

RETI DI CALCOLATORI

1

1. INTRODUZIONE

1.1 Computer networks e internet

Computer networking = Il processo di connettere più computer insieme così da poter condividere informazioni

Internet è il più importante e il più diffuso computer network se non probabilmente il più grande sistema di ingegneria mai creato: internet include centinaia di milioni di connessioni (dispositivi, collegamenti fisici, computer, ecc) ed offre centinaia di servizi agli utenti. Esistono anche altre network più piccole (locali o comunque non collegate ad internet).

Negli anni scorsi internet veniva principalmente utilizzato al fine di connettere dispositivi quali desktop pc, workstations and server che immagazzinavano e trasmettevano informazioni quali pagine web e messaggi mail.

Al giorno d'oggi però non solo i computer sono tra loro connessi (laptops, smartphones, tablets, tv, gaming consoles ecc), ma il networking è ora pervasivo. Le connessioni internet sono dappertutto e dispositivi complessi potrebbero avere una propria connessione.

1.2 DISPOSITIVI, COLLEGAMENTI FISICI ED HOST

I **dispositivi ed i collegamenti network** sono delle infrastrutture che permettono agli host di connettersi (router, access point, switch, hub).

Gli **host** sono dispositivi sui quali le applicazioni o programmi vengono eseguiti (Laptop, server, smartphone, pc).

1.3 COMPUTER NETWORKS E LA TEORIA DEI GRAFI

Le computer networks condividono terminologie con la teoria dei grafi: i dispositivi connessi tramite network sono chiamati **nodi**, mentre le connessioni tra i nodi sono chiamate **links di comunicazione o canali**. Una sequenza di nodi e link è chiamata **percorso**. Gli end point o end systems del network che forniscono o utilizzano servizi sono dei nodi speciali chiamati **host**.

Aggiungere descrizione con =

Idealmente gli host sono le foglie della network (spesso non è così), mentre i nodi intermedi sono in genere dispositivi di routing (routers and switch).

1.4 COMUNICAZIONE DATI

Lo scopo del networking è il condividere informazioni: per permettere funzionalità internet su larga scala così come funzionalità network a bassa scala è necessario che due host siano collegati così che possano comunicare dati a distanza. Il termine telecomunicazione significa letteralmente comunicazione a distanza, mentre la parola dato si riferisci all'informazione

presentata in qualsiasi forma che è stata decisa dalle controparti che hanno creato e utilizzato i suddetti dati. La comunicazione dati è il processo che consiste nello scambio dei dati tra due dispositivi mediante una qualche forma di trasmissione intermedia quale un cavo o wireless assicurando un certo grado di affidabilità (i dati vengono ricevuti correttamente) e performance (i dati vengono ricevuti in una quantità di tempo ragionevole).

La comunicazione dati ha 5 componenti fondamentali:

- 1- Il messaggio contiene i dati che si vogliono comunicare
- 2- Il mittente è l'entità che invia il messaggio
- 3- Destinatario è l'entità che riceve il messaggio
- 4- Mezzo è il canale tra il mittente il ricevente tramite cui il messaggio viaggia
- 5- Protocollo è una serie di regole conosciute dal mittente e dal destinatario ed utilizzate per gestire il messaggio

1.5 RAPPRESENTAZIONE E FLUSSO DEI DATI

I dati da scambiare possono essere rappresentati in diverse forme (testo, numeri, immagini, audio, video ecc.).

A seconda del tipo e dello scopo della comunicazione, il flusso dei dati può essere:

- Simplex: monodirezionale
- Half-duplex: bidirezionale (a turnazione)
- Full-duplex: bidirezionale (contemporaneo).

La **velocità di trasmissione** (transmission rate) è la massima quantità di informazioni che un canale può trasmettere ed è misurata in bits/sec (o bytes/sec).

La **larghezza di banda** (bandwidth) è la massima quantità di informazione che un percorso (collegamenti e nodi) può trasmettere misurati in bits/sec o bytes/sec.

La **portata** (throughput) è la quantità corrente di informazioni che un percorso o un collegamento trasmette, anch'esso misurato in bits/sec (o bytes/sec).

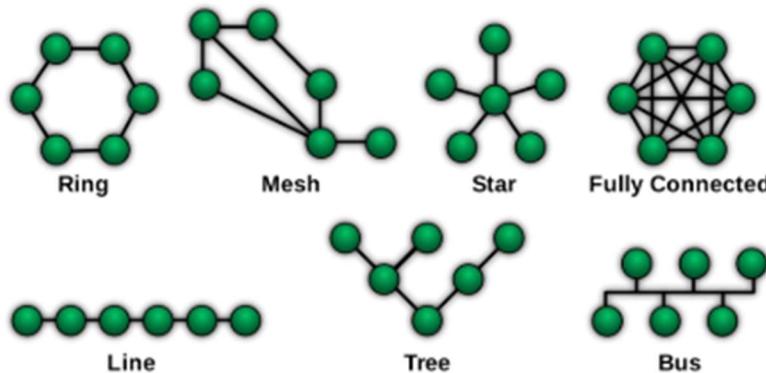
1.6 TIPI DI CONNESSIONI

Gli host di una network possono essere connessi in diversi modi

- Point-to-point: un collegamento tra due dispositivi (wireless o cablato)
- Multipoint (broadcast): più di due dispositivi specifici che condividono un'unica connessione

1.7 TOPOLOGIE NETWORK

La tipologia network è la disposizione degli elementi di un network di comunicazione.



- 1.7.1 Nella tipologia bus gli host sono connessi ad un cavo backbone centrale. I messaggi inviati dai due host generano collisioni.

Pro: Semplice ed economico, ottimo per piccole networks

Contro: singolo punto di guasto (bus rotto) ma le sottoreti possono essere ancora disponibili.

Le connessioni di tipo bus hanno un unico collegamento fisico duplex oppure hanno una backbone con n collegamenti.

- 1.7.2 Nella topologia ring ogni host è connesso mediante point-to point ad esattamente altri due host. Il segnale è inoltrato lungo il ring, da dispositivo a dispositivo fino a quando non raggiunge la destinazione.

Pro: Semplice ed economico, ha prestazioni migliori rispetto alla tipologia bus

Contro: aggiungere nuovi nodi è difficile, il malfunzionamento dei nodi può compromettere la rete

Le connessioni di tipo ring hanno n collegamenti di tipo duplex.

- 1.7.3 Nella topologia star gli host sono collegati ad un controller centrale (hub, switch o router) non vi è una connessione diretta tra gli host.

Il controller centrale direziona i messaggi.

Pro: non molto costoso, semplice, robusto e più scalabile

Contro: Il controller deve essere raggiungibile da tutti gli host, singolo punto di guasto.

Le network di tipo star hanno un controller e n collegamenti duplex.

- 1.7.4 Nella topologia tree (albero) sono integrate diverse tipologie star tipicamente collegate mediante un cavo tipo bus.

Pro: versatile, scalabile e robusto. Ben supportato da fornitori di hardware e software

Contro: difficile da configurare, eredita le debolezze della tipologia bus.

1.7.5 Nella topologia mesh gli host sono connessi mediante point-to-point in una modo non gerarchico.

Full mesh: tutti i nodi sono connessi tra di loro (full-connected)

Partial mesh: i nodi sono connessi solo ad alcuni altri.

Pro: basso livello di traffico, robusto, sicuro e dedicato.

Contro: Difficilmente scalabile, costoso (richiede dispositivi con più porte)

Le networks fullmesh hanno $n(n-1)/2$ connessioni duplex fisiche

1.7.6 Nella topologia ibrida

Le varie topologie sopra citate possono essere meschiate assieme in una network ibrida.

Immagine: star backbone con 3 networks di tipo bus

1.8 CATEGORIE DI NETWORK

Le networks vengono categorizzate per dimensione, numero di host e bandwidth (larghezza di banda).

Local Area Network (LAN) o Wireless Local Area Network (WLAN)

Metropolitan Area Network (MAN)

Wide Area Network (WAN)

1.9 COMPLESSITA' DELLE RETI WANs

La complessità di una tipologia può incrementare con l'aumento delle dimensioni della network. WANs che connettono intere nazioni o continenti possono ovviamente essere molto complesse ed eterogenee.

2 SERVIZI

2.1 INTERNET

Internet è la rete WAN definitiva, che connette tutti i dispositivi nel mondo. Internet è un network di networks. Tecnicamente parlando si tratta di un enorme WAN che consente a sottoreti e dispositivi di essere connessi tra diverse nazioni e continenti.

Breve annotazione storica su internet: L'ARPA (Advanced research project agency) del Dipartimento di difesa degli U.S. ha vinto nel 1969 i contratti per lo sviluppo del progetto ARPANET. Inizialmente ARPANET era principalmente una network chiusa realizzata allo scopo di includere cerchi di ricerca e università. Networks simili vennero poi indipendentemente creati negli US e in Europa.

Nel 1974 vennero realizzati il TCP (Transmission Control Protocol) e il IP (Internet Protocol), due protocolli del Internet Protocol Suite (framework).

Nel 1986 NSF (National Science Foundation) vince i contratti per lo sviluppo del progetto NSFNET, una network basata sui protocolli TCP-IP.

La NSFNET e i protocolli TCP/IP permisero di instradarsi verso l'idea di network di networks. Sia ARPANET e NSFNET avevano delle restrizioni commerciali.

Nel 1990 ARPANET fu dismesso e nel 1995 con la dismissione di NSFNET fu rimossa l'ultima delle restrizioni sull'utilizzo di internet portando internet sulla strada del traffico commerciale.

2.1 INTRODUZIONE

Il principale ruolo delle networks tra computer è quello di permettere lo scambio dei dati tra molteplici dispositivi fra differenti luoghi, ma i raw data sono soltanto un mezzo per raggiungere servizi e risorse fornite dalla network.

Una risorse di una network è un oggetto remoto a cui noi vogliamo accedere (pagine web, storage, computazione, file video audio ecc)

Un servizio di una network è un'azione che un dispositivo remoto esegue per noi (dimmi l'ora corrente, permettimi di inviare una mail)

La differenza tra queste due definizioni è sottile in quanto “fammi vedere un video” o “dammi accesso ad una pagina web” sono azioni che coinvolgono oggetti.

Le entità che offrono servizi su internet sono chiamate service providers.

2.2 INTERNET SERVICE PROVIDERS

L'accesso ad internet è il servizio base. Un internet service provider (ISP) è un'organizzazione che fornisce servizi allo scopo di accedere, usare o partecipare ad internet. Le ISP possono essere organizzate come commerciali, di proprietà della community, no profit o di proprietà privata (compagnie private o università).

Le differenti ISP scambiano dati mediante network (neutral) access point (NAPs) oppure Internet Exchange Point (IXPs).

Una network ISP è gerarchicamente definita come segue:

- Un point of presence (PoP) è un gruppo di uno o più router utilizzati dagli ISP per raggiungere i consumatori.
- ISPs di accesso per le aree locali
- ISPs regionali per aree più vaste
- ISPs nazionali per le nazioni

Un internet exchange point IXP lavora come punto di incontro tra le molteplici ISP ed è tipicamente gestito da terze parti.

2.3 ESEMPIO DI ISPs

Su una scala nazionale ci sono diverse ISP commerciali che forniscono servizi internet a compagnie e a privati. Alcune ISPs italiane sono: Telecom Italia, Fastweb, Vodafone, Tiscali

2.4 CONNESSIONE INTERNET

Le connessioni internet private sono stabilite tramite Modem abilitate tramite network telefoniche.

- Analogic (56kbps)
- Integrated Services Digital Network ISDN (128kbps)
- Asymmetric Digital Subscriber Line ADSL (1Mbps a 20 Mbps)
- Twisted-Pair Copper Wire (10Mbps a 100Mbps)
- Fibra Ottica (50 Mbps a 40 Gbps)

Le compagnie specialmente le medio-grandi possono avere una connessione diretta o dedicata tramite una ISP.

2.5 ESEMPIO DI NETWORK GARR

Ad esempio la connessione internet delle università italiane è gestita dalla GARR (Gruppo per l'Armonizzazione delle Reti della Ricerca), la quale è una network nazionale di computer sviluppata per università e ricerca.

La network GARR è connessa ad altre networks nazionali di ricerche ed educazioni in Europa e nel mondo, ed è parte integrante dell'internet globale.

2.6 SERVIZI E RISORSE DI CONDIVISIONE

Lo scopo principale delle networks di computer e di internet è quello di condividere servizi e risorse tra differenti dispositivi (e utenti al di fuori dei dispositivi).

C'è una varietà di servizi abilitati dalle computer networks:

- file sharing: spostamento, lettura, copia dei file

- e-mails: mandare e ricevere e-mail elettroniche
- instant messaging: inviare e ricevere on-line

2.7 SERVIZI DI CONDIVISIONE NELLE NETWORKS LOCALI

Le risorse possono essere private e localmente disponibili: ad esempio, nelle networks locali ci possono essere dispositivi (stampanti, condizionatori), che possono offrire servizi e dispositivi (tablet, laptop, smartphone) che li usano.

È possibile usare tali servizi (ad esempio stampare un documento o accendere l'aria condizionata) persino senza accesso ad internet.

Tuttavia, la maggior parte dei dispositivi di oggi sono pubblici e forniti mediante internet dai service providers.

2.8 SERVICE PROVIDERS AL DI FUORI DELLA CONNETTIVITA'

Ci sono diverse compagnie famose il cui business principale è di fornire servizi tramite internet: Amazon, Google, Microsoft, IBM, Oracle, Aruba, Alibaba

Ci sono differenti modi di fornire servizi.

2.9 SERVER DEDICATI

Risorse e servizi possono essere pubblici e forniti da server distanti. Un server è uno speciale host designato per fornire uno o multipli servizi.

Per esempio, un web server o un e-mail server potrebbero trovarsi dall'altra parte del mondo, ma noi siamo quasi del tutto inconsapevoli di questa distanza.

Inoltre, potrebbero esserci più server che cooperano per fornire servizi e noi potremmo essere nuovamente inconsapevoli.

2.10 GRID COMPUTING

Il grid computing è un'infrastruttura decentralizzata di condivisione di risorse che tipicamente combina hardware (risorse) da differenti host in differenti luoghi geografici per raggiungere un obiettivo comune:

- per realizzare calcoli complessi
- immagazzinare enormi quantità di dati

Un famoso esempio è il SETI il cui progetto SETI@home permette a computer di tutto il mondo di condividere risorse allo scopo di analizzare segnali radio dallo spazio alla ricerca di intelligenza extra-terrestre.

