creati (I bitcoin vengono creati solo in questa determinata maniera).

Se non trovo la soluzione, ho solo sprecato corrente elettrica. Ecco perché solitamente ci si iscrive a un mining pull, ove si mette a disposizione il proprio computer. Se il pull vince, la quota in bitcoin ottenuta viene divisa tra i partecipanti in base alla potenza di calcolo messa a disposizione.

12.1.3 Block and transaction

La transazione generica *Transaction(tx)* può essere creata da chiunque possieda unità di cripto valute ed effettua un trasferimento di quest'ultime. Il **Blocco**, invece, incapsula al suo interno un numero di transazioni, può essere creato solo dai **miners** e l'intestazione è costituita dai seguenti campi:

- **nVersion:** Indica la versione attuale del protocollo. Difatti il protocollo cambia pian piano tramite bug fixing o nuove features. Inoltre vi possono essere soft and hard forks (Delle vere e proprie biforcati della block chain temporanee o permanenti, biforcati nel senso che si creano due liste diverse a partire dall'ultimo nodo)
- **Hash of the header of the previous block:** Puntatore che si ottiene applicando *SHA-256* due volte all'header (nessun campo escluso) del blocco precedente
- **Hash of the merkle root:** Il Merkle tree è una struttura dati utilizzata per memorizzare in maniera sintetica l'hash di un insieme di campi. In questo contesto vi è un albero i quali nodi foglia contengono l'hash crittografico di un blocco di dati, ogni nodo che si trova ai livelli superiori contiene l'hash crittografico della concatenazione dei suoi figli e così via dicendo fino ad arrivare alla root.

Il root è l'hash di tutti i nodi sottostanti. (Si memorizza in modo condensato l'hash di un insieme ampio d'informazioni)

- **nTime:** Timestamp che indica quando il blocco è stato creato.
- **nBits:** Contiene il **Difficult target**, ossia stabile quanto difficile è per un miner vincere la gara volta alla creazione di un blocco.
- **nNonce:** Il numero che è stato utilizzato dal miner per vincere la gara e generare il blocco.

nVersion
hashPrevBlock
hashMerkleRoot
nTime
nBits
nNonce

Figure 69: Struttura Blocco

Abbiamo detto che il blocco contiene al suo interno una **lista di transazione**. Ma qual'è la dimensione di tale lista? Solitamente è quasi 1000 volte più grande della dimensione dell'header del medesimo blocco. Inoltre la prima transazione è detta **coinbase**:

- Transazione che codifica il trasferimento della ricompensa più le fee per la transazioni
- Non consuma denaro ma la moneta coinvolta è un bitcoin appena creato dal nulla.
- Le fee invece consumano denaro.
- L'output è l'indirizzo o gli indirizzi del miner che ha vinto la gara.

12.1.4 Proof of Work (PoW)

Proof of Work è un meccanismo che permette di determinare quanta potenza di calcolo è stata utilizzata per un certo intervallo di tempo da parte di un entità. Il calcolo si basa su un **problema di crittografia** che **può essere risolto** e che **richiede un significativo sforzo lavorativo** (Calcolo computazionale).

Problema difficile da risolvere ma facile da verificare che sia stato svolto correttamente

Tale tecnica non è utilizzata (ma neanche inventata) solamente dal mining di Bitcoin, ma è utilizzata anche in altri contesti come:

- Scoraggiare gli attacchi di tipo DOS:

In particolare è concesso agli utenti di accedere a un determinato servizio, solo se prima risolvono un problema computazionale. Quindi se prima di accedere a una risorsa, devo utilizzare la cpu per un determinato intervallo di tempo (che è piccolo, ma sufficiente), mi è impossibile effettuare un numero elevato di richieste di accesso alla risorsa contemporaneamente.

Si osservi che deve risultare difficile per il richiedente svolgere il lavoro e facile per il provider verificare che il lavoro sia stato svolto veramente. Scorruggiare lo spamming delle e-mail: In particolare per ogni messaggio spedito vi è un costo computazionale.

Un messaggio di posta necessita di un francobollo per essere spedito, analogamente un messaggio di posta digitale necessita l'impiego di cicli di cpu.

Da ciò ne consegue che per una media di 15/20 email al giorno, non vi è alcun tipo di problema ma se si effettua mediamente un invio di 1000/10000 email, quella piccola quantità di lavoro richiesta per ogni invio risulterà debilitante.

Nello scenario del mining di bitcoin la PoW è così effettuata:

- Viene assegnata una **challenge string c**, che è l'intestazione di un blocco contenente un insieme di transazioni.
- Colui che deve dimostrare la propria attività, deve risolvere un determinato problema. In particolare deve determinare una **nonce stringa x** che concatenata alla **stringa c** soddisfi la seguente proprietà:

x tale che $x + c$ (Concatenazione) sottoposto a hashing genera un valore di hash caratterizzato da n zeri all'inizio. Ove $n \geq$ numero previsto di 0 (numero strettamente legato al valore di difficoltà della challenge affrontata).

È importante osservare che è impossibile applicare una strategia che consenta di costruire un hash specifico che soddisfi determinate proprietà, e quindi l'unica via possibile per determinare x è effettuare un approccio brute force, ossia per tentativi.

I parametri in gioco sono i seguenti:

- **difficulty - d**: Un numero positivo che il sistema può modificare e dalla quale dipende il numero di zeri previsti.
- **challenge - c**: La stringa data in input
- **nonce - x**: La stringa da determinare per risolvere il problema

Si parla quindi di una funzione *PoW* così costituita:

$$F_d(c, x) \rightarrow \{true, false\}$$

- d e c sono conosciuta a priori
- $F_d(c, x)$ è facile da determinare se d, c e x si conoscono. Viceversa determinare x in modo tale che $F_d(c, x) = true$ non è immediato.

Più specificatamente in bitcoin la funzione di *PoW* è la seguente:

$$SHA256(SHA256(c|x)) < \frac{2^{224}}{d}$$

Intuitivamente all'aumentare di d si ottiene un numero più piccolo, quindi costituito da un maggiore numero di zeri, o meglio da un numero maggiore di bit iniziali pari a 0. Questo numero di bit, detto *nBits* è un parametro presente nell'intestazione di ogni blocco e di cui abbiamo già discusso precedentemente. Si osservi Fig 69

Proof of Work: An example

- $c = "Hello, world!"$ and $\text{nonce } x = 0$ then (output is in hexadecimal)

$h("Hello, world! 0") = 1312Af17Bc253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64$

- first try with $x = 0$ is a failure: the output does not begin with any zeroes at all
- try nounce $c = "Hello, world!"$ and $\text{nonce } x = 1$

$h("Hello, world! 1") = e9afc424b79e4f6ab42d99c81156d3a17228d6e1ee4139be78e948a9332a7d8$

- second try with $x = 1$ is a failure
- keep trying different values for the nonce, $x = 2, 3, \dots$ finally, at $x = 4250$

$h("Hello, world! 4250") = 0000c3af42fc31103fffdc015ffa747ff87349a4714df7cc52ea464e12dcd4e9$

success if the proof of work requires 4 leading zeros !

Figure 70: Proof of Work: An example

Ma qual è il beneficio di rendere costoso il proporre nuovi blocchi all'interno dei blockchain? Il beneficio è che la validazione non è influenzata dal numero di partecipanti, ma necessità di potenza di calcolo e anche nel caso in cui si volesse effettuare una qualsivoglia di "azione" illecita, la quantità di denaro necessaria per ottenere un "vantaggio" è maggiore a ciò che si può ricavare dal vantaggio stesso. (detto in parole semplici, attacco sybil economicamente irrealizzabile).

12.1.5 Miner tasks

Riepilogando, ciò che si è potuto intuire dai paragrafi precedenti è che il miner essenzialmente è un programma che viene eseguito in un data center e svolge le seguenti attività:

- Verifica le transazioni in ingresso sulla base delle firme. (In sostanza chiave pubblica e privata, non abbiamo visto tutti i dettagli del protocollo e ci accontentiamo così).
- Crea un blocco di transazioni, contenente transazioni valide eseguendo il processo di mining:
 - Trovare il nounce x (tramite la procedura descritta precedentemente) e scrivere il medesimo nel campo **nNonce**
 - Informa gli altri nodi presenti nella rete e aspetta che il nuovo blocco sia accettato anche dagli altri nodi. Difatti può accadere che qualche altro nodo sia arrivato alla soluzione prima del nodo corrente.
 - Se il blocco è accettato, allora il nodo corrente risulterà il vincitore della task.

Ma cosa succede nello specifico quando si riceve un nuovo blocco?

Come detto in precedenza, gli altri nodi, una volta ricevuto il nuovo blocco, verificano la correttezza di quest'ultimo visualizzando il campo *nNonce* nell'intestazione del blocco (E quindi verificano che x risolva il problema computazionale)

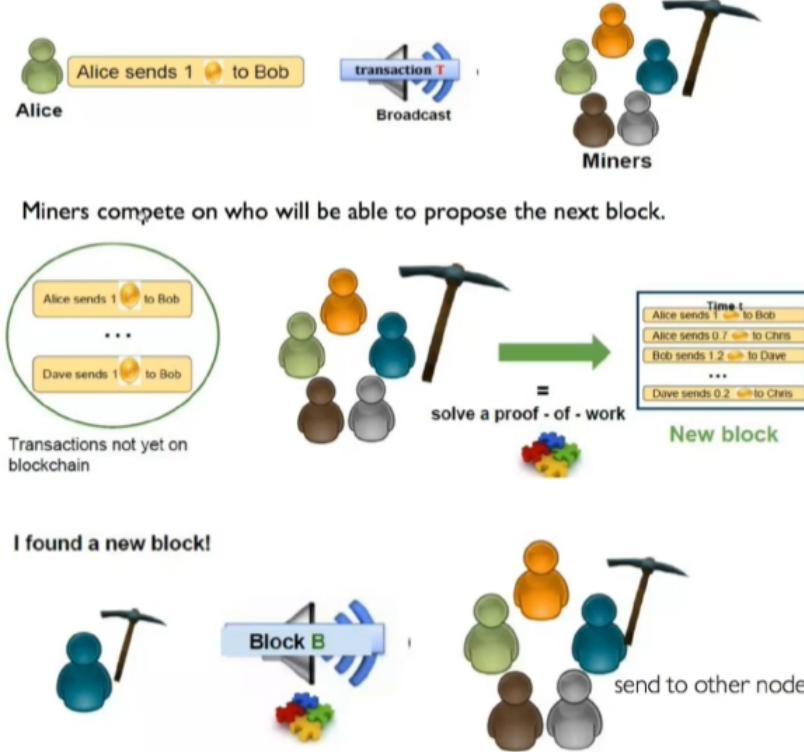


Figure 71: Recap generale

Se il blocco è corretto, allora il processo di mining viene interrotto, il blocco viene aggiunto in posizione N e viene inizializzato un processo di mining per il blocco in altezza $N + 1$.

Si assume ovviamente che ogni nodo partecipante alla rete abbia la medesima copia della blockchain con ultimo blocco in altezza $N - 1$.

Nel caso in cui il blocco è ricevuto da un nodo che non effettuando un processo di mining, viene eseguita solo l'add del nuovo blocco alla copia locale della blockchain.

12.1.6 How to choose the difficulty

La difficoltà è un parametro che viene di volta in volta modificata. Questo accade principalmente perché con il passare del tempo i computer diventano sempre più veloci e performanti. In particolare si ha una regola costante sulla base del protocollo Bitcoin ossia:

Mediamente ogni 10 minuti vi deve essere la creazione di un nuovo blocco all'interno della blockchain

Essendo che la potenza di calcolo della rete aumenta continuamente (I miners possono entrare quando vogliono all'interno della rete), si ha che l'intervallo di tempo per risolvere il problema computazionale e creare un nuovo blocco diminuisce drasticamente. Allora per bilanciare la frequenza con cui avviene il soddisfacimento delle mining tasks, viene aumentata la difficoltà del problema computazionale.

12.1.7 Incentives: Block rewards

Perché qualcuno dovrebbe fare il miner? L'abbiamo ripetuto centinaia di volte, vista il servizio offerto nel creare un nuovo blocco all'interno della blockchain, il miner viene ricompensato con **nuove** monete bitcoins. Questo inoltre è l'unico metodo per creare nuovi bitcoins.

- L'hardware utilizzato per il mining, è caratterizzato da GPU molto performanti, visto che l'hashing è molto più efficiente se svolto da un Graphics process unit.

12.1.8 Blockchains forks

Può succedere alcune volte che due miners effettuano la creazione di un nuovo blocco da aggiungere all'interno della blockchain, quasi simultaneamente. In questo caso accade una biforcazione, ossia ambedue i blocchi vengono appesi ad altezza N e contengono il medesimo *hashPointer* allo stesso blocco precedente. Inoltre ambedue i blocchi vengono considerati "validi", poiché potenzialmente creati da miner onesti i quali hanno seguito correttamente il protocollo.

Tuttavia la rete necessita di convergere a una singola versione della blockchain (senza biforazioni per capirci meglio). Difatti tale tipo di fork è detta soft-fork vista la sua natura casuale, e non dura a lungo. In particolare vige una regola ossia:

Creato un nuovo blocco, quest'ultimo viene appeso al branch (Ramo) più lungo

Da ciò ne consegue che i rami potrebbero "vivere" per sempre solo se venissero estesi simultaneamente in maniera indefinita. Altamente improbabile o quasi impossibile.

13 Lezione n13 19 05 2022

13.1 Livello di trasporto

Rivediamo brevemente ciò che sappiamo del livello trasporto. Un protocollo a livello trasporto mette a disposizione una **comunicazione logica** tra processi applicativi di host differenti, ove per **comunicazione logica** si intende:

Dal punto di vista dell'applicazione, tutto procede come se host, che eseguono i processi, fossero direttamente connessi;

In particolare si ha che i processi applicativi utilizzano la comunicazione logica fornita dal livello di trasporto per scambiare messaggi, senza preoccuparsi dei dettagli dell'infrastruttura fisica utilizzata.