2.11 CLOUD COMPUTING

Il Cloud computing è un'architettura centralizzata in cui le risorse (hardware e software) tipicamente gestite da compagnie (service providers), sono offerte su richiesta come servizi da host esterni.

In un'architettura di tipo cloud diversi servizi possono essere offerti quali ad esempio:

- Applicazioni per la produttività, giochi, servizi di comunicazione, social network ecc
- Storage, database condivisi, web servers
- Macchine virtuali con specifiche configurazioni, server, firewalls ecc

Una definizione largamente usata di cloud computing è fornita dal NIST (U.S. National Institute of Standards and Technology): Il cloud computing è un modello che consente un network access onnipresente, conveniente e su richiesta ad una pool condivisa di risorse informatiche configurabili (ad esempio networks, servers, storage, applicazioni e servizi), che possono essere rapidamente previste e rilasciate con uno sforzo gestionale minimo o con un'interazione tra service provider.

Questo modello cloud è composto da cinque caratteristiche essenziali:

- Sel-service su richiesta: un consumatore può unilateralmente disporre di capacità informatiche, come richiesto automaticamente senza richiedere alcun tipo di interazione umana con ciascun server provider.
- Accesso al broad network: le capacità sono disponibili tramite la network e accedute mediante i meccanismi standard che promuovono l'uso da parte di piattaforme client eterogenee thin o thick. (mobile phones, tablets, laptops and workstations)
- Resource pooling: le risorse informatiche del provider vengono prelevate per servire diversi consumatori utilizzando un modello multi-tenant, con differenti risorse fisiche e virtuali, dinamicamente assegnate e riassegnate a seconda della richiesta del consumatore.
- Elasticità rapida: le capacità possono essere elasticamente previste e rilasciate, in alcuni casi automaticamente, per scalare rapidamente al di fuori e all'interno in base alla domanda. Al consumatore, le capacità disponibili per le previsioni sembrano essere illimitate e possono essere appropriate in qualsiasi quantità in qualsiasi momento.
- Servizio misurato: i sistemi di tipo cloud controllano e ottimizzano automaticamente l'utilizzo delle risorse sfruttando una capacità di misurazione ad un livello tale di astrazione appropriato alla tipologia di servizio. L'utilizzo delle risorse può essere monitorato, controllato e riportato, fornendo trasparenza sia al consumatore che al fornitore del servizio utilizzato.

2.11.1 CLOUD COMPUTING: MODELLI DI SERVIZIO

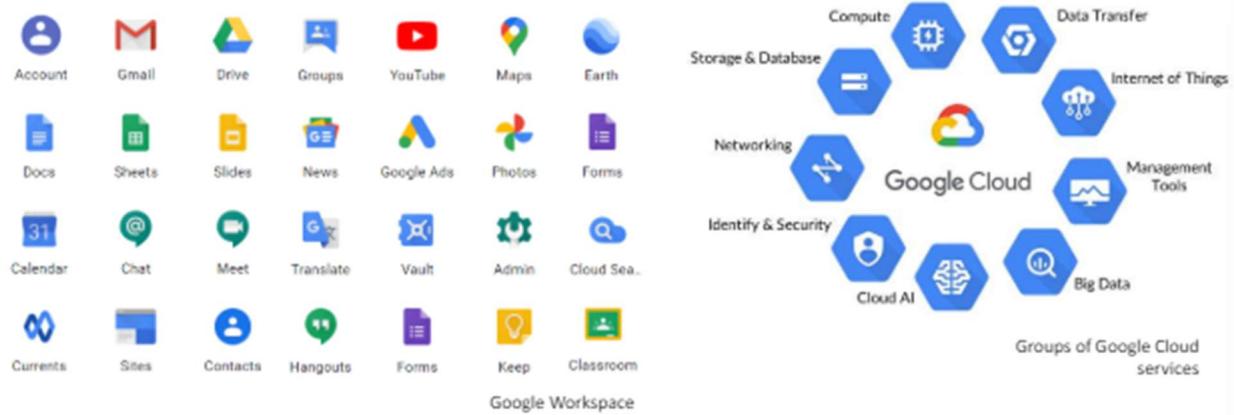
I servizi raggruppati NIST sono forniti da un'architettura di cloud computing in 3 categorie (service models) a seconda di quanto dell'infrastruttura di computing sia gestita dal service provider.

- Software as a Service (SaaS): applicazioni complete fornite insieme alle componenti software/hardware necessarie per essere eseguite.

- Platform as a Service (PaaS): piattaforme funzionanti fornite con specifiche configurazioni (es. sistemi operativi, APIs ecc)
- Infrastructure as a Service (IaaS): principalmente ciò che viene fornito è l'hardware.



Un classico esempio di Cloud Service Provider (CSP) è google. I servizi più popolari vengono forniti mediante il cosiddetto Google Workspace che è principalmente un SaaS. Servizi aggiuntivi sono forniti come parte integrante del Google Cloud Infrastructure.



2.12 INTERNET OF THINGS

L'Internet of Things (IoT) è l'approccio di dotare i semplici dispositivi come smart devices di sensori, unità di elaborazione, connettività ecc. per controllarli/monitorarli in maniera remota tramite internet.

IoT può essere usato in combinazione con il cloud computing come segue:

- Gestione online dei dispositivi utilizzando interfacce utente grafiche (GUIs) e applicazione da remoto.
- Immagazzinare dati e informazioni dai dispositivi.
- Elaborare informazione attraverso algoritmi di previsione o tecniche di machine learning.

Diverse “cose” che sono capaci di connettersi ad internet sono già state integrate nella nostra vita: smartphones, smart tv, smartwatches ecc.

Oggetti collegati enfatizza in particolare il problema della privacy e della sicurezza.

3 STANDARDIZZAZIONE

3.1 IL FABBISOGNO DI STANDARD E PROTOCOLLI

I computer networks come internet possono essere piuttosto complessi e diffusi tra i differenti luoghi, paesi, utenti ecc.

Chiaramente un set di regole o un suolo comune dovrebbe essere definito così che tutti i partecipanti sappiano come fornire o utilizzare i servizi.

Sin dalle origini di internet, uno dei maggiori sforzi era quello di realizzare e definire protocolli e standard che regolassero la comunicazione.

La comunicazione mediante dispositivi comprende diversi problemi quali ad esempio il trasporto fisico, l'indirizzamento, il controllo di errori, la regolazione e la conversione dei dati, la sicurezza, la sincronizzazione ecc.

Per permettere la comunicazione sia il ricevente che il mittente devono seguire un protocollo.

Quando più o differenti dispositivi provenienti da diversi posti comunicano, tutti loro devono essere d'accordo su un protocollo comune che diventa di conseguenza uno standard.

Ci sono centinaia di protocolli che regolano tutti gli aspetti di una comunicazione dai collegamenti fisici alle applicazioni:

- Come cavi e collegamenti dovrebbero essere creati (materiali, frequenze, schermatura, ecc)
- Come gli indirizzi dovrebbero essere assegnati
- Come i dati dovrebbero essere inglobati in pacchetti o frames per la trasmissione
- Come gli errori dovrebbero essere individuati o corretti
- Come le applicazioni dovrebbero scambiarsi i dati
- Ecc ecc

3.2 CHI STANDARDIZZA GLI STANDARD?

Al giorno d'oggi i protocolli e gli standard di internet sono definiti da una comunità di esperti chiamata Internet Engineering Task Force (IETF).

IETF è tipicamente organizzata in gruppi open working, ciascuno dei quali si focalizza su specifici aspetti di internet, i cui membri interagiscono mailing list ed incontri.

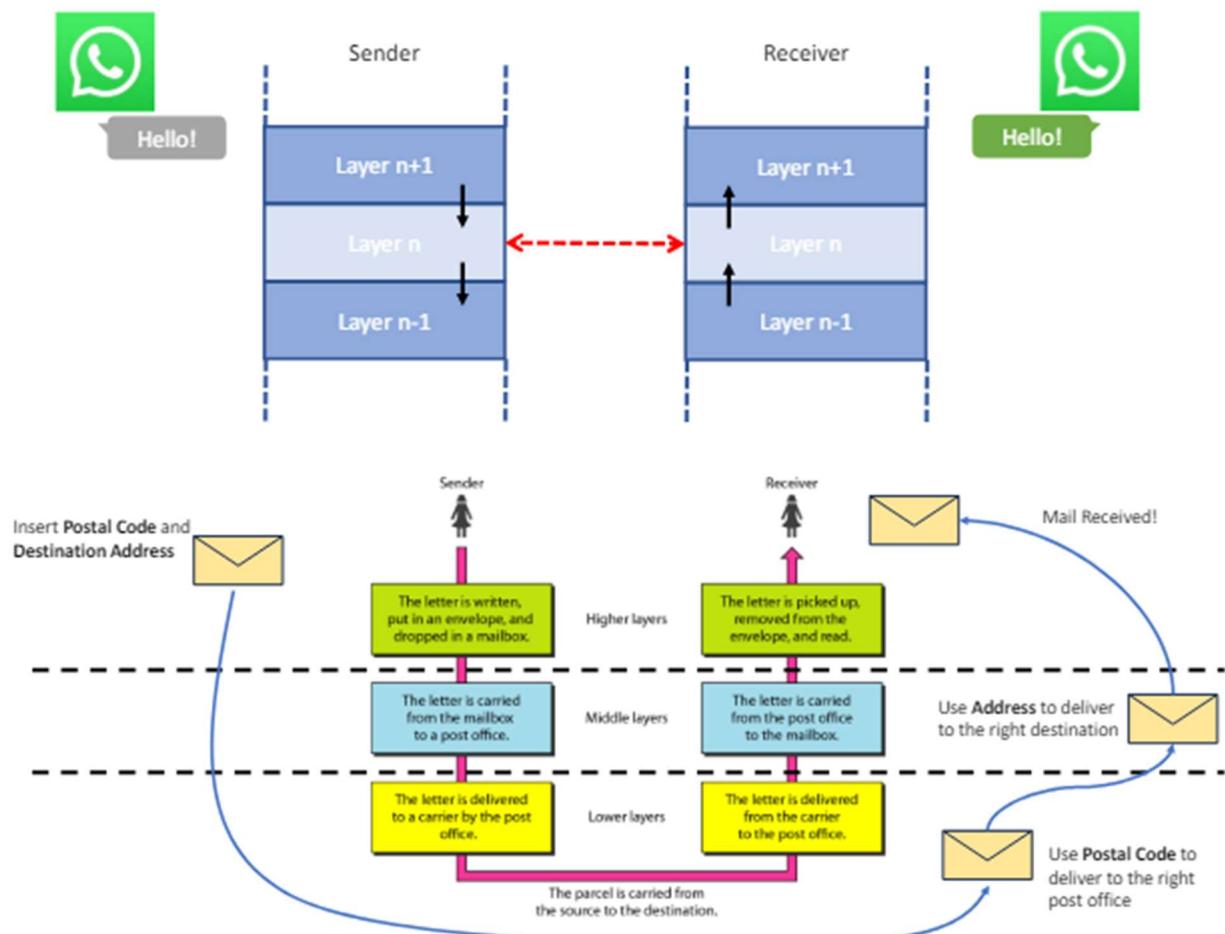
Ciascuno di questi working groups produce documenti numerati chiamati Request for Comments (RFC) includendo descrizione e definizione di protocolli, concetti, metodi sottostante un protocollo. Per esempio il protocollo IPv4 fu definito nel 1981 mediante RFC 791.

3.3 MODELLI DI NETWORK: IL MODELLO A STRATI

Un approccio ragionevole per affrontare i differenti problemi di comunicazione è quello di realizzare un modello a strati (Divide et Impera):

- Ciascun livello è concettualmente responsabile per uno specifico compito (risolve un preciso problema)
 - Ciascun livello fa affidamento ai servizi provenienti dal livello sottostante e fornisce servizi al livello sovrastante.
- Pro: modularità -> i livelli sono semplici ed indipendenti.
 Contro: scalabilità -> scalare troppi livelli è inefficiente.

Quando due dispositivi comunicano ciascun livello del mittente comunica con il rispettivo livello dal lato del ricevente mediante uno specifico protocollo.

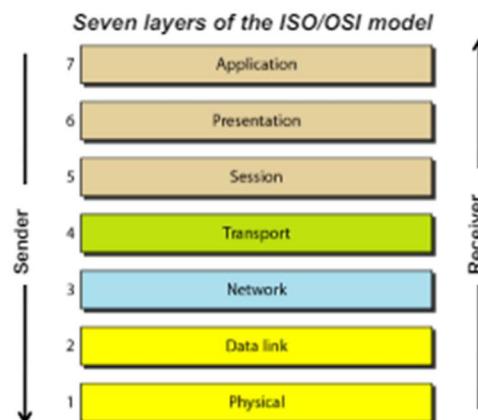


3.4 IL MODELLO ISO-OSI

Negli anni '80 la ISO (International Standards Organization) definì un modello a strati per le computer networks: il modello OSI (Open System Interconnection).

Il modello ISO/OSI include 7 livelli:

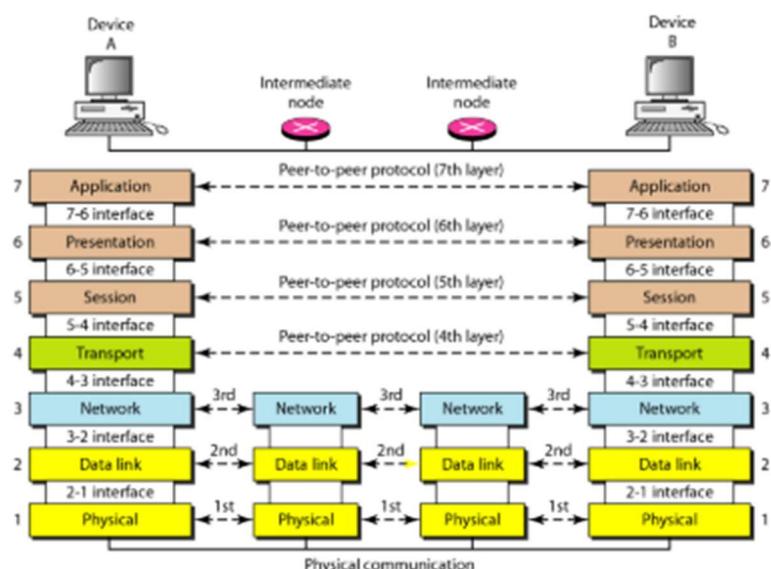
- 1. Trasmissione fisica dei raw bits (livello fisico)
- 2. Definizione del formato dei dati (livello data-link)
- 3. Instradamento dei messaggi attraverso la rete (livello network)
- 4. Protocolli di trasmissione (TCP, UDP) (livello di trasporto)
- 5. Gestione delle porte e delle sessioni (livello sessione)
- 6. Traslazione dei messaggi (codifica, compressione, decriptazione ecc) (livello di presentazione)
- 7. Utilizzo dei dati e interazione uomo-macchina (livello applicazione)



Non sempre tutti gli strati sono implementati.