- **Lato mittente:** Il livello trasporto converte i messaggi che riceve da un processo applicativo in pacchetti a livello di trasporto, noti anche come **segmenti**. Quest'ultimo passa poi al livello di rete ed è bene osservare che i router intermedi agiscono solo i campi del pacchetto al livello di rete (datagramma).
- **Lato ricevente:** Il livello di rete estraie il segmento dal datagramma e lo passa al livello superiore, ossia il livello trasporto. Quest'ultimo elabora il segmento ricevuto, rendendo disponibili all'applicazione destinataria i dati del segmento.

È possibile mettere a disposizione delle applicazioni di rete, più di un protocollo a livello di trasporto, per esempio Internet ne possiede due:

- TCP e UDP, il cui principale compito è l'estensione del servizio di consegna IP tra sistemi periferici a quello di consegna tra **processi in esecuzione sui sistemi periferici**. Tale passaggio da consegna *host – to – host* a consegna *process – to – process*, viene detto **multiplexing e demultiplexing a livello di trasporto**.
 - **Demultiplexing nell'host ricevente:** Consegnà i segmenti ricevuti al socket appropriato
 - **Multiplexing nell'host mittente:** Raccoglie i dati da vari socket e li incapsula con l'intestazione (utilizzata poi per il demultiplexing).
- Trasporto orientato alla connessione: TCP
 - Struttura dei segmenti, Trasferimento affidabile (corretto e ordinato), Controllo di flusso, gestione della connessione. Converte quindi il servizio inaffidabile tra sistemi periferici di IP, in un servizio affidabile di trasporto dati tra processi.
- Trasporto senza connessione: UDP
 - Servizio inaffidabile, non garantisce che i dati inviati arrivino intatti e neppure che arrivino al destinatario. Controllo degli errori.

13.1.1 Come funziona il demultiplexing

- **Demultiplexing:** L'host riceve i datagrammi IP, ognuno dei quali caratterizzato da un **Indirizzo IP di origine** e un **Indirizzo IP di destinazione**. Inoltre ogni datagramma altro non trasporta che un segmento di livello trasporto, il quale all'interno detiene l'informazione riguardo il **numero di porta di origine** e **numero di porta di destinazione**.

Quindi l'host usa gli **Indirizzi IP** e i **numeri di porta** per inviare il segmento alla socket appropriata.



Figure 72: Struttura del segmento TCP/UDP

- **Demultiplexing senza connessione:** Si ha che il socket UDP è identificata da 2 parametri: Indirizzo IP di destinazione e numero della porta di destinazione.

Quando l'host riceve il segmento UDP vi è un controllo del numero della porta di destinazione nel segmento e si effettua lo smistamento del segmento UDP al socket con quel numero di porta.

Da ciò ne conseguono i Datagrammi IP con indirizzi IP di origine e/o numeri di porta di origine differenti vengono inviati allo stesso socket.

- **Demultiplexing orientato alla connessione:** Il socket TCP è identificato da 4 parametri: Indirizzo Ip di Origine/Destinazione e numero di porta di origine/Destinazione.

L'host ricevente utilizza tutti e quattro i parametri per inviare il segmento al socket appropriato.

Si osservi che un host server può supportare più socket TCP contemporanei, ove ogni socket è identificato dai suoi 4 parametri. Inoltre i server web hanno socket differenti per ogni connessione client e in HTTP non-persistente si avrà un socket differente per ogni richiesta.

13.1.2 Trasporto senza connessione UDP

UDP è un protocollo di trasporto **Senza connessione** caratterizzato da un servizio di consegna a "massimo sforzo" poiché i segmenti UDP possono essere perduti o consegnati fuori sequenza all'applicazione (scartati).

Senza connessione: no handshaking tra mittente e destinatario UDP; Ogni segmento UDP è gestito indipendentemente dagli altri

Ma allora perché esiste UDP? Poiché risulta semplice, ossia non vi è nessuno stato di connessione nel mittente e destinatario, i segmenti sono caratterizzate da intestazioni corte e non vi è alcun

controllo di congestione e ciò permette d'inviare una moltitudine di dati contemporaneamente (A RAFFIFCA). Inoltre alla base non vi è alcuna connessione stabilita (che potrebbe aggiungere ritardo).

UDP viene utilizzato spesso nelle applicazioni multimediali:

- Tollera piccole perdite
- Sensibile alla frequenza

Ma viene utilizzato sia in DNS o SNMP.

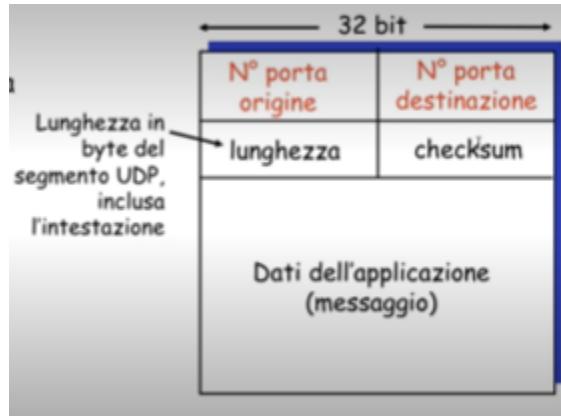


Figure 73: Struttura del segmento UDP

Si osservi che può essere ottenuto un certo grado di affidabilità in UDP, ossia implementando il recupero degli errori delle applicazioni (Aumentiamo quindi l'affidabilità del livello applicazione). La **Checksum UDP** ha lo scopo di rilevare i bit alterati (errori) nel segmento trasmesso. In particolare si ha:

- **Lato mittente**: Tratta il contenuto del segmento come una sequenza di interi da 16 bit e somma tali contenuti del segmento (Effettua complemento a 1). Successivamente pone il valore della checksum nel campo checksum del segmento UDP.
- **Lato ricevente**: Calcola la checksum del segmento ricevuto. Se la checksum calcolata è uguale al valore del campo checksum allora nessuno errore rilevato (Ma più avanti capiremo che ciò non è totalmente garantito). Altrimenti vi è un errore.

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
a capo																
somma	1	0	1	1	1	0	1	1	1	0	1	1	1	1	1	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Figure 74: Esempio di checksum

Quest'anno è stata eliminata la descrizione di come effettuare tramite automi a stati finiti a livello trasporto un trasporto affidabile. Ossia che il mezzo inaffidabile (Livello rete) sia trasformato in un mezzo affidabile, che in presenza di errori tramite la ritrasmissione garantisca sempre la consegna dei pacchetti nell'ordine e nei pacchetti. Ma la descrive in linea di massima, mannaggia a lui. (Te se vuole bene paolone).