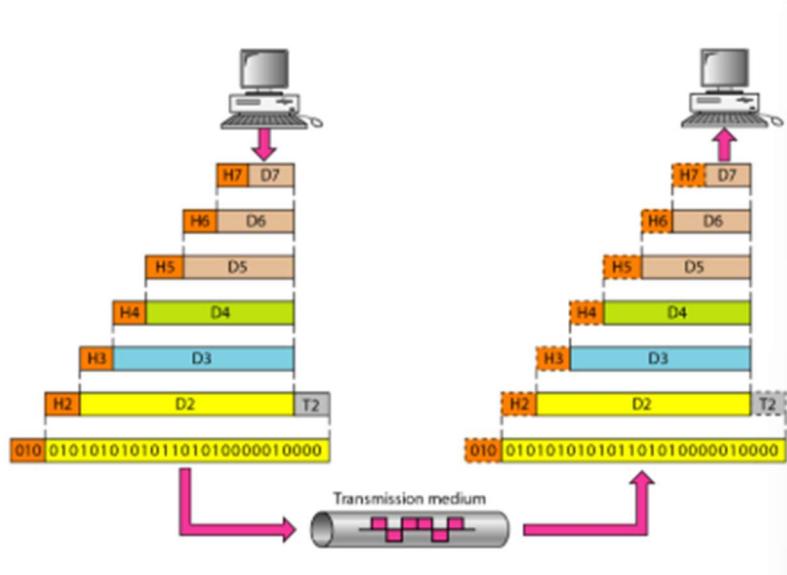
Generalmente, nei nodi intermedi (switches, routers) solo 2 o 3 strati vengono implementati.

Solo gli end-points implementano l'intero stack dal livello fisico al livello applicazione.



Incapsulamento (top-down): ogni livello del mittente aggiunge uno specifico campo del livello al messaggio, chiamato payload, sottoforma di header (H) o trailer (T).

Decapsulamento (bottom-up): questi campi vengono rimossi ed interpretati dai corrispondenti livelli presenti nel ricevente.



3.4.1 LIVELLO UNO: LIVELLO FISICO

Il livello fisico è responsabile dei movimenti dei singoli bit da un nodo a quello successivo.

- Presenta caratteristiche fisiche delle interfacce e dei media (connettori, cavi, segnali elettrici)
- Presenta una configurazione lineare (point-to-point o multipoint)
- Presenta una tipologia fisica del tipo mesh, star, ring o bus
- Presenta una modalità di trasmissione di tipo simplex, half-duplex o duplex
- Si occupa della rappresentazione dei bits
- Si occupa della velocità dei dati
- Si occupa della sincronizzazione dei bits

3.4.2 LIVELLO DUE: LIVELLO DATA LINK

Il livello data link è responsabile del movimento dei frames da un nodo al prossimo segmento.

- Lavora sui frammenti che sono porzioni di data tipicamente qualche centinaio di bytes.
- Si occupa del controllo del flusso e degli errori. (controllo delle sequenze dei frame)
- Si occupa del controllo degli accessi.

3.4.3 LIVELLO TRE: LIVELLO NETWORK

Il livello network è responsabile della consegna di pacchetti individuali dall'host sorgente all'host destinazione.

- Lavora sui pacchetti (tipicamente più grandi e più complessi dei frame)
- Si occupa della consegna dei pacchetti dalla sorgente alla destinazione

- Si occupa dell'indirizzamento logico
- Si occupa dell'instradamento

3.4.4 LIVELLO 4: LIVELLO DI TRASPORTO

Il livello di trasporto è responsabile della consegna del messaggio da un'estremità all'altra (il messaggio è ricevuto dal programma corretto).

- Si occupa della consegna end-to-end
- Si occupa del controllo di connessione (connection-oriented o connection-less)
- Si occupa della segmentazione e riassemblamento (a/da pacchetti del livello 3)
- Si occupa dell'indirizzamento delle porte
- Si occupa del controllo degli errori e del flusso

3.4.5 LIVELLO 5: LIVELLO SESSIONE

Il livello sessione è responsabile del controllo del dialogo e della sincronizzazione della richiesta/risposta.

- Stabilisce, mantiene e sincronizza l'interazione tra i sistemi che stanno comunicando
- Si occupa del controllo e della gestione del dialogo
- Si occupa della sincronizzazione mediante checkpoints

3.4.6 LIVELLO SEI: LIVELLO DI PRESENTAZIONE

Il livello di presentazione è responsabile della rappresentazione dei dati

- Si occupa della traslazione
- Si occupa di crittografia e decriptazione
- Si occupa della compressione

3.4.7 LIVELLO SETTE: LIVELLO APPLICAZIONE

Il livello applicazione è responsabile di fornire servizi all'utente

- Accesso al World Wide Web
- Trasmissione ed accesso ai file
- Servizi di mail

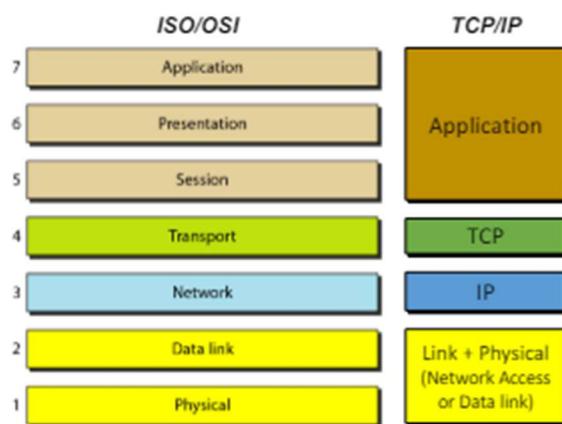
3.5 IL MODELLO TCP/IP

Il modello ISO/OSI è di diritto lo stack standard per le computer networks, ma è piuttosto complesso e dettagliato.

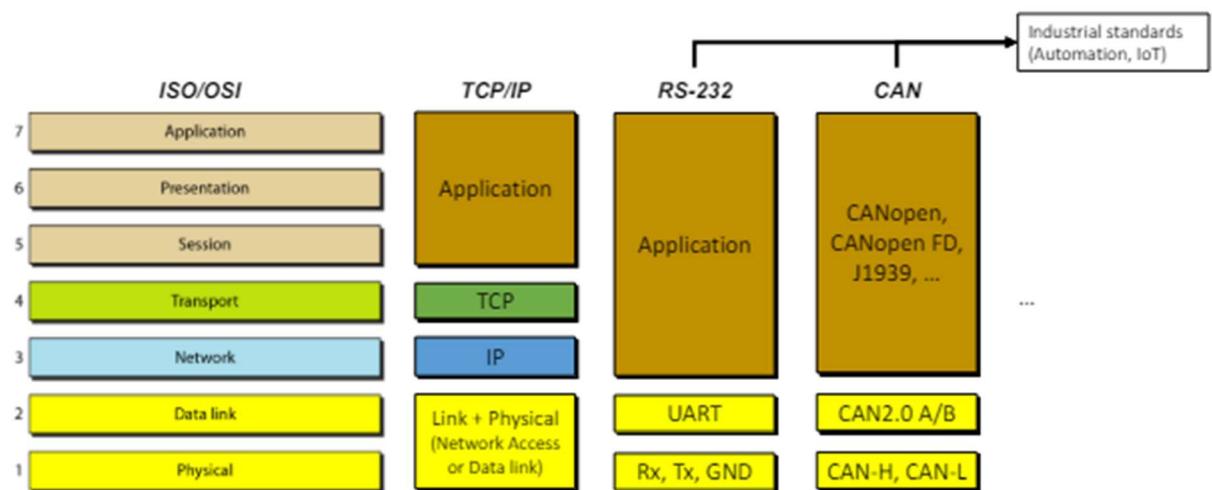
Il modello TCP/IP (Internet Protocol Suite) è un set di protocolli di comunicazione attualmente utilizzato sia in internet che in networks locali.

- TCP Transmission Control Protocol
- IP Internet Protocol

Il modello TCP/IP è di fatti lo stack standard per la comunicazione internet.



Esistono differenti protocolli per la comunicazione tra dispositivi che possono essere utilizzati a seconda della situazione, ciascuno di questi si basa tutt'ora sul modello a strati.



4. LIVELLO APPLICAZIONE

4.1 APPLICAZIONI NETWORK

Un'applicazione network è composta da programmi multipli che vengono eseguiti su differenti end-systems e comunicano l'uno con l'altro tramite la network.

Ad esempio una web application include due distinti programmi:

- Il programma browser che viene eseguito sull'host dell'utente che può essere desktop, laptop, tablet, smartphone ecce cc
- Il Programma Web server che viene eseguito sull'host web server

Le applicazioni sono programmi che vengono eseguiti sui dispositivi a consentono agli utenti di accedere ai servizi.

4.2 CREAZIONE DELLE APPLICATION NETWORK

La creazione di application network consiste nello scrivere programmi che

- Vengono eseguiti su differenti end systems, magari utilizzando linguaggi o sistemi operativi differenti
- Comunicano attraverso la network mediante sockets, attraverso specifici protocolli

Non è necessario scrivere software per dispositivi network-core.

- I dispositivi network non eseguono applicazioni utente.
- Dal punto di vista delle applicazioni, la network è idealmente una black-box.
- Ci sono specifiche librerie ad esempio sockets che implementano le funzionalità network.

4.3 APPLICAZIONI NETWORK (II PARTE)

Una network application architecture è designata dall'application developer e detta come l'applicazione è strutturata sui vari end systems:

- Client-server: i nodi sono eterogenei, vi è sempre un host costantemente attivo (server), che fornisce servizi che possono essere richiesti da tanti altri host ovvero i clients.
- Peer-to-peer (P2P): i nodi sono idealmente omogenei, l'applicazione sfrutta la comunicazione diretta tra coppie di host (peers) connessi ad intermittenza

4.3.1 APPLICAZIONE CLIENT-SERVER

In un'architettura client-server, i clients non comunicano direttamente l'uno con l'altro.

Il server ha un indirizzo fisso e ben noto (IP address) così che un client può sempre contattare il server inviando un pacchetto (richiesta) ad esso.

In un'architettura client-server più server sono spesso coinvolti per stare dietro a tutte le richieste da parte dei clients. Il client è tipicamente inconsapevole di ciò e li percepisce come un unico server.

Molteplici server possono essere:

- Raggruppati in data centers contenenti un enorme numero di server in un luogo specifico. (i server devono essere costantemente alimentati, manutenuti e ben connessi)
- Sparpagliati sottoforma di server distribuiti in tutto il mondo (i server devono essere interconnessi e coordinati)
- Organizzati in data centers distribuiti (esempio Google)

Un web application è un tipico esempio di struttura client-server:

- Vi è sempre un costantemente attivo web server che riceve richieste da parte dei browsers presenti sui client host
- Quando un web server riceve una richiesta per un oggetto da un client host, esso risponde inviando l'oggetto richiesto al client host
- Il web serve è sempre raggiungibile dai vari host
- Google ha dai 30 ai 50 data centers distribuiti in tutto il mondo, che collettivamente gestiscono la ricerca, ma anche YouTube, Gmail e altri servizi.

4.3.2 APPLICAZIONE PEER-TO-PEER

È spesso usato per applicazioni con traffico intenso.

Non vi è un singolo server con un indirizzo fisso, ma tutti i clients hanno il loro proprio indirizzo.

Le applicazioni puramente P2P sono rare: la maggior parte delle applicazioni hanno una struttura ibrida, che combina sia elementi client-server che elementi P2P.

- Per esempio, per molte applicazioni di instant messaging, i server vengono usati per tracciare gli indirizzi degli utenti, ma i messaggi da utente a utente sono inviati direttamente tra gli user host (senza passare attraverso server intermediari).

Le architetture P2P sono scalabili e distribuite.

- Per esempio, in un'applicazione P2P di file sharing, anche se ciascun peer genera un carico di lavoro richiedendo files, ciascun peer allo stesso tempo aggiunge capacità di servizio al sistema distribuendo i files ad altri peers.

L'architettura P2P sono inoltre economiche (non hanno bisogno di infrastrutture o bandwidth significativi) ma presentano problemi di sicurezza, performance e affidabilità.

Le applicazioni tipicamente P2P sono quelle di video e telephony conference (Skype) oppure file-sharing (BitTorrent)

- Nelle applicazioni di file-sharing il server può anche tracciare tutti i file disponibili in modo da velocizzare la ricerca.

4.4 COMUNICAZIONE TRA I PROCESSI

In una network application ci sono processi eseguiti su differenti macchine (potenzialmente eseguiti su differenti sistemi operativi) che comunicano attraverso la network.

- Nelle applicazioni web il processo di un browser del client scambia messaggi con il processo del web server.
- In un P2P file-sharing un file viene trasferito da un processo in un peer ad un processo in un altro peer.

Tra una coppia di processi in comunicazione vi è tipicamente un processo client e un processo server.

- Nelle applicazioni web un processo del browser è il client, mentre il processo del server è il server
- In un P2P file-sharing il peer che sta scaricando può essere visto come un client mentre il peer che sta caricando come un server.

In un'applicazione P2P i processi possono cambiare ruolo.

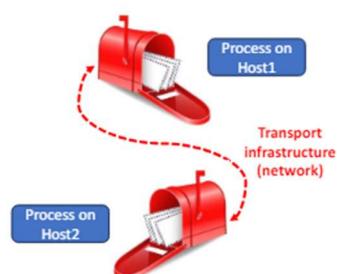
La maggior parte delle applicazioni network consistono in una coppia di processi in comunicazione inviando/ricevendo messaggi da/all'altro attraverso la suddetta network.

Un socket è un interfaccia software che permette ai processi di inviare/ricevere messaggi tramite network.

Considerando la stack TCP/IP, una socket è l'interfaccia che si pone tra lo strato di applicazione e lo strato di trasporto (TCP).

Un'analogia tipica è considerare le sockets come caselle postali:

- Quando un processo vuole mandare un messaggio ad un altro processo su un altro host, mette il messaggio nella casella postale
- Il processo di invio assume che vi è un'infrastruttura di trasporto dall'altra parte della propria porta che trasporterà il messaggio alla casella postale del processo destinatario.
- Una volta che il messaggio arriva all'host di destinazione, il messaggio passa attraverso la casella di posta del processo ricevente. (socket)



Le sockets sono usati come Application Programming Interface (API) tra l'applicazione e la network.

L'application developer ha il controllo su tutto ciò che riguarda il lato del livello di applicazione della socket ma ha poco controllo sul lato del livello di trasporto della socket.

Il solo controllo che l'application developer ha sul lato del livello di trasporto è:

- La scelta del protocollo di trasporto (TCP/UDP)
- Forse l'abilità di settare alcuni parametri del livello di trasporto quali il buffer massimo e la dimensione di segmento massima.

Una volta che l'application developer sceglie il protocollo di trasporto (se una scelta è possibile), l'applicazione è costruita assumendo come dati i servizi del livello di trasporto forniti da quel protocollo.

Seguendo l'analogia della casella postale, i processi hanno bisogno di un indirizzo per mandare messaggi.

Da quando le molteplici applicazioni network possono essere eseguite su un singolo host, per identificare il processo ricevente, i due pezzi di informazione necessitano di essere specificate:

- Un indirizzo dell'host (per trovare l'host giusto all'interno della network)
- Un identificatore del processo ricevente (per individuare il giusto processo nell'host)

L'indirizzo lavora al livello network mentre l'identificatore lavora al livello di trasporto.

Nelle networks, un host è identificato da un IP address (32 bit) mentre il processo è identificato da un numero di porta.

Le applicazioni più diffuse sono state convenzionalmente assegnate a specifici numeri di porta

- Per esempio, un web server è identificato dal numero di porta 80, un processo mail server è identificato dal numero di porta 25.

4.5 SERVIZI DAL LIVELLO DI TRASPORTO

Il livello di trasporto offre protocolli per garantire:

- Affidabilità del trasferimento dei dati: un dato inviato da un'estremità di un'applicazione è consegnato correttamente e completamente all'altra estremità.
- Throughput: la velocità alla quale il processo mittente può spedire bits al processo ricevente (bit/sec)
- Timing: ogni bit che il mittente carica nella socket arriva alla socket ricevente entro un preciso lasso di tempo
- Sicurezza: criptografia e decriptazione dei messaggi.

TCP: include una handshake tra i processi di servizio e un servizio di trasferimento dati affidabile (connection-oriented)

UDP: è leggero con servizi minimi, non vi è alcuna handshake, non vi è garanzia che i messaggi vengano ricevuti (connectionless).

4.6 PROTOCOLLI

Un protocollo a livello applicazione definisce come i processi della network application vengono eseguiti su differenti end systems, e su come si inoltrano messaggi tra di loro.

Più nello specifico, i protocolli a livello applicativo definiscono:

- Le tipologie di messaggi scambiati
- La sintassi delle varie tipologie di messaggio, come ad esempio i campi in cui è strutturato il messaggio e come questi ultimi sono delineati
- La semantica dei campi, ovvero il significato dell'informazione nei campi
- Le regole per determinare quando e come un processo invia messaggi o risponde ai messaggi

Ci sono differenti protocolli a livello applicativo che sono comunemente utilizzate nelle network e in internet. Alcune applicazioni possono scambiare informazioni sensibili (credenziali degli utenti, dati personali ecc). Su reti pubbliche tali messaggi vanno messi in sicurezza.

Application	Description
DHCP	Dynamic Host Configuration Protocol, assigns IP addresses
DNS	Domain Name System, translate website names to IP addresses
HTTP/HTTPS	HyperText Transfer Protocol (Secure), transfer web pages
SMTP/SMTPS	Simple Mail Transfer Protocol (Secure), sends email messages
SNMP	Simple Network Management Protocol, manages network devices
Telnet/SSH	Teletype Network (Secure SHell), allows command-line interfacing with remote hosts
FTP/FTPS	File Transfer Protocol (Secure), used to transfer files

4.6.1 FTP E FTPS

FTP (File Transfer Protocol) è uno dei più datati protocolli definiti in Internet ed è utilizzato per trasferire file tra host mediante la network.

Nella versione base di FTP i dati trasferiti sono in puro testo e pertanto la cosa migliore è di usarlo in applicazioni locali e private.

La versione sicura di FTPS (FTP Secure) protegge le informazioni sensibili crittografando i contenuti.

Sia FTP che FTPS hanno due componenti:

- Il protocollo che specifica i comandi (mostra, ottieni, cancella file ecc)
- Un applicazione software che implementa il protocollo ovvero un software client-side e server-side.

```
someuser@somehost:~$ ftp localhost
Connected to localhost.
220 (vsFTPD 3.0.3)
Name (localhost:guest): guest
331 Please specify the password.
Password:
230 Login successful.
200 System type is UNIX.
200 Will use binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
159 Here comes the directory listing.
drwxr-xr-x 2 1000 1000 4096 Aug 01 11:48 Desktop
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Documents
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Downloads
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Music
drwxr-xr-x 2 1000 1000 4096 Sep 22 12:25 Pictures
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Public
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Templates
drwxr-xr-x 2 1000 1000 4096 Aug 03 11:48 Videos
drwxr-xr-x 2 1000 1000 4096 Sep 22 12:14 remote_dir
221 Directory send OK.
ftp> get remote_dir/remote_file.txt
Local: /Documents/remote_file.txt Remote: remote_dir/remote_file.txt
200 Opening MEDIUM mode data connection for remote_dir/remote_file.txt (6 bytes).
200 Transfer complete.
6 bytes received in 0.00 secs (20.0553 kB/s)
ftp> exit
221 Goodbye.
someuser@somehost:~$ ls Documents/
remote_file.txt
someuser@somehost:~$
```

Common FTP commands:

- help - List all available FTP commands.
- cd - Change directory on remote machine.
- lcd - Change directory on local machine.
- ls - View the names of the files and directories in the current remote directory.
- mkdir - Create a new directory within the remote directory.
- pwd - Print the current working directory on the remote machine.
- delete - Delete a file in the current remote directory.
- rmdir - Remove a directory in the current remote directory.
- get - Copies a file from the remote server to the local machine.
- put - Copies a file from the local machine to the remote machine.

4.6.2 TELNET E SSH

Telnet (abbreviazione per teletype network) è un protocollo di applicazione client-server che fornisce l'accesso a terminali virtuali di sistemi remoti in networks di aree locali o in internet.

Il Secure Shell Protocol (SSH) è un rimpiazzo per il poco sicuro Telnet in quanto SSH è un protocollo crittografato. Esso viene utilizzato nell'eseguire servizi network in network poco sicure. Tuttavia è sempre basato su un'architettura client-server.

Sia Telnet che SSH hanno due componenti:

- Il protocollo che specifica come i messaggi sono strutturati e come gli host comunicano all'interno di una sessione
- Un'applicazione software che implementa il protocollo (un software dal lato client e dal lato server).

```
someuser@somehost:~$ ssh guest@localhost
guest@localhost's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-83-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

80 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Fri Sep 22 12:21:05 2023 from 127.0.0.1
guest@primalab:~$
```

Here we connect to ourselves (localhost), but an IP address can be specified:

1. Run ssh with `username@address` of the remote host.
2. Insert password
3. The terminal will show `username@hostname` of the remote shell. Now you may enter commands or close the connection (exit command).

4.6.3 WEB E HTTP

Fino agli inizi degli anni 90 internet era usato principalmente dai ricercatori, accademici o studenti universitari per accedere agli host remoti, per trasferire files dagli host locali agli host remoti e viceversa, ricevere ed inviare notizie, ricevere ed inviare mail elettroniche.

Sebbene queste applicazioni erano e continuano ad essere estremamente utili, internet era essenzialmente sconosciuto fuori dalle comunità accademiche e di ricerche.

Nei primi anni 90 il World Wide Web fu la prima applicazione Internet che catturò il pubblico generale.

Il World Wide Web è una collezione di informazioni quali documenti, immagini, video, audio ecc a cui si può accedere tramite Internet secondo uno specifico protocollo chiamato HyperText Transfer Protocol (HTTP)

La comunicazione basata su HTTP è tipicamente client-server:

- Un programma client che traduce le richieste degli utenti in messaggi HTTP. Questo programma è implementato nei web browsers. (Chrome, Firefox, Edge)
- Un server program che esegue la richiesta HTTP e restituisce una risposta HTTP.

HTTP principalmente definisce la struttura dei messaggi e come il client e il server dovrebbero scambiare tali messaggi.

Il web e i suoi protocolli servono come una piattaforma per le applicazioni basate sul web come YouTube, Gmail, i social networks.

4.6.4 URL

Le informazioni sul web sono chiamate risorse (o oggetti) e sono identificate da un Uniform Resource Locator (URL) che è una stringa composta come segue:

[protocol]://[usrinfo@[host][:port]]/[path][?query][#fragment]

dove:

- [protocol] è il protocollo che si usa nell'accedere alla risorsa (HTTP, HTTPS, FTP, ecc)
- [usrinfo] (opzionale) sono le informazioni dell'utente quali username e password seguiti da una @. Questo campo non viene quasi mai usato per motivi di sicurezza.
- [host] è il nome o l'IP address del server
- [port] (opzionale) è la porta da usare (spesso dedotta dal protocollo)
- [path] è il percorso della risorsa all'interno del server (/path/to/resource)
- [query] è preceduto da ? e da specifiche possibili richieste
- [fragment] preceduto da # identifica un elemento nella risorsa (ad esempio un form)

La maggior parte delle pagine Web consistono di un file di base HTML (HyperText Markup Language) e di diverse riferimenti ad oggetti addizionali.

- Esempio: una pagina web contenente un testo HTML e 5 immagini JPEG ha sei oggetti: il file HTML di base più le cinque immagini.

Un esempio di URL è: <http://www.someSchool.edu/someDepartment/picture1.gif>

dove

- http è il protocollo
- www.someSchool.edu è l'hostname
- /someDepartment/picture1.gif è il percorso all'oggetto

Un esempio reale di query URL: https://en.wikipedia.org/w/index.php?title=SSC_Napoli

dove:

- https è il protocollo
- en.wikipedia.org è l'hostname
- /w/index.php è il percorso alla pagina
- ?title=SSC_Napoli è la query

4.6.5 HTTP APPROFONDIMENTI

HTTP definisce come i web clients richiedono oggetti da web servers e come i servers li trasferiscono ai clients.

Quando un utente richiede un oggetto, il browser manda messaggi di richiesta http per gli oggetti nella pagina al server.

Il server riceve le richieste e risponde con messaggi di risposta http che contengono gli oggetti.

HTTP usa TCP come suo protocollo di trasporto sottostante (piuttosto che UDP). L'HTTP client prima inizializza una connessione TCP con il server. Una volta che la connessione è stabilita, i processi del browser e del server accedono al TCP tramite le loro interfacce socket.

- Dal lato del client l'interfaccia socket è la porta tra il processo del client e la connessione TCP.
- Dal lato del server è la porta tra il processo del server e la connessione TCP.

Il client invia messaggi di richiesta http nella propria interfaccia socket e riceve messaggi di risposta http dalla propria interfaccia socket. Allo stesso modo, il server http riceve messaggi di richiesta http dalla sua interfaccia socket e invia messaggi di risposta http nella sua interfaccia socket.

Una delle ragioni per cui http usa TCP come protocollo di trasporto è che diversi servizi utili sono forniti da TCP, soprattutto un trasferimento dati affidabile.

Affidabilità del richiamo: TCP garantisce che messaggi di richiesta/risposta http arrivino intatti a destinazione.

Qui vediamo uno dei grandi vantaggi dell'architettura a strati: applicazioni basate su http non devono preoccuparsi di raggiungibilità, perdita dati ecc

Le applicazioni https possono essere più semplici (sia logicamente che computazionalmente), delegando la maggior parte del lavoro al livello più basso dello stack.

4.6.5.1 HTTP: STATELESSNESS

La semplicità è molto importante per server basati su http che hanno a che fare con un ingente numero di richieste al secondo.

Le applicazioni http sono tipicamente stateless: il server non mantiene nessuna informazione circa l'interazione (sequenze di richieste/risposte) con uno specifico client.

- Esempio: Se un client chiede per lo stesso oggetto due volte in poco tempo, il server non considera questa richiesta ridondante, ma rimanda soltanto l'oggetto, in quanto ha completamente dimenticato il passato.

Non tutte le applicazioni sono stateless, diverse applicazioni al contrario sono statful.

4.6.5.2 HTTP: CONNESSIONI PERSISTENTI E NON PERSISTENTI

In molte applicazioni, il client e il server possono comunicare per un esteso periodo di tempo, inviando coppie multiple di richiesta-risposta.

- Questo flusso di messaggi può essere realizzato in modo immediato, periodicamente o ad intermittenza

Quando ci si affida ad una connessione TCP possiamo avere:

- Connessioni persistenti: tutte le richieste e le loro corrispettive risposte sono inviate sulla stessa connessione TCP.
- Connessioni non persistenti: per ogni coppia richiesta/risposta una nuova connessione TCP è stabilita.

Di default, al giorno d'oggi le applicazioni HTTP usano connessioni persistenti, mentre i client http e i server possono essere configurati per usare connessioni non persistenti se necessario. Le prime versioni di http usavano connessioni non persistenti di default.