Generalmente il livello di trasporto offre al livello soprastante due metodi:

- **rdt_send** : Trasferimento dati affidabile lato mittente.
- **deliver_data**: Che semplicemente quando riceve i dati dal livello applicazione passa i dati allo strato superiore.

Questi due metodi costituiscono quella che è l'interfaccia del livello applicazione, implementati tramite due protocolli distinti: **Protocollo di invio** e **Protocollo di ricezione**. Che rappresentano il **protocollo TCP** (in internet). Tali due protocolli, a loro volta interagiscono con il livello di rete, che può essere visto come un canale inaffidabile, che può quindi perdere contenuto.

L'interfaccia tra il livello rete e il livello trasporto è caratterizzata da due metodi:

- **udt_Send**: metodo tramite la quale si invia il pacchetto sul canale non reliable
- **rdt_rcv**: metodo tramite la quale si riceve il pacchetto dal canale unreliable.

Quindi per rendere il sistema di trasporto affidabile, dobbiamo configurare ad hoc il protocollo di trasferimento lato invio, il protocollo di trasferimento lato ricezione e i quattro metodi descritti precedentemente.

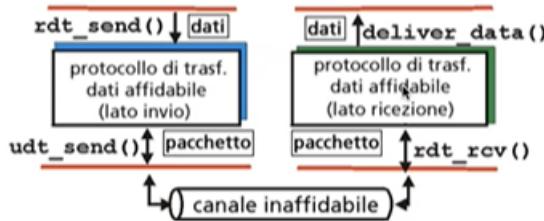


Figure 75: Struttura

13.2 Protocolli con pipeline

Il protocollo TCP non può essere confrontato direttamente con i protocolli che andremo a vedere nelle prossime righe (GoBackN e SelectiveRepeat), poiché il suo funzionamento non si basa sui pacchetti, ma ragiona in termini di flusso di byte, anche se è stato sviluppato prendendo i vantaggi dell'uno e dell'altro.

- **GoBackN**: Il mittente può inviare fino a N pacchetti senza ACK in pipeline.
Il ricevente invia solo ACK cumulativi (Se c'è un gap l'ack di un pacchetto non viene trasmesso).
Inoltre si osservi che il mittente ha un timer associato al più vecchio pacchetto senza ack. Se il timer scade, ritrasmette tutti i pacchetti senza ACK .
- **Selective-Repeat**: Il mittente può avere fino a N pacchetti senza ACK in pipeline.
Il ricevente associa l' ACK solo ai singoli pacchetti.
Il mittente associa un timer a ciascun pacchetto che non ancora ricevuto l' ACK . Quando il timer scade, ritrasmette solo i pacchetti che non hanno avuto ACK.

Tali due protocolli riescono a risolvere il problema della "lentezza" che caratterizza i protocolli Stop-And-Wait, tramite l'implementazione della pipeline.

13.2.1 GoBackN

- **Mittente:**

A ogni pacchetto è associato un numero di sequenza a k bit memorizzato nell'intestazione del pacchetto (Usato ovviamente in modo ciclico, ovvero che si resetta).

La pipeline è implementata tramite il concetto di "Finestra" che può contenere fino a N pacchetti consecutivi non riscontrati. Tale finestra si sposta in avanti ogni qualvolta è ricevuto

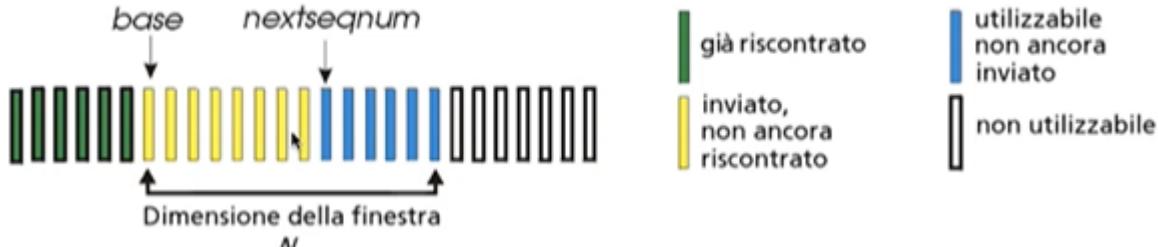


Figure 76: Struttura finestra

un riscontro cumulativo, ove riscontro cumulativo non significa che invio un riscontro ogni tot pacchetti, ma invio un riscontro per ogni pacchetto se e solo se ho già ricevuto regolarmente anche i precedenti. Formalmente si ha che $Ack(n)$ è riscontro di tutti i pacchetti con numero di sequenza minore o uguale ad n ed eventuali pacchetti duplicati potrebbero essere scartati. Ovviamente come già detto in precedenza ad ogni pacchetto in transito è associato un timer e se avviene il $timeout(n)$ si effettua la ritrasmissione del pacchetto n e di tutti i pacchetti, con i numeri di sequenza più grandi, contenuti nella finestra. Si osservi che nel GoBackN

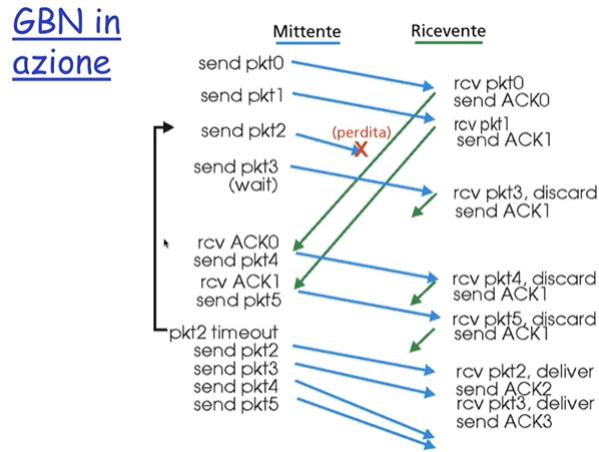


Figure 77: GBN in action

la finestra ricevente può essere implementata, ma solo se si vuole effettuare un controllo di flusso.

13.2.2 Selective Repeat

Il **ricevente** invia i riscontri specifici per tutti i pacchetti ricevuti correttamente. Necessità dell'implementazione di buffer dei pacchetti che eventualmente può garantire consegne in sequenza al livello superiore. Il **mittente** ritrasmette allo scadere del timer i pacchetti per i quali non ha

ricevuto un ACK.

Per meglio capire il funzionamento del Selective Repeat, introduciamo il concetto di **Finestre del mittente e del ricevente**.