ESEMPIO DI CONNESSIONE NON PERSISTENTE:

Assumiamo di richiedere una pagina web che consiste nel file HTML di base e 10 immagini JPEG mediante il seguente URL: • <http://www.someSchool.edu/someDepartment/home.index>

La connessione prende forma come segue:

- 1. Il processo client http inizializza una connessione TCP con il server www.someSchool.edu sulla porta 80 (la porta di default http), avremo di conseguenza due socket: uno per il client e uno per il server
- 2. L'http client invia un messaggio di richiesta http al server mediante la socket includendo il nome del percorso /someDepartment/home.index.
- 3. Il processo server http riceve il messaggio di richiesta mediante la socket, recupera l'oggetto /someDepartment/home.index (dalla RAM o dal disco), incapsula l'oggetto in un messaggio di risposta http e invia al client il messaggio di risposta attraverso la socket.
- 4. Il processo server http comunica alla TCP di chiudere la connessione, la TCP terminerà successivamente la connessione una volta che si è assicurato che il client ha ricevuto il messaggio di risposta intatto.
- 5. Il client http riceve il messaggio di risposta, il protocollo TCP termina la connessione. Il messaggio indica che l'oggetto incapsulato è un file HTML, il client estrae il file dal messaggio di risposta, esamina il file HTML e trova riferimenti ai dieci oggetti JPEG.
- 6. I primi quattro punti vengono eseguiti e ripetuti per ciascuno degli oggetti JPEG referenziati.

Poiché la connessione non è persistente per i diversi oggetti saranno necessarie 11 connessioni TCP con conseguenti 11 coppie di sockets per il trasferimento dell'intera pagina, in questo caso elementi della stessa pagina possono arrivare in momenti diversi.

Il modo in cui la pagina web viene poi mostrata all'utente dipende dal browser.

http definisce soltanto il protocollo di comunicazione tra il programma http client e il programma http server.

Le differenti connessioni possono essere stabilite in sequenza o in parallelo.

- Nell'esempio che abbiamo appena fatto, una volta che si ottengono i riferimenti alle dieci pagine JPEG, queste possono anche essere scaricate tutte insieme in parallelo.

L'utilizzo del parallelismo riduce il tempo di risposta, tuttavia incrementa il costo computazionale.

Tuttavia è importante sapere che stabilire connessioni multiple TCP potrebbe indurre ad un significativo aumento di tempo.

HTTP: CONNESSIONI NON PERSISTENTI (RTTs)

E' difficile stimare precisamente i tempi su internet. Possiamo stimare il tempo necessario a finalizzare la richiesta HTML in termini di round-trip times (RTTs).

RTT è il tempo che serve ad un piccolo pacchetto per viaggiare da un client ad un server e tornare al client.

Quando un utente clicca su un hyperlink il browser inizia una connessione TCP con il web server.

Le connessioni TCP sono garantite (connection-oriented), per fare ciò viene eseguito un handshake a tre vie.

La procedura handshake a tre vie TCP prende forma attraverso i seguenti tre passaggi:

- 1. Il client manda un piccolo segmento TCP al server per segnalare che una connessione è richiesta
- 2. Il server riconosce e risponde con un piccolo segmento TCP
- 3. Il client riconosce di nuovo al server

Dalla prospettiva RTT servono due RTTs per stabilire una connessione TCP (assumendo che non vi sia perdita di pacchetto)

Da notare che un simile handshake viene eseguito anche quando la connessione viene chiusa.

Le prime due parti di un handshake a tre vie richiede un RTT.

In http, il client manda il messaggio di richiesta http combinato con la terza parte di un handshake a tre vie (acknowledgment).

Una volta che il messaggio di richiesta arriva, il server manda il file HTML. Questa richiesta/risposta http impiega un altro RTT.

Orientativamente, il tempo totale di risposta è due RTTs più il tempo di trasmissione da parte del server del file HTML.

HTTP: CONNESSIONI PERSISTENTI

Nelle connessioni persistenti, il server lascia aperta una connessione TCP dopo aver inviato una risposta.

Le richieste e le risposte tra lo stesso client e server possono essere inviate sulla stessa connessione: un'intera pagina web può essere inviata su una singola persistente connessione TCP.

Multiple pagine web risiedenti sullo stesso server possono anche essere inviate da un server allo stesso client su una singola persistente connessione TCP.

Seguendo questo approccio, si possono risparmiare circa 2-4 RTTs per oggetto.

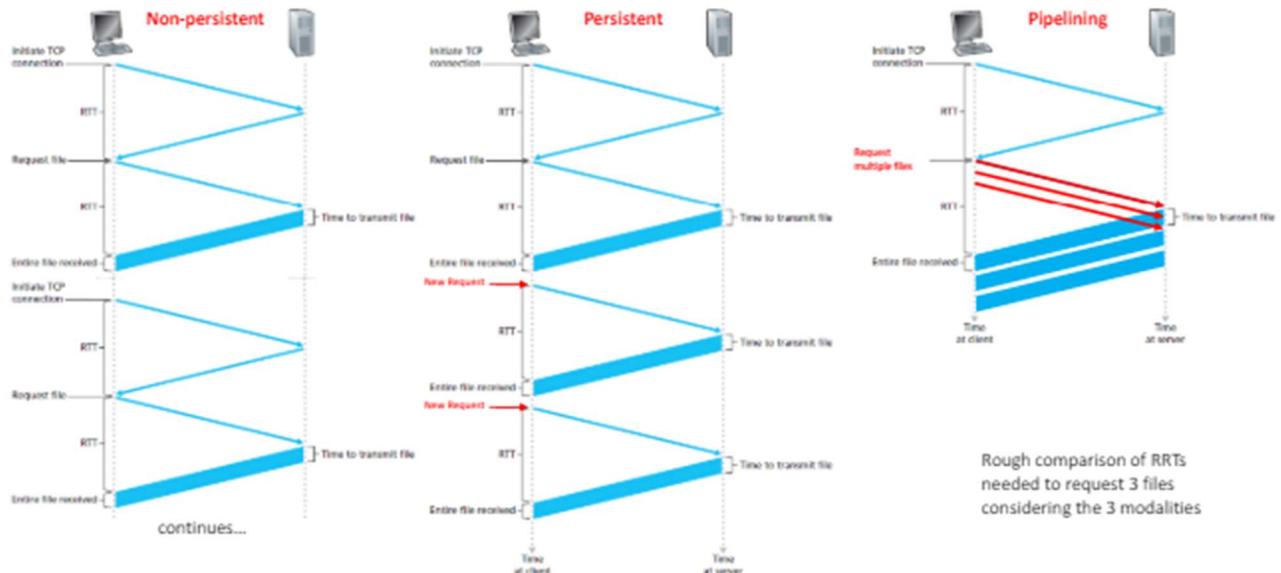
HTTP: CONNESSIONI PERSISTENTI CON PIPELING

Pipelining: richieste per oggetti possono essere fatte back-to-back, senza attendere risposte alle richieste pendenti.

Negli ultimi anni (a partire da http/2) molteplici richieste e risposte possono essere intercalate all'interno della medesima connessione.

Vi è anche un meccanismo per prioritizzare le richieste e le risposte http all'interno della medesima connessione.

Attualmente la modalità di default in http consiste nell'utilizzo di connessioni persistenti con pipelining.



HTTP: CONNESSIONI PERSISTENTI VS NON PERSISTENTI

Persistente:

- Più veloce, specialmente usando il pipelining
- Meno risorse necessarie (CPU e memoria)
- Più complesso da implementare, specialmente usando pipelining
- Le connessioni possono essere lasciate aperte anche se inusate. Tipicamente il server http chiude una connessione quando non viene usata per un certo tempo (timeout).

Non persistente:

- Facile da implementare e compatibile con la statelessness di http.
- Necessita di più risorse, per ciascuna di queste connessioni, i buffers TCP devono essere allocati e variabili TCP devono essere tenute sia dal client che dal server.
- Più lenta, 1-2 RTTs addizionali necessari per richiesta.
- Le connessioni non possono essere lasciate aperte.

4.6.5.3 HTTP: FORMATO DEL MESSAGGIO

Nel protocollo http ci sono due formati differenti per i messaggi di risposta e di richiesta.

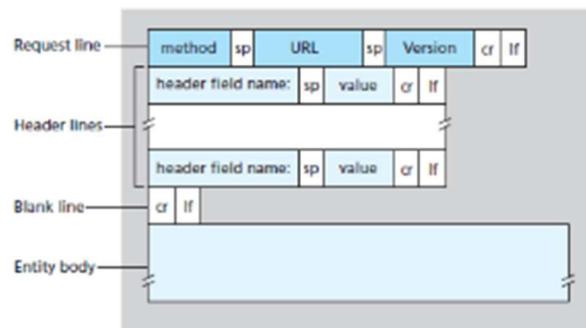
- Richiesta: specifica un comando o un metodo che il server http deve eseguire
- Risposta: riporta l'esito del comando (successo o fallimento) e i possibili dati (il file richiesto)

Entrambi i messaggi sono scritti in testo ASCII ordinario, pertanto è facilmente leggibile dagli umani.

FORMATO DEL MESSAGGIO DI RICHIESTA

Il formato del messaggio di richiesta include i seguenti campi:

- 1. Il metodo: specifica il comando richiesto che deve essere eseguito dal server
- 2. L'URL: è utilizzato per identificare l'oggetto su cui vogliamo operare
- 3. La versione: specifica la versione di http
- 4. Le header lines: contengono i parametri di richiesta, il numero e la tipologia di queste linee non sono fisse. Ciascuna linea include il nome e il valore del parametro Per esempio qui possiamo specificare se vogliamo una connessione persistente o non persistente.
- 5. Il corpo: è specifico del metodo e contiene i dati che sono potenzialmente associati al comando richiesto (ad esempio il testo da utilizzare per riempire un form).



Riferimento all'immagine – i campi sono separati da caratteri speciali:

- **Sp** è il carattere di spazio
- **Cr** è il carattere per andare a capo (\r)
- **If** è il carattere di avanzamento riga (\n)

FORMATO DEL MESSAGGIO DI RICHIESTA – METODI

1. Il metodo GET è usato per recuperare oggetti o risorse dal server.
 - Questo è uno dei più usati comandi in quanto ogni volta che cerchiamo una nuova webpage i file associati devono essere recuperati-
 - Nel metodo GET il campo corpo è vuoto.
2. Il metodo POST è usato per settare le informazioni all'interno di oggetti o risorse del server.
 - Una richiesta generata con un form non necessariamente utilizza il metodo POST. I form HTML utilizzano spesso il metodo GET ed includono i dati da inserire nel campo fragment dell'URL richiesto.
 - Il corpo contiene le informazioni da postare.

3. Il metodo HEAD è simile al metodo GET, quando un server riceve una richiesta con un metodo HEAD esso risponde con un http message che non include l'oggetto.
 - Gli application developers usano questo modo per fare debugging.
4. Il metodo PUT è spesso usato in concomitanza con gli strumenti di pubblicazione web.
 - Permette ad un utente di caricare un oggetto in una specifica directory in uno specifico web server.
 - Il metodo PUT è anche utilizzato dalle applicazioni che necessitano di dover caricare oggetti nei web server.
5. Il metodo DELETE permette ad un utente o ad un'applicazione di eliminare un oggetto da un web server.

FORMATO DI RICHIESTA DEL MESSAGGIO – ESEMPIO

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

In questo esempio abbiamo cinque linee:

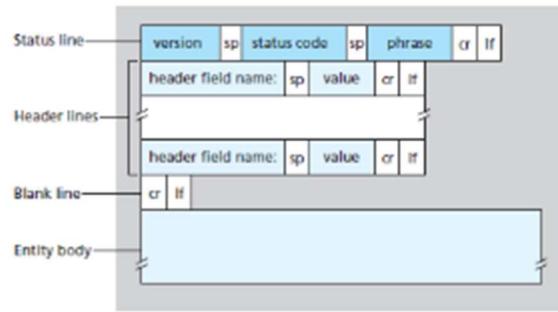
- La GET request line, formato da GET, risorse e versione.
 - Le quattro header lines
1. La prima header line specifica l'host in cui risiede l'oggetto . La destinazione non sempre è il prossimo host, potrebbe essere necessario che il messaggio debba essere inizialmente inoltrato prima ad un altro host, perciò l'indirizzo nel TCP potrebbe differire da quello dell'effettivo destinatario.
 2. La seconda header line specifica che il browser sta richiedendo una connessione non persistente
 3. La terza header line specifica la tipologia del browser. Questo campo potrebbe essere utile in quanto il server potrebbe mandare diverse versioni dello stesso oggetto a seconda della tipologia del browser.
 4. La quarta header line specifica che l'utente preferisce ricevere se disponibile una versione in francese dell'oggetto.

FORMATO DI RISPOSTA DEL MESSAGGIO

Il format generale del messaggio di risposta è simile a quello di richiesta.

In questo caso, invece della linea di richiesta, vi è una status line che riporta il risultato del comando che include:

1. La versione: riporta la versione http della risposta del server
2. Lo status code: un codice (numero) che specifica il risultato del comando
3. La frase: contiene il risultato della richiesta



FORMATO DI RISPOSTA DEL MESSAGGIO – STATUS CODE

Gli status code sono divisi in classi:

- 100-199 informativo: info riguardanti la richiesta
- 200-299 successo: la richiesta è stata eseguita con successo
- 300-399 redirection: ci sono ulteriori azioni richieste al client
- 400-499 client-error: la richiesta non può essere eseguita a causa di un problema con il client
- 500-599 server-error: la richiesta non può essere eseguita a causa di un problema con il server

Alcuni esempi di status code sono:

- 200 OK: richiesta eseguita con successo e l'informazione viene restituita nella risposta
- 301 spostato permanentemente: l'oggetto richiesto è stato permanentemente spostato, un nuovo URL è specificato nella header line “location” presente all'interno del messaggio di risposta
- 400 bad request: errore generico che indica che la richiesta non può essere compresa dal server
- 404 not found: il documento richiesto non esiste all'interno del server
- 505 versione http non supportata: la versione del protocollo http richiesta non è supportata dal server

```

HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)

```

Nel suddetto esempio, vi sono sei header lines. Il corpo invece contiene l'oggetto richiesto.

1. La prima header line informa il client che la connessione sarà chiusa dopo questo messaggio.
2. La seconda header line indica tempo e data in cui la risposta http è stata creata ed inviata dal server.

3. La terza header line indica che il messaggio è stato generato da un Apache Web server (simile alla header line user-agent del messaggio di richiesta)
4. La quarta header line indica il tempo e la data in cui l'oggetto è stato creato o è stato modificato l'ultima volta.
5. La quinta header line indica il numero di bytes dell'oggetto.
6. La sesta header line indica che l'oggetto è di tipo testo HTML.

4.6.5.4 COOKIES

Un server http è tipicamente stateless, ciò semplifica il design del server, riduce l'utilizzo di risorse e permette ai server di gestire migliaia di connessioni TCP contemporaneamente.

L'essere completamente stateless è una forte limitazione dato che diverse funzioni web sono client specific (ad esempio il carrello di Amazon dipende dal client, i contenuti suggeriti da Netflix sono basati sulle preferenze del client ecc)

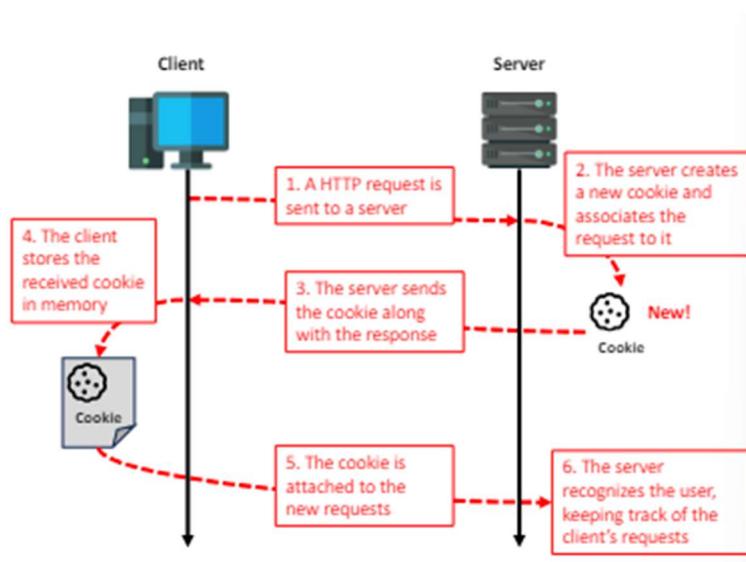
Per questi motivi, http usa i cookies. Un cookie è un token digitale (ID alfanumerico) usato dai server per identificare uno specifico client.

- I cookies permettono ai siti web di tenere traccia degli utenti, la maggior parte dei siti web commerciali usano i cookies.
- I cookies possono anche avere degli attributi (ad esempio data di scadenza).

Un cookie è creato dal server e consegnato al client.

La tecnologia cookie comprende quattro principali componenti:

- 1. Una header line “set-cookie” nel messaggio di risposta http.
- 2. Una header line “cookie” nel messaggio di richiesta http.
- 3. Un file sul client system gestito dal browser dell'utente.
- 4. Un database back-end sul server.



COOKIE – ESEMPIO

Supponiamo che un cliente host che usa eBay regolarmente contatti Amazon.com per la prima volta.

Il cliente ha già un cookie per eBay, ma non ne ha uno per Amazon.

Quando la richiesta arriva al web server di Amazon, il server crea un cookie e un'entry ad esso associato nel suo database back-end.

Il web server di Amazon poi risponde al browser del client ed include nella risposta http un header line “set-cookie” che contiene l’ID precedentemente creato.

Quando la risposta viene ricevuta, il browser aggiunge una riga ad uno speciale file cookie che include:

- Hostname del server
- L’ID number del cookie

Da adesso in poi, le richieste a partire dal client verso Amazon saranno associate al nuovo cookie includendo l’header line “cookie” al messaggio http.

Il server Amazon è quindi in grado di tracciare l’attività del client attraverso il suo database in quanto sa perfettamente quali pagine l’utente in questione visita, in quale ordine e quando.

Se il client ritorna sul sito di Amazon una settimana dopo il browser continuerà ad inserire l’header line “cookie” corrispondente nei messaggi di richiesta.

COOKIE – APPROFONDIMENTO

Amazon così come altri siti web utilizzano i cookies per fornire diversi servizi:

- Servizio dello shopping cart, dove il server può mantenere una lista di possibili acquisti durante la ricerca
- Raccomandazioni di prodotti, basato sulle pagine web visitate
- Registrazione dell’utente, infatti associando le informazioni dell’utente al cookie nel database non è necessario reinserire tali informazioni ogni volta.

Sebbene i cookies spesso semplificano l’esperienza di shopping in internet per l’utente vi sono delle controversie perché possono anche essere considerata un’invasione della privacy.

In teoria, usando una combinazione di cookies e di informazioni dell’account fornite dall’utente un sito web può imparare molto su un utente e potenzialmente vendere tali informazioni a terze parti.

4.6.5.5 WEB CACHING

Una web cache (anche chiamata proxy server) è un’entità network che soddisfa http al posto dell’originario web server.

La web cache dispone di un proprio disco fisso e contiene copie di oggetti recentemente richiesti in esso.

Un browser può essere configurato così che tutte le richieste http vengono prima dirette alla web cache per verificare se una copia dell'oggetto richiesto è già disponibile.

Supponiamo che un browser richieda un oggetto passando per una web cache:

1. Il browser stabilisce una connessione TPC con la web cache ed invia una richiesta http per un'oggetto alla web cache.
2. La web cache controlla se una copia di tale oggetto è immagazzinata localmente. Se sì, la web cache restituisce l'oggetto ponendolo all'interno del messaggio di risposta http che invia al browser del client.
3. Se la web cache non dispone di tale oggetto, essa apre una connessione TCP con il server di origine ed invia una richiesta http per il medesimo oggetto.
4. Quando la web cache riceve l'oggetto ne immagazzina una copia nel suo storage locale e ne invia una copia all'interno di un messaggio di risposta http al browser del client.

Da questa descrizione è facile comprendere che una cache è allo stesso server e client.

Il web caching ha visto il suo sviluppo in internet per due principali ragioni:

1. Ridurre sostanzialmente il tempo di risposta alle richieste dei client, particolarmente utile se vi è una connessione ad alta velocità tra il client e la cache.
2. Le web caches possono sostanzialmente ridurre il traffico di una compagnia o di un'istituzione in internet, riducendo di conseguenza i costi dovuti alla bandwidth.

Le web caches sono spesso installate all'interno di network locali di compagnie o istituzioni per velocizzare e ridurre il traffico.

Uno hit si verifica quando una cache fornisce con successo un oggetto senza necessariamente contattare il server originale.

Lo hit rate, ovvero la frazione delle richieste che vengono soddisfatte dalla cache tipicamente si aggira dallo 0.2 allo 0.7 questo vuol dire che circa il 70% delle richieste possono essere servite localmente.

La stessa rete dell'UNINA possiede un proxy istituzionale.

I proxy server possono anche essere usati, oltre che per performance, per accedere a servizi istituzionali.

Poiché le richieste vengono inoltrate da un proxy istituzionale dal punto di vista del server originale tutte le richieste provengono dall'istituzione.

Ci sono anche server proxy commerciali e gratuiti disponibili in internet.

Il web caching introduce un nuovo problema: la copia di un oggetto che risiede nella cache potrebbe essere datata.

Per aggirare questo problema, http dispone di un meccanismo che consente alla cache di verificare l'oggetto immagazzinato.

Il conditional GET è un messaggio di richiesta http che include il metodo GET e la header line "If-Modified-Since".

La cache controlla che l'oggetto sia aggiornato è se sì la versione immagazzinata viene inviata all'host richiedente.

Caching è anche eseguito localmente dai browsers. Il principio è lo stesso come per i server, i browser immagazzinano gli oggetti localmente così che non debbano più recuperarli dal server.

Questa è una tecnica molto comune nei moderni browser in quanto migliora drasticamente le performance.

La versione locale dell'oggetto tuttavia potrebbe non essere aggiornata, generando degli errori.

4.6.6 MAIIS E SMTP

La e-mail è una delle applicazioni più vecchia e più importante di internet.

Il Simple Mail Transfer Protocol (SMTP) è il principale protocollo a livello applicativo per le mail elettroniche in internet.

Ci sono due elementi coinvolti:

- 1. User agent: l'applicazione che consente la gestione delle mail (Gmail)
- 2. Mail server: un server che immagazzina le mail e mantiene caselle postali specifiche per ogni utente.

SMTP principalmente lavora su mail servers per consentire loro di scambiare mail.

Una volta che arriva una nuova mail al server viene immagazzinata all'interno di una casella postale specifica dell'utente, in attesa di essere scaricata localmente dall'user agent.

Gli utenti non scambiano mail direttamente (come nella messaggistica istantanea). È preferibile utilizzare mail server specializzati ed affidabili.

In SMTP vi è un client-side (mittente) e un server-side destinatario (ricevente) che vengono eseguiti sui mail servers, dove entrambi utilizzando l'affidabile TCP per trasferire mail.

SMTP

Assumiamo di avere un host A Alice che invia una mail ad un host B Bob, in questo caso abbiamo 4 elementi coinvolti:

- L'agente di Alice
- Il mail di server di Alice
- L'agente di Bob
- Il mail server di Bob

Il processo funziona come segue:

- 1. Alice invoca il suo user agent per l'e-mail fornendo l'e-mail address di Bob, compone il messaggio e istruisce l'user agent nell'inviare un messaggio.
- 2. L'user agent di Alice invia un messaggio al suo mail server, entro cui viene piazzato in una coda di messaggi.
- 3. Il lato client dell'SMTP eseguito sul mail server di Alice vede il messaggio nella coda dei messaggi. Apre una connessione TCP con un server SMTP che invece è eseguito sul mail server di Bob.
- 4. Dopo alcuni iniziali handshaking di SMTP, il client SMTP invia il messaggio di Alice nella connessione TCP.
- 5. Al mail server di Bob, il lato server dell'SMTP riceve il messaggio. Il mail server di Bob quindi piazza il messaggio nella casella postale di Bob.
- 6. Bob invoca il suo user agent per leggere il messaggio a sua convenienza.

Prima di tutto, il client SMTP che viene eseguito sul mail server dell'host mittente chiede a TCP di stabilire una connessione alla porta 25 del server SMTP, eseguito sul mail server dell'host ricevente.

Se il server non è disponibile, il client prova di nuovo più tardi. Una volta che tale connessione viene stabilita, il server e il client eseguono una serie di handshaking a livello applicativo. I client SMTP e i server si presentano prima di trasferire informazioni.

- Durante questa fase di handshaking, il client SMTP indica che l'indirizzo e-mail del mittente e l'indirizzo e-mail del ricevente.

SMTP può far affidamento sull'affidabile servizio di data transfer di TCP per far sì che il messaggio arrivi al server senza errori.

ACCESSING MAILS

La distribuzione dei server MSTD è più affidabile rispetto all'invio diretto di posta da utente a utente.

- I server sono sempre operativi
- I server sono certificati
- Il server continuamente cerca di reinviare le mail se la consegna fallisce

D'altro canto per accedere alle mail da un server è necessaria un'ulteriore applicazione client-server con un differente protocollo, le più popolari sono:

- Post Office Protocol – Versione 3 (POP3)
- Internet Mail Access Protocol (IMAP)
- HTTP

ACCESSING MAILS: POP3

Il Post Office Protocol – Versione 3 è un protocollo d'accesso e-mail estremamente semplice.

L'user agente (ovvero il client) apre una connessione TCP verso il mail server attraverso la porta 110.

Con la connessione TCP stabilita, il POP3 prosegue con le seguenti tre fasi:

- 1. Autorizzazione: l'user agent invia username e password per autenticare l'utente
- 2. Transazione: l'user agent può recuperare i messaggi, segnare i messaggi per l'eliminazione, rimuovere i mark per l'eliminazione e ottenere statistiche mail
- 3. Update: dopo che un client ha impartito il comando di chiusura, terminando così la sessione POP3, il mail server elimina i messaggi che erano stati marcati come da eliminare.

ACCESSING MAILS: IMAP

Con l'accesso POP3, i messaggi possono essere scaricati o eliminati. Azioni come la ricerca o l'organizzazione di mail in cartelle non sono prese in considerazione.

L'Internet Mail Access Protocol (IMAP) permette ai server di fornire caratteristiche addizionali:

- Gestire e creare cartelle
- Eseguire la ricerca nelle cartelle remote per i messaggi che rispettano specifici criteri.
- Permette all'user agent di ottenere alcune parti dei messaggi (per esempio solo header line ecc)
- Permette a diversi client di essere connessi al medesimo server.

ACCESSING MAILS: HTTP

Sempre più utenti oggi inviano ed accedono alle loro mail tramite i loro web browsers.

L'accesso basato su web tramite http fu introdotto da Hotmail nella metà del 1990 ed è adesso l'approccio tradizionale (Google, Yahoo, Virgilio ecc) ed è usato quasi dalla maggior parte delle università e corporazioni (UNINA inclusa).

Con questo servizio, l'user agent non è altro che un ordinario browser web, e l'utente comunica con la sua remota mailbox mediante http piuttosto che con i protocolli POP3 e IMAP.

Questo funziona per gli user-agents, mentre i mail servers fanno ancora affidamento sullo standard SMTP per lo scambio dei messaggi.

4.6.7. DNS

Ci sono due modi per identificare un host: mediante l'hostname e mediante l'IP address. Le persone preferiscono un identificatore hostname più mnemonico, mentre i dispositivi network preferiscono un IP address con dimensioni prefissate e gerarchicamente strutturato.

Il Domaine Name System (DNS) è un protocollo a livello applicativo che gestisce la traduzione dall'hostname agli indirizzi IP.

Si tratta di un protocollo client-server in cui un client DNS chiede al server DNS per una specifica traduzione hostname-address.

I server DNS sono spesso macchine UNIX che eseguono il software noto come Berkeley Internet Name Domain (BIND), che tipicamente usa una connessione UDP e il numero di porta 53.

Per esempio, se un server richiede l'URL www.someschool.edu/index.html, il relativo indirizzo IP è recuperato come segue:

- 1. Il browser estrae l'hostname, www.someschool.edu, dall'URL e lo passa al lato client dell'applicazione DNS.
- 2. Il client DNS invia una query contenente l'hostname ad un server DNS
- 3. Il client DNS riceve una risposta contenente l'indirizzo IP per quello specifico hostname.
- 4. Una volta che il browser riceve l'indirizzo IP da DNS esso può inizializzare una connessione TCP con il processo server http localizzato alla porta 80 a quello specifico indirizzo IP.