- **Finestra del mittente:** Lato mittente, esiste ovviamente la finestra di lunghezza N . Tuttavia i due insiemi "pacchetti inviati e riscontrati" e "pacchetti inviati ma non riscontrati" non sono due gruppi consecutivi, ma possono essere mischiati tra loro (Un mix insomma). In altri termini si possono avere, nella finestra degli invii, in maniera alternata "pacchetto inviato e riscontrato" e "pacchetto inviato e non riscontrato". (Si osserva meglio dall'esempio). Questo può accadere perché il riscontro non è cumulativo ma è selettivo.
- **Finestra ricevente:** Si ha la necessità d'implementare un buffer, nel quale memorizzare i "pacchetti ricevuti ma non ordinati".
Infatti, mentre nel go-back-n ogni pacchetto riscontrato può essere già trasmesso al livello soprastante, perché ordinato per definizione, nel selective repeat ogni pacchetto riscontrato può essere trasmesso al livello soprastante se e solo se anche tutti pacchetti precedenti che sono nel buffer sono riscontrati. In parole più semplice se si ha una sequenza di "Pacchetti bufferizzati e già riscontrati" che parte dalla base della finestra, tale sequenza può essere passata al livello applicazione, spostando la finestra in avanti.

I CAMPI DELL'INTESTAZIONE TCP NON VANNO IMPARATI TUTTI, ovviamente si chiedono di conoscere quelli principali. (parole del paolone).

13.3 Trasporto orientato alla connessione TCP

13.3.1 Panoramica

Il TCP è punto-punto, ovvero coinvolge un mittente e un destinatario. Si basa sul flusso di byte in sequenza (non pacchetti) e anche qui vi è l'applicazione della pipeline la quale dimensione è definita dal controllo di flusso e di congestione TCP. Si ha un buffer d'invio e di ricezione.

Inoltre il TCP è Full-Duplex, ossia si ha un flusso di dati bidirezionali nella stessa connessione e con MSS si indica la dimensione massima di segmento.

È un protocollo orientato alla connessione, quindi viene effettuato l'handshaking (tre fasi che permettono lo scambio di messaggi di controllo) per inizializzare lo stato del mittente e del destinatario prima di scambiare dati.

Il flusso è controllato, ossia il mittente non sovraccarica il destinatario.

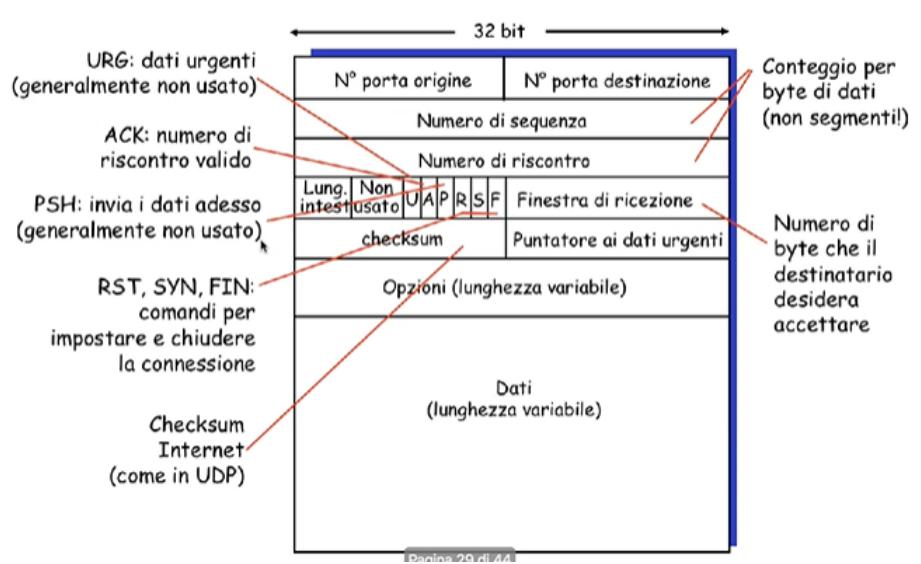


Figure 78: Struttura dei segmenti TCP

I **Numeri di sequenza** sono rappresentati dal primo byte del segmento nel flusso di byte e l'ACK cumulativo ed è associato di volta in volta al numero di sequenza del prossimo byte atteso dall'altro lato.

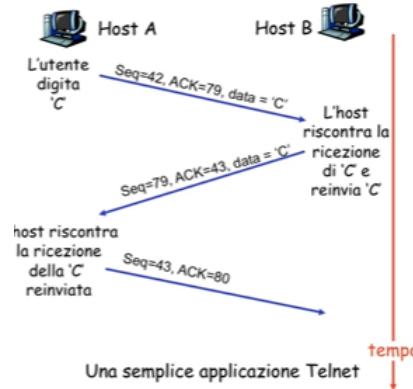


Figure 79: Esempio

Si osservi che il modo in cui il destinatario gestisce i blocchi fuori sequenza, non viene specificato da TCP, ma dipende strettamente dall'implementazione.

13.3.2 Tempo di andata e ritorno e timeout

Il problema fondamentale di tutti i protocolli basati su timeout è stimare il timeout. Se troppo piccolo infatti causeremmo delle ritrasmissioni non necessarie, se invece troppo grande avremmo una reazione troppo lenta alla perdita dei segmenti. Una scelta potrebbe essere stimare RTT (Round-Trip-Time):

- **SampleRTT:** Tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK (ignora le ritrasmissioni). Tuttavia SampleRTT varia, quindi occorre una stima più levigata,

ossia una media di più misure recenti.

- **EstimatedRTT**: Media mobile esponenziale ponderata. Quindi l'influenza dei vecchi campioni decresce esponenzialmente.

$$(1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$$

13.3.3 Trasferimento dati affidabile

TCP crea un servizio di trasferimento dati affidabile sul servizio inaffidabile di IP, tramite l'utilizzo di Pipeline dei segmenti, ACK accumulativi e un solo timer di ritrasmissione. Le ritrasmissioni sono avviate da venti di timeout e Ack Duplicati.

Negli esempi successivi si considera un mittente TCP semplificato, ossia vengono ignorati gli ACK duplicati, il controllo di flusso e il controllo di congestione.

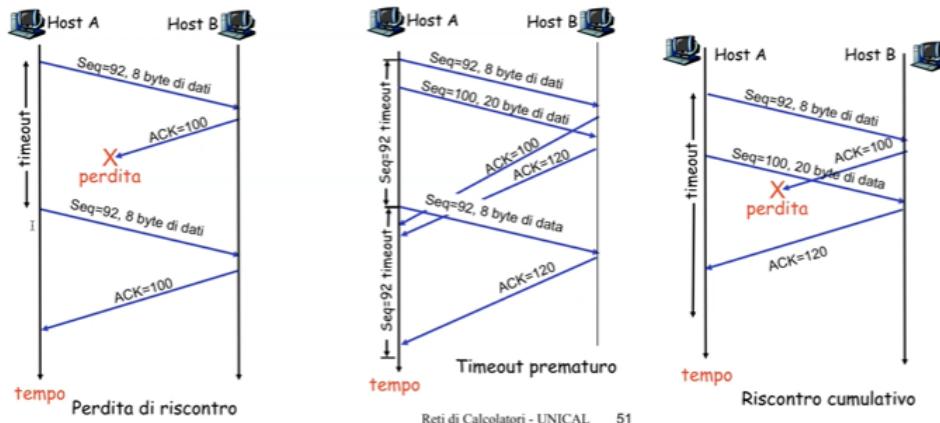


Figure 80: Esempio

Per quanto concerne il **Controllo di flusso** il lato ricevente della connessione TCP, presenta un buffer di ricezione. Lo scopo del mittente è non sovraccaricare il buffer del destinatario trasmettendo troppi dati, troppo velocemente. In particolare si deve tener conto della velocità con cui il processo applicativo legge dal buffer.