DNS: SERVERS DISTRIBUITI

Internet DNS è distribuito e gerarchicamente organizzato. Ci sono diverse servers sparsi in tutto il mondo che forniscono un servizio DNS.

Questo approccio è preferibile ad un DNS centralizzato per diverse ragioni:

- Evitare un singolo punto di fallimento: se il server DNS crasha, l'intero internet crasha.
- Regolare il volume del traffico: centinaia di milioni di host utilizzano DNS
- Una maggior raggiungibilità: un singolo server DNS non può essere “vicino” a tutti i client
- Una più semplice manutenzione e aggiornamento: è possibile aggiornare DNS locali senza influenzare l'intero internet.

DNS: GERARCHIA

Ci sono principalmente tre classi di server DNS:

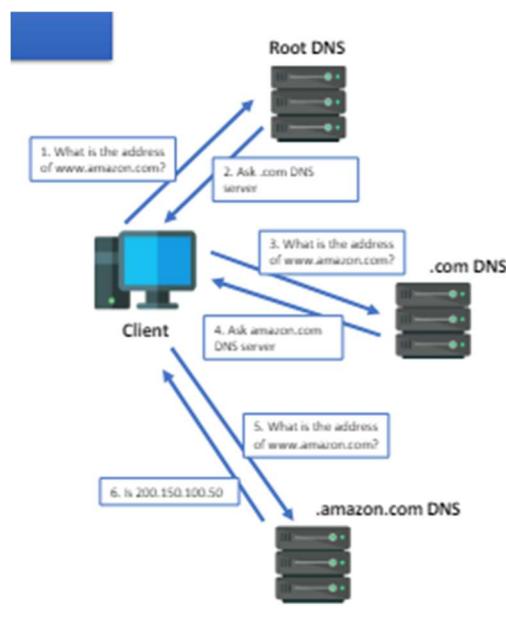
- 1. Root DNS servers: ci sono più di 400 root name servers sparsi in tutto il mondo, gestito da 13 differenti organizzazioni. I root name servers forniscono gli indirizzi IP ai TLD servers.

- 2. Top-level domain (TLD) servers: ci sono uno o più TLD server per ciascun top-level domain (per esempio .com, .org, .net, .it ecc). I TLD servers forniscono indirizzi IP agli authoritative DNS servers.
- 3. Authoritative DNS servers: forniscono una mappatura reale dell'associazione hostname-IP. Questi possono essere direttamente posseduti da organizzazioni come amazon, facebook ecc o offerti da terze parti.

DNS: ESEMPIO

Quando un client DNS vuole determinare l'indirizzo IP per uno specifico hostname (ad esempio www.amazon.com) succede questo:

- 1. Il client prima contatta uno dei root servers, che restituisce gli indirizzi IP per i server TLD per il dominio top-level ".com"
- 2. Il client poi contatta uno di questi TLD servers, che restituisce un indirizzo IP di un authoritative server per "amazon.com"
- 3. Alla fine, il client contatta uno degli authoritative servers per amazon.com, il quale restituisce l'indirizzo IP per l'hostname "www.amazon.com"



DNS: RESOLVER

C'è un altro importante tipo di server DNS chiamato local DNS server oppure DNS resolver.

Un local DNS server non strettamente appartiene alla gerarchia dei server ed è tipicamente gestito dagli ISPs.

Quando un host si connette ad un ISP, l'ISP fornisce all'host gli indirizzi IP di uno o più dei suoi local DNS servers, generalmente quello più vicino possibile all'host.

Quando un host fa una DNS query, la query è inviata al local DNS server, che agisce come un server proxy, inoltrando la query alla gerarchia dei DNS server.

DNS: ALIASING

DNS fornisce un servizio di aliasing per i server in cui nomi lunghi o complicati sono associati a nomi più semplici e brevi

Host aliasing: gli host con hostname complicati possono essere associati ad aliases più mnemonici.

Tuttavia è importante ricordare che DNS può essere sempre invocato da un'applicazione con lo scopo di ottenere l'hostname canonico per un alias fornito così come per un indirizzo IP associato.

DNS: ESEMPIO

Un possibile esempio potrebbe essere il seguente: in Linux così come in altri sistemi operativi è possibile utilizzare il comando nslookup per richiedere una traduzione hostname-address.

```
guest@prismalab:~$ nslookup www.unina.it
Server:      127.0.0.53
Address:    127.0.0.53#53

Non-authoritative answer:
Name:      www.unina.it
Address:   143.225.19.50

guest@prismalab:~$ nslookup 143.225.19.50
50.19.225.143.in-addr.arpa      name = www.unina.it.

Authoritative answers can be found from:
guest@prismalab:~$
```

Possiamo ottenere la traduzione in entrambi i seguenti modi:

- Possiamo ottenere l'indirizzo IP a partire dall'hostname.
- Possiamo ottenere l'hostname a partire dall'indirizzo IP.

DNS: LOAD DISTRIBUTION

Un DNS può essere utilizzato per eseguire load distribution tra i servers replicati (round-robin DNS)

Tipicamente siti impegnati vengono replicati su diversi server, con ciascun server che viene eseguito su un differente end system e ciascuno di essi aventi indirizzo IP differente (molteplici indirizzi IP associati al medesimo hostname canonico).

Il database DNS contiene questo set di indirizzi IP. Quando i clients eseguono una query DNS per un nome mappato su un set di indirizzi, il server può ruotare l'ordine in cui gli indirizzi vengono restituiti.

4.6.8 PEER-TO-PEER

A differenza del client-server, l'architettura P2P fa uso minimo o del tutto assente dei servers, perché in questo caso si ha una coppia di host (peers) connessi ad intermittenza che comunicano direttamente l'uno con l'altro.

I peers non sono posseduti da alcun service provider ma sono invece desktop e laptop controllati da utenti.

Un'applicazione naturale del P2P è il file sharing:

- In un'architettura client-server, il server deve condividere i file a tutti i clients (il che significa un vistoso peso per il server, nonché una estremamente larga bandwidth richiesta).
- In un'architettura P2P, i peers che ricevono il file possono anche condividerlo con altri peers. In sostanza i peers sono sia client che server allo stesso tempo.

È facile comprendere che nel campo del file sharing l'approccio P2P tipicamente scala meglio rispetto a quello client-server.

D'altro canto, l'approccio P2P è piuttosto complesso da implementare.

Inoltre vi possono essere anche dei problemi di sicurezza nell'avere tutti questi clients che comunicano in maniera diretta.

4.6.8.1 BitTorrent

Un popolare protocollo P2P per il file sharing e la distribuzione è BitTorrent.

Un torrent è una collezione di tutti i peers che partecipano nella distribuzione di un particolare file. I peers all'interno del torrent scaricano chunks di egual dimensione del file da uno all'altro.

Quando un peer si unisce ad un torrent:

- 1. Comincia accumulando chunks.
- 2. Mentre il peer scarica chunks, allo stesso tempo carica chunks per altri peers.
- 3. Una volta che il peer acquisisce l'intero file, potrebbe egoisticamente lasciare il torrent, o altruisticamente rimanere nel torrent e continuare a caricare chunks per altri peers.

I peers possono lasciare il torrent in qualsiasi momento possedendo anche soltanto una parte dei chunks potendo poi in seguito riunirsi al torrent.

BitTorrent: TRACKER

Ciascun Torrent ha un nodo infrastrutturale chiamato tracker che tiene traccia dei peers che partecipano al torrent. Tali trackers sono sostanzialmente servers.

Quando un nuovo peer si unisce ad un torrent:

- Si registra con il tracker e periodicamente lo informa riguardo il suo status.
- Il tracker fornisce gli indirizzi IP di un sottogruppo di peers provenienti dal torrent selezionato casualmente.

Avendo la lista dei peers, il nuovo host tenta di stabilire connessioni TCP concorrenti con tutti i peers presenti nella lista.

- Durante l'esecuzione, alcuni dei peers connessi potrebbero andarsene mentre altri peers provenienti al di fuori della lista iniziale possono tentare di stabilire nuove connessioni.

In qualsiasi momento, ciascun peer avrà un sottogruppo di chunks del file, con differenti peers che dispongono di differenti sottogruppi. Periodicamente, un host chiederà a ciascun peer connesso mediante le connessioni TCP la lista dei chunks che loro posseggono.

Il client scaricante decide quali chunks richiedere e a chi seguendo il principio del “più raro per primo”:

- Rarest-first: i chunks con il minor numero di copie ripetute tra i vicini vengono prioritizzati. In questo modo, i chunks più rari vengono redistribuiti più velocemente, così da pareggiare il numero di copie all'interno del torrent.

Il client caricante decide quali richieste vengono servite seguendo due principi intrecciati:

- Trading: gli host danno priorità ai 4 migliori neighbors che stanno correntemente fornendo dati alla velocità più alta. Questo controllo viene eseguito periodicamente (10 secondi) e la lista dei 4 migliori neighbors costantemente aggiornata.
- Random selection: ogni trenta secondi un host seleziona un neighbor addizionale in maniera casuale e gli invia chunks. Se lo scambio casuale è buono i due host entreranno nelle loro rispettive “liste dei migliori”. Questo processo permette anche ai peers di trovare dei partner con velocità di caricamento compatibili.

4.7 CREAZIONE DI NUOVE APPLICAZIONI

Si è parlato di diverse protocolli e applicazioni network, adesso si parlerà di come le applicazioni network vengono create.

La maggior parte delle applicazioni network includono un programma client-side e un programma server-side che comunicano mediante la network.

Quando questi due programmi vengono eseguiti, un processo client e un processo server vengono creati, e questi processi comunicano l'uno con l'altro leggendo e scrivendo tramite sockets.

Quando si crea un'applicazione network, il compito principale dello sviluppatore è quindi quello di scrivere il codice per entrambi i programmi, sia client che server.

Ci sono due tipi di applicazioni network:

- 1. Applicazioni che si basano su un standard protocol. Ci sono diversi protocolli “aperti” le cui regole sono note. In questo caso i programmi client e server devono essere conformi a tali regole per essere compatibili con il protocollo.
- 2. Applicazioni che si basano su un proprietary protocol. In questo caso i programmi client e server impiegano un protocollo personalizzato (che non è stato apertamente pubblicato in un RFC o altro). In questo caso un singolo sviluppatore o un team di sviluppo creano sia i programmi client che i programmi server.

Ci sono due protocolli di trasporto che possono essere usati:

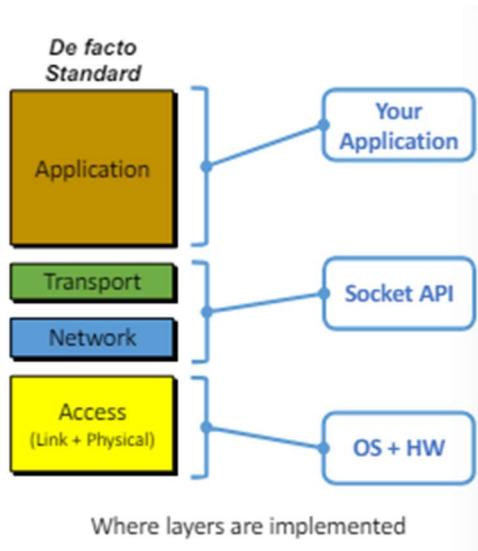
- 1. TCP (Transmission Control Protocol) che è connection oriented e fornisce un canale byte-stream affidabile attraverso il quale i dati fluiscono tra i due end systems.
- 2. UDP (User Datagram Protocol), che è connectionless ed invia pacchetti indipendenti di dati da un end system ad un altro senza alcuna garanzia riguardo la consegna.

I sockets sono l’elemento centrale per la creazione delle applicazioni network.

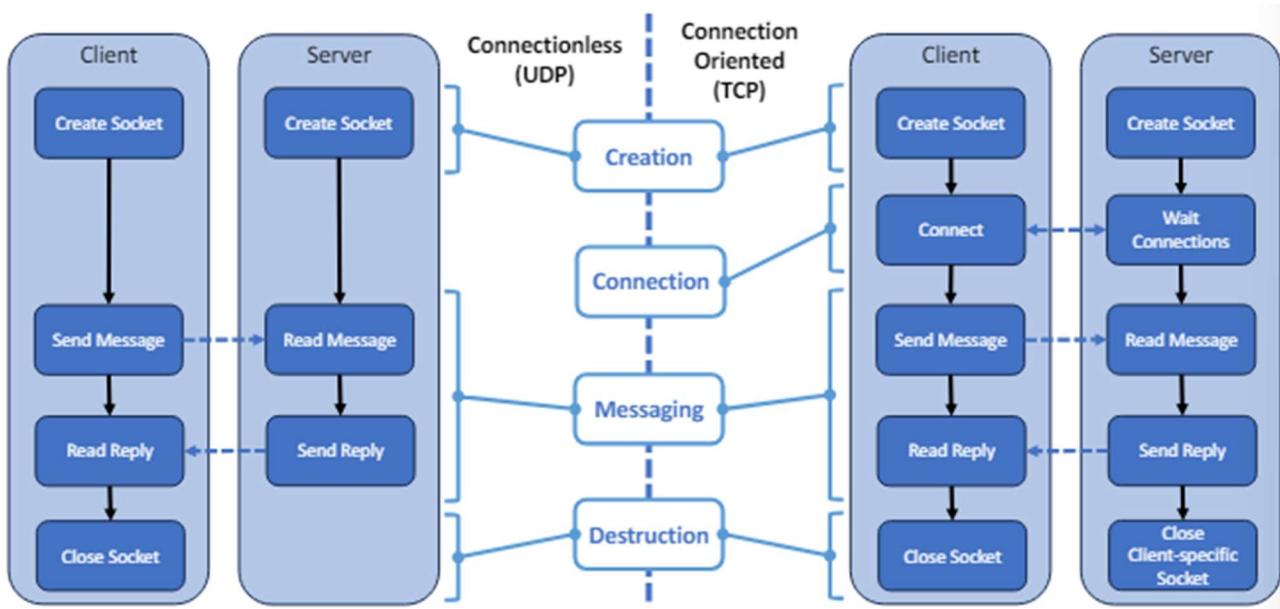
Essi tipicamente gestiscono le funzionalità dei livelli network (IP) e di trasporto (UDP/TCP) che sono visti come una black-box.

È importante sapere che esistono diverse applicazioni dette middleware che possono avvolgere le sockets semplificandone la loro creazione e fornendo funzionalità più complesse.