Si ha quindi che la frequenza d'invio deve corrispondere alla frequenza di lettura dell'applicazione ricevente. (**Servizio di corrispondenza delle velocità**)

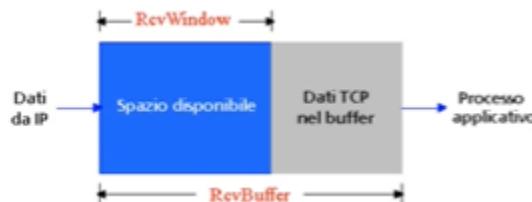


Figure 81: Struttura buffer

Per il funzionamento del controllo del flusso si ha che il mittente comunica lo spazio disponibile includendo *RcvWindow* nei segmenti. Così facendo, il mittente limita i dati non riscontrati a *RcvWindow* e si garantisce che il buffer di ricezione non vada in overflow.

13.3.4 Gestione della connessione TCP

Si ricorda che mittente e destinatario TCP stabiliscono una "connessione" prima di scambiare i segmenti dati. In particolare vengono inizializzate le variabili TCP (Numeri di sequenza, buffer e informazioni per il controllo di flusso) e successivamente il client avvia la connessione e il server aspetta di essere contattato dal client.

Si basa su un HandShake a tre vie:

- **Passo 1:** Il client invia un SYN al server specificando il numero di sequenza iniziale. (Nessun dato)
- **Passo 2:** Il server riceve un Syn e risponde con un segmento SYNACK, viene allocato un buffer e specifica il proprio numero di sequenza iniziale.
- **Passo 3:** Il client riceve SYNACK e risponde con un segmento ACK che può contenere dati.

L'opposto dell'apertura della connessione è la **chiusura della connessione**:

1. Passo 1: Il client invia un segmento di controllo FIN al server.
2. Passo 2: il server riceve il segmento FIN e risponde con un ACK. Chiude la connessione e invia FIN.
3. Passo 3: Il client riceve FIN e risponde con un ACK. Inizia l'attesa temporizzata.
4. Passo 4: Il server riceve un ACK e la connessione viene chiusa.

SI osservi che con qualche piccola modifica si possono gestire più segmenti FIN contemporaneamente

L'attesa temporizzata indica un intervallo di tempo necessario affinché le risorse appena deallocate non vengano immediatamente riallocate.

14 Lezione n14 26 05 2022

14.1 Livello di rete

Le funzioni chiavi del livello rete sono le seguenti

- Forwarding (inoltro): Trasferisce i pacchetti dall'input di un router all'output del router appropriato.
- Routing (instradamento): Determina il percorso seguito dai pacchetti dall'origine alla destinazione . Generalmente effettuato tramite algoritmi di'instradamento.

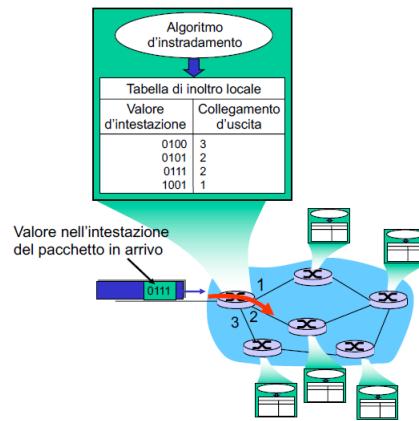
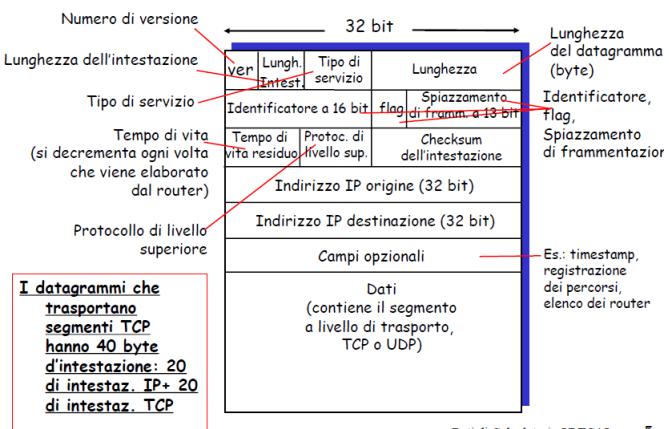


Figure 82: Esempio

Nel livello di reti sono contenuti tutti i **protocolli d'instradamento** tramite la quale gestire le tabelle d'inoltro, il **Protocollo IP** che indica le convenzioni di indirizzamento, il formato dei datagrammi e le convenzioni riguardante la manipolazione dei pacchetti e in fine vi è il **Protocollo ICMP** necessario per la notifica di errori e segnalazioni del router.

In particolare il formato dei datagrammi è il seguente:



Reti di Calcolatori - UNICAL 5

Figure 83: Formato dei datagrammi

Si osservi che l'unità massima di trasmissione (MTU) indica la massima di quantità di dati che un frame a livello di collegamento può trasportare (MTU varia al variare del tipo di link)

utilizzato). E per tale motivo i Datagrammi IP voluminosi vengono "frammentati" in datagrammi IP più piccoli. Tali frammenti verranno riassemblati solo una volta raggiunta la destinazione. Per identificare e assemblare correttamente un datagramma IP "frammentato" vengono utilizzati i bit dell'intestazione IP, nello specifico i campi flag e spiazzamento.

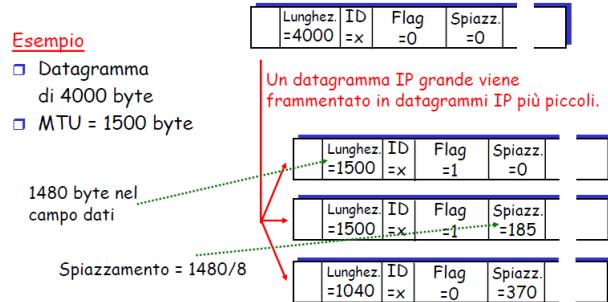


Figure 84: Esempio frammentazione e riassemblaggio

14.1.1 Indirizzamento IPv4

Ogni interfaccia di host e router di Internet ha un **Indirizzo IP** globalmente univoco a 32 bit. I bit di alto ordine indicano la parte di sottorete (In internet sono anche chiamate reti IP) e i bit di basso ordine invece la parte dell'host. (Il numero di bit dedicate all'una e all'altra parte è variabile)

DEFINIZIONI:

- L'**Interfaccia** è il confine tra host e collegamento fisico. Un host in genere ha una sola interfaccia, invece il router deve necessariamente essere connesso ad almeno due collegamenti (Due interfacce).
- È detta **sottorete** una rete isolata i cui punti terminali sono collegati all'interfaccia di un host o di un router.

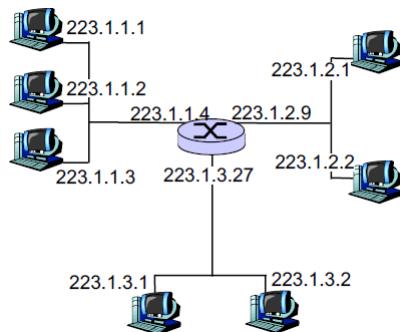


Figure 85: Esempio

Ma cosa bisogna fare per assegnare un indirizzo IP a un host? Vi sono essenzialmente due vie, una manuale e una svolta in automatico. La configurazione manuale avviene tramite la modifica delle impostazioni della macchina, invece la configurazione automatica viene svolta dal DHCP.

Il **DHCP** è il Dynamic Host Configuration Protocol che permette a un host di ottenere un Indirizzo IP in maniera dinamica "plug and play". Ciò consente il riuso degli indirizzi, supportando

soprattutto gli utenti mobili che si uniscono alla rete.

DHCP si basa essenzialmente su 4 passi:

1. **DHCP discover**: L'host invia un messaggio broadcast.
2. **DHCP offer** risposta da parte del server DHCP
3. **DHCP request** richiesta di un indirizzo IP da parte dell'host
4. **DHCP ack** invio dell'indirizzo da parte del sever DHCP

Inoltre per ridurre lo spreco di indirizzi pubblici vi è l'utilizzo di classi di indirizzi IPv4 riservate alle reti locali (Intranet). Ma c'è da sottolineare che i pacchetti con indirizzi IP privati (di sorgente/destinazione) non possono viaggiare su internet perchè si tratta di indirizzi IP non routabili.

14.1.2 Traduzione degli indirizzi di rete (NAT)

Per ovviare alla scarsità di indirizzi pubblici disponibili si è implementato il NAT, ossia il network address translation (traduzione degli indirizzi di rete). Il NAT è una tecnica che consiste nel modificare gli indirizzi IP contenuti negli header dei pacchetti in transito su un sistema che agisce da router all'interno di una comunicazione tra due o più host.

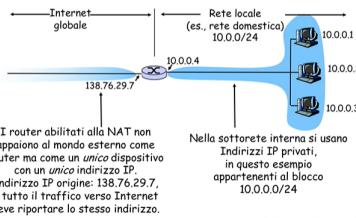


Figure 86: Esempio

Si ha che il router abilitato alla NAT nasconde i dettagli della rete domestica al mondo esterno, in tal modo per tutte le macchine di una rete locale è sufficiente un unico indirizzo IP.

Inoltre è possibile modificare gli indirizzi delle macchine di una rete privata senza doverlo comunicare all'Internet globale, oppure vale anche il viceversa, ossia è possibile modificare ISP senza modificare gli indirizzi delle macchine della rete privata.

Il Nat rappresenta anche un plus per la sicurezza perchè i dispositivi interni alla rete non sono esplicitamente indirizzabili e visibili dal mondo esterno.

- **Implementazione:** Quando un router NAT riceve un datagramma da inviare da un router della rete privata, cambia l'indirizzo origine del datagramma con il proprio indirizzo IP lato WAN (il pubblico) e sostituisce il numero di porta d'origine (Identificatore dell'host) con un nuovo numero di porta generato. (Inserisce un record nella tabelle di traduzione NAT).

Si osservi che il campo di numero di porta è lungo 16 bit, poiché NAT nacque per utilizzare un unico indirizzo con dietro le quinte un numero elevatissimo di HOST.

Tuttavia nat è contestato da alcuni in quanto:

1. I router dovrebbero elaborare i pacchetti solo fino al livello 3.
2. Viola il cosiddetto argomento end-to-end : gli host dovrebbero comunicare direttamente senza intromissione di nodi né modifica di indirizzi IP e di numeri di porta

3. Interferenza con le applicazioni P2P, a meno che non sia specificamente configurato per quella specifica applicazione P2P
4. Per risolvere la scarsità di indirizzi IP si dovrebbe usare IPv6

14.1.3 Port forwarding

La funzionalità del NAT descritta nelle slide precedenti permette agli host della rete locale, muniti di un indirizzo IP privato, di agire come client verso server che si trovano su Internet. Tramite il **port forwarding**, invece, è possibile fare in modo che un host della rete locale agisca da server per client che si trovano all'esterno.

La configurazione del port forwarding dipende dallo specifico router NAT utilizzato, ma i passi da seguire sono sostanzialmente gli stessi. Difatti occorre configurare il router in modo tale che all'host accessibile dall'esterno venga assegnato un indirizzo statico (e ovviamente privato) e che ogni richiesta ricevuta sull'indirizzo lato WAN con porta 8080 (Porta scelta arbitrariamente) sia inoltrata all'indirizzo statico scelto con numero di porta dipendente dal servizio presente sull'host (Porta scelta arbitrariamente).

Per assegnare un indirizzo statico ad un host è possibile configurare tale host in modo che usi un indirizzo specifico non utilizzato da altri host e fuori dal range usato dal DHCP. Oppure configurare DHCP in modo che assegni sempre lo stesso indirizzo IP ad un determinato MAC (Il mac Identifica l'host).

Ora però per evitare di dover comunicare ai client l'indirizzo IP pubblico del router che può variare (perchè non sempre è statico), è possibile utilizzare un servizio di Dynamic DNS. Registrandosi infatti a un servizio di Dynamic DNS, si può scegliere un hostname HN, associandolo ad una username e ad una password. Quando il client chiede al DNS di risolvere HN, ottiene automaticamente l'indirizzo IP del router e poi tramite port forwarding accede al servizio attivo sull'host.

Si osservi che è necessario configurare il proprio router in modo che comunichi dinamicamente il proprio indirizzo IP al server specificato dal servizio di Dynamic DNS.

14.1.4 Internet Control Message Protocol (ICMP)

Il protocollo ICMP viene utilizzato da host e router per scambiarsi informazioni a livello di rete riguardo eventuali report degli errori o semplici echo request/reply. In particolare si ha che i **Messaggi ICMP** presentano un campo tipo e un campo codice e contengono l'intestazione e i primi 8 byte del datagramma IP.

ICMP è considerato parte di IP.

ICMP viene utilizzato dal Traceroute. Nello specifico si ha che:

1. Il programma invia una serie di datagrammi IP alla destinazione: Il primo con TTL pari a 1, il secondo con TTL pari a 2 e così via.
2. Quando l'n-esimo datagramma arriverà all'n-esimo router: Il router scarta il datagramma e invia all'origine un messaggio di allerta ICMP (TTL scaduto). Il messaggio include il nome del router e l'indirizzo IP.
3. Quando il messaggio ICMP arriva, l'origine può calcolare RTT.
4. Traceroute lo fa per 3 volte

14.2 Firewall

Un firewall è una combinazione di hardware e software che isola la rete interna di una società dal resto di Internet. Nello specifico, controlla l'accesso tra le risorse interne e il mondo esterno, permettendo ad alcuni pacchetti di passare e bloccandone altri.

Distinguiamo due tipi di firewall:

- **Firewall a filtraggio di pacchetto:**

Operano nello strato di rete, esaminano le intestazioni IP e TCP/UDP ed applicano delle regole di filtraggio basate su:

- Indirizzi IP, Pote TCP o UDP di sorgente/destinazione.
- Tipo di messaggi ICMP
- Segmenti TCP d'inizializzazione della connessione: In particolare il primo segmento di una connessione TCP ha il bit ACK impostato a 0 e tutti gli altri pari ad 1. Quindi filtrando i segmenti in arrivo con ACK pari a 0 si bloccano tutte le connessioni originate all'esterno, mentre sono consentite quelle originate all'interno

Attenzione che l'ordine delle regole impostate è importantissimo, infatti al pacchetto in arrivo viene applicata la prima regola che è compatibile con gli indirizzi sorgente e destinazione del pacchetto. (con R1 R2 R3 si ottiene un risultato diverso da R2 R1 R3).

Esempi sul pdf.

- **Gateway a livello di applicazione:**

Permette di consentire/rifiutare connessioni sulla base di dati specifici dell'applicazione (Accettare/Rifiutare una coppia di username password). E si osservi che più gateway relative a diverse applicazioni, possono risiedere sullo stesso host. Il server di posta ed il Web proxy sono gateway delle applicazioni.

14.3 Attacchi Informatici

Un **cyberattack** è un'operazione, compiuta da singole persone o da organizzazioni, che ha l'obiettivo di colpire reti di computer, singoli dispositivi o infrastrutture informatiche di larga scala, utilizzando tecniche malevole, provenienti generalmente da sorgenti anonime oppure inconsapevoli. Tra gli obiettivi dei cyberattack si possono annoverare il furto di informazioni sensibili, l'alterazione o la distruzione di dati appartenenti a individui e/o organizzazioni, l'alterazione o la distruzione di sistemi informativi pubblici e/o privati, in alcuni casi per compromettere le infrastrutture informatiche di intere nazioni (cyberterrorismo).

Distinguiamo principalmente gli attacchi informatici in:

- **Attacco Interno:** Cyberattack che viene avviato da un'entità interna al perimetro di sicurezza dell'organizzazione il quale ha accesso alle risorse del sistema, ma fa un uso non approvato di tali risorse.
- **Attacco Esterno:** Cyberattack avviato da un'entità esterna al perimetro di sicurezza dell'organizzazione.
- **Attacco attivo:** Cerca di alterare le risorse di un sistema informatico o di compromettere la sua operatività
- **Attacco passivo:** Finalizzato ad apprendere o fare uso di informazioni del sistema, senza tuttavia compromettere la sua operatività

14.3.1 Denial of Service (DoS) e IP spoofing

Gli attachhi di tipo Denial of Service (DoS) tentano di sovraccaricare un server impedendogli di offrire il servizio. Se il traffico in entrata che inonda la vittima proviene da più fonti distribuite, si parla di attacchi di tipo Distributed Denial Of Service (DDos).

Di norma, soprattutto nel caso del DDos, l'attacco viene sferrato da una botnet, ovvero una moltitudine di computer bot o zombie sui quali sia stato precedentemente inoculato un malwer per attacchi DDos.

Un altro tipo di attacco è L'IP spoofing ch è una tecnica spesso utilizzata per effettuare attacchi di tipo Dos o DDos. Consiste essenzialmente nel mettere nel pacchetto IP un indirizzo sorgente falso. L'uso dell'IP spoofing rende difficile identificare l'indirizzo IP del computer che ha iniziato l'attacco.

Descriviamo ora aluni esempi di attacchi Dos/DDos:

- SYN Flooding:

L'attaccante inonda un server di pacchetti SYN (pacchetti di inizio connessione aventi bit SYN=1 e bit ACK=0) aventi indirizzi IP sorgente camuffati.

Il server risponde con pacchetti SYNACK (aventi bit SYN=1 e bit ACK=1) inviati agli host cui corrispondono gli indirizzi IP camuffati

Tali host ovviamente non completano il 3-way handshake, e la connessione non verrà instaurata

Tuttavia il server è costretto a creare i socket e ad allocare le rispettive strutture dati, consumando così memoria e risorse del processore

- UDP Flooding:

E' un tipo di attacco nel quale un elevato numero di pacchetti UDP sono inviati ad un server, con il fine di sovraccaricarlo e impedirgli di elaborare e rispondere alle richieste legittime

Se i pacchetti UDP non sono indirizzati ad una porta attiva, il server risponde con un messaggio ICMP per informare l'inviaente che l'applicazione di destinazione (corrispondente alla porta specificata nel pacchetto UDP) non è raggiungibile

E' possibile proteggere il server con un firewall che filtra i pacchetti UDP indesiderati, ma anche il firewall può sovraccaricarsi nello svolgere il lavoro di filtraggio

- Smurfing:

L'attaccante invia un gran numero di pacchetti ICMP di tipo Echo Request (es. col programma Ping) ad un ampio numero di host

L'indirizzo sorgente di tali richieste è un indirizzo IP camuffato che corrisponde all'host che si vuole attaccare

I pacchetti ICMP di tipo Echo Reply saranno quindi inviati a tale host che sarà quindi sottoposto ad un bombardamento che ne può pregiudicare le funzionalità

L'attacco è ancora peggiore se i pacchetti ICMP sono inviati ad un indirizzo multicast di una rete locale (es. l'indirizzo 150.145.1.255 della rete 150.145.1.0/24): tutti gli host della rete locale risponderanno allo stesso host sotto attacco

Per limitare gli attacchi basati su IP spoofing bisogna assicurarsi che i pacchetti in uscita dalla rete non abbiano indirizzi sorgente diversi da quelli compresi nel range assegnato alla rete stessa. Allo stesso modo bisogna bloccare i pacchetti in entrata che hanno indirizzi sorgenti appartenenti al range della rete interna.

Inoltre bisogna bloccare i pacchetti in entrata aventi indirizzi sorgente appartenenti a reti riservate per il NAT (Ossia indirizzi privati). Gli indirizzi privati infatti sono indirizzi utilizzabili solo all'interno di una rete locale ma non visibili all'esterno.

Per limitare gli attacchi di tipo smurfing, un'altra buona regola è non accettare pacchetti in entrata aventi come indirizzo destinazione un indirizzo multicast e impedire l'accesso di segmenti TCP di inizio connessione (con ACK=0) in modo da consentire solo le connessioni originate all'interno.(Non esiste una simile possibilità con UDP, ecco perché il traffico UDO viene consentito solo per uno specifico set di porte).

In generale è sempre consigliato bloccare il traffico relativi a servizi "pericolosi" (Ad esempio NFS) e consentire il traffico solo a servizi importanti (dns,http,ftp,ssh,smtp ecc).