Comunque le API di base sono tutt’ora largamente usate.



4.7.1. UDP E TCP



Le sockets UDP e TCP implementano il processo precedente per permettere rispettivamente la comunicazione connectionless e connection-oriented.

Le API di base che forniscono le sockets UDP e TCP sono generalmente disponibili in quasi tutti i linguaggi di programmazione e sistemi operativi (C, C++, Java ecc).

L'implementazione sottostante e l'utilizzo possono essere più o meno differenti a seconda della configurazione delle macchine.

Nel dominio UNIX si usano le Berkeley Sockets (BSD o POSIX sockets) sia per le connessione TCP che per quelle UDP. Sono delle API originarie del linguaggio di programmazione C.

4.7.1.1. SOCKETS

DEFINIZIONE SOCKETS

Le sockets del dominio internet in C fanno affidamento su una struttura utilizzata per definire gli indirizzi e le porte degli host mittente e ricevente.

```
#include <netinet/in.h>
#include <arpa/inet.h>           // for inet_addr()
#include <sys/types.h>           // some custom types are defined here

struct sockaddr_in {
    short      sin_family;       // family of the address, typically set to AF_INET (IPv4)
    unsigned short sin_port;     // port number, e.g. htons(3490)
    struct in_addr sin_addr;     // see struct in_addr below
    char       sin_zero[8];      // typically zeros, now this structure has same size as sockaddr and can be casted to it
};

struct in_addr {
    unsigned long s_addr;        // IP address, can be loaded with inet_addr() or set to INADDR_ANY for localhost
};
```

CREAZIONE DEI SOCKETS

In C i sockets vengono creati mediante la funzione socket()

```
#include <sys/socket.h>
int sockfd = socket(int domain, int type, int protocol);
```

Dove:

- Domain: specifica la famiglia come abbiamo visto nella precedente struttura (AF_INET)
- Type: specifica il tipo di socket che viene creato.
 - o SOCK_STREAM -> TCP socket
 - o SOCK_DGRAM -> UDP socket
- Protocol: specifica un particolare protocollo che va utilizzato all'interno della socket (spesso lasciato a 0, indicando che nessuno specifico protocollo viene utilizzato)
- Sockfd: è un descrittore di file che indica la nuova socket appena creata.

Nota: anche le sockets seguono la filosofia di Unix del “tutto è un file” pertanto dalle sorgenti input e output tali sockets vengono trattate al pari di file.

SOCKETS: BINDING

Si può associare una socket ad una porta e indirizzo locale usando la funzione bind():

```
#include <sys/socket.h>
int val = bind(int socket, const struct sockaddr *address, socklen_t address_len);
```

Dove:

- Socket: è il file descrittore della socket che si vuole collegare
- Address: è un puntatore ad una struttura sockaddr che specifica con quale porta e indirizzo legare la socket (address.sin_addr è spesso settato a INADDR_ANY così che le connessioni provenienti da tutti gli indirizzi vengono ascoltate)
- Address_len: specifica la lunghezza della struttura sockaddr puntata dall'argomento dell'indirizzo. (si può utilizzare la funzione sizeof() per ottenerlo)
- Val: valore di ritorno che è pari a 0 in caso di successo e -1 altrimenti.

È importante sapere che questa funzione è utilizzata sui server perché è necessario collegare la socket ad una specifica porta ma è opzionale per quanto riguarda i client.

SOCKETS: SEND AND RECEIVE (UDP)

I processi di invio o ricezione dei messaggi in UDP sono eseguiti dalle funzioni sendto() e recvfrom():

```
#include <sys/socket.h>
int ob = sendto(int osock, const void *obuf, size_t olen, int oflags, const struct sockaddr *oaddr, socklen_t oaddr_len);
#include <sys/socket.h>
int ib = recvfrom(int isock, void *ibuf, size_t ilen, int iflags, struct sockaddr *iaddr, socklen_t *iaddr_len);
```

Dove:

- Osock/isock: sono i file descrittori su cui inviare/ricevere un messaggio.
- Obuf/ibuf: sono i puntatori ai buffer che contengono il messaggi da inviare o ricevere.
- Olen/ilen: sono le lunghezze dei messaggi espressi in bytes.
- Oflags/iflags: specifica le flags (tipicamente il loro valore è uguale a 0 oppure settato a MSG_WAITALL per attendere fino a quando tutti i bytes di olen/ilen sono stati inviati o ricevuti)
- Oaddr/iaddr: puntatori a strutture sockaddr contenenti gli indirizzi di ricezioni o invio
- Oaddr_len/iaddr_len: la lunghezza espressa in bytes della struttura sockaddr.
- Ob/ib: numero di bytes che sono stati attualmente inviati o ricevuti.

SOCKETS: CLOSING

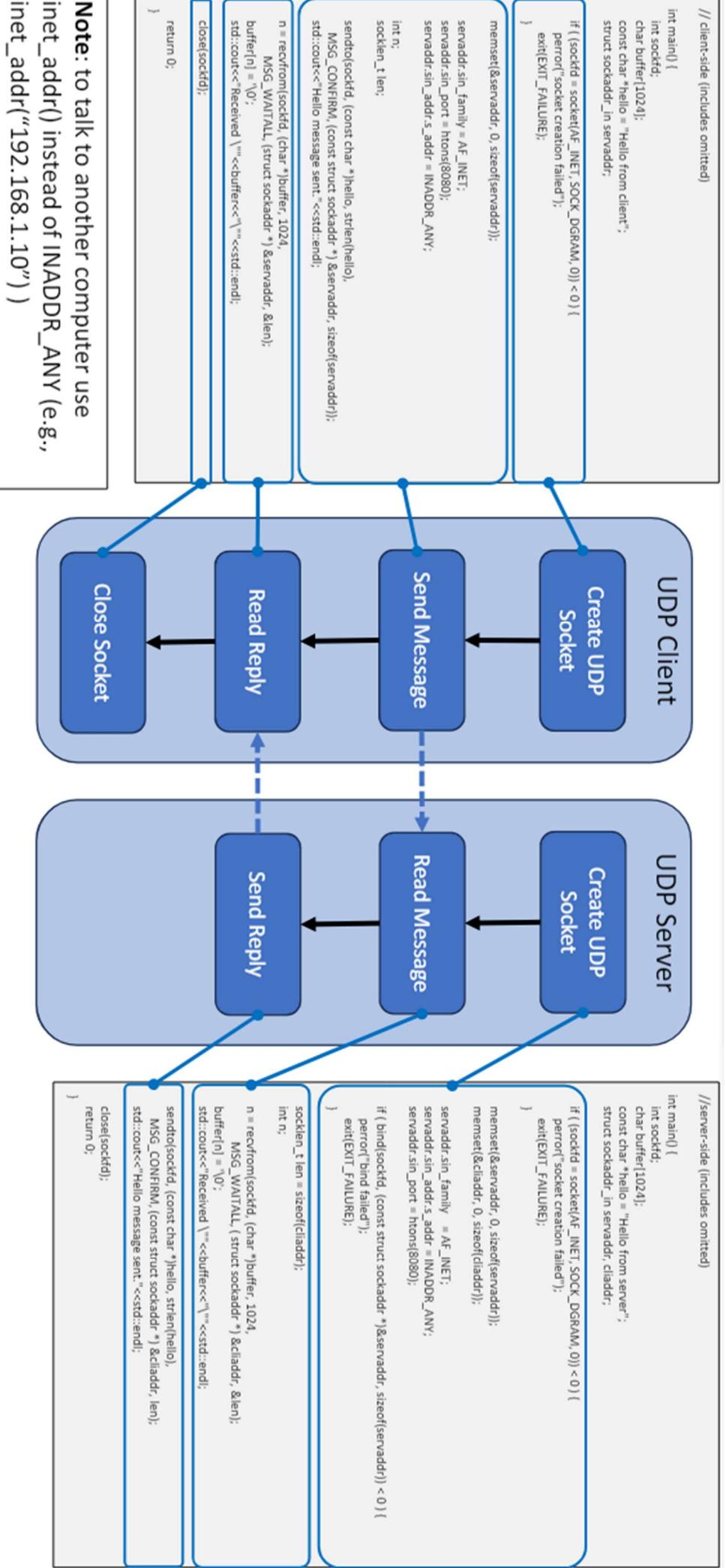
Le sockets possono essere chiuse mediante la funzione close():

```
#include <unistd.h>
int val = close(int socket);
```

Dove:

- Socket: è il file descrittore della socket da chiudere
- Val: è il valore di ritorno che è pari a 0 in caso di successo o a -1 altrimenti

PROGRAMMAZIONE UDP SOCKET (C/C++)



DA UDP A TCP

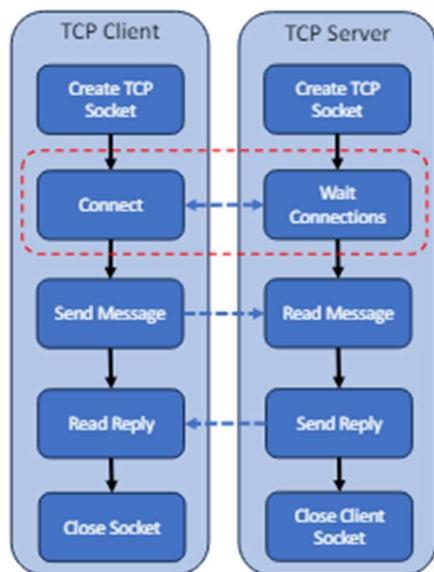
A differenza di UDP, in TCP client e server devono eseguire una handshake prima che i messaggi possano essere trasmessi.

In questo caso vengono utilizzate due sockets dal lato del server:

- 1. Una welcoming socket che è sempre attiva e permette ai clients di eseguire handshakes con il server
- 2. Una socket client-specific che viene creata dopo l'esecuzione dell'handshake ed è utilizzata da client e server per comunicare

La handshake a tre vie che viene eseguito all'interno del livello di trasporto risulta completamente invisibile ai programmi client e server.

Una volta che la connessione è stata accettata dal server (handshake conclusa con successo), la comunicazione si sposta sulla socket appena creata.



SOCKETS: CONNESSIONE (solo TCP)

La connessione TCP client-side è eseguita mediante la funzione connect():

```
#include <sys/socket.h>
int val = connect(int socket, const struct sockaddr *address, socklen_t address_len);
```

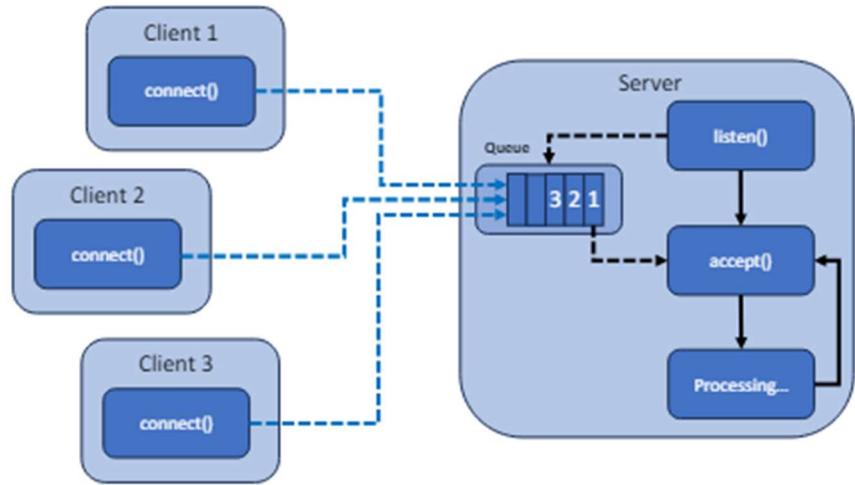
Dove:

- **Socket:** è il file descrittore della socket che si usa per connettersi
- **Address:** è un puntatore ad una struttura sockaddr contenente l'indirizzo del server
- **Address_len:** specifica la lunghezza della struttura sockaddr puntata dall'argomento dell'indirizzo (possiamo utilizzare sizeof() per ottenerla)
- **Val:** valore di ritorno che è pari a 0 in caso di successo o a -1 altrimenti.

SOCKETS: WAIT-FOR-CONNECTION (solo TCP)

Il wait-for-connection TCP server-side è eseguito mediante due funzioni che sono `listen()` e `accept()`, le quali lavorano in combinazione con la funzione per i clients `connect()`.

- 1. Listen(): apre una coda dove le connessioni in arrivo sono immagazzinate
- 2. Accept(): apre una socket client-specific.



```
#include <sys/socket.h>
int val = listen(int socket, int backlog);
int new_sockfd = accept(int socket, struct sockaddr *address, socklen_t *address_len);
```

Dove:

- Socket: è il file descrittore della socket su cui si attendono le connessioni (devono essere collegate a indirizzo e porta)
- Backlog: è la lunghezza massima della coda delle connessioni pendenti
- Address: può essere un puntatore nullo, o un puntatore ad una struttura `sockaddr` dove l'indirizzo della socket che si sta connettendo dovrebbe essere restituito.
- Address_len: è la lunghezza dell'indirizzo
- New_sockfd: è il descrittore della nuova socket su cui il client e il server comunicheranno
- Val: è il valore di ritorno che è pari a 0 in caso di successo o a -1 altrimenti

SOCKETS: SEND E RECEIVE (TCP)

I processi di invio e ricezione di messaggi in TCP vengono eseguiti dalle funzioni send() e read() (più semplici di quelle di UDP)

```
#include <sys/socket.h>
int ob = send(int osock, const void *obuf, size_t olen, int flags);

#include <unistd.h>
int ib = read(int isock, void *ibuf, size_t ilen);
```

Dove :

- Osock/isock: sono i file descrittori si cui viene inviato o ricevuto il messaggio
- Obuf/ibuf: sono i puntatori ai buffers contenenti il messaggio da inviare o ricevere
- Olen/ilen: sono le lunghezze dei messaggi espresse in bytes
- Flags: specifica il tipo di trasmissione del messaggio (dipende dal protocollo tipicamente settato a 0)
- Ob/ib: numero di bytes che sono stati effettivamente mandati o ricevuti.

È importante ricordare che gli indirizzi di client e server sono già stati specificati durante la fase di connessione. Non vi è quindi la necessità di farlo qui come in sendto/recvfrom.

Application Layer

TCP Socket Programming (C/C++)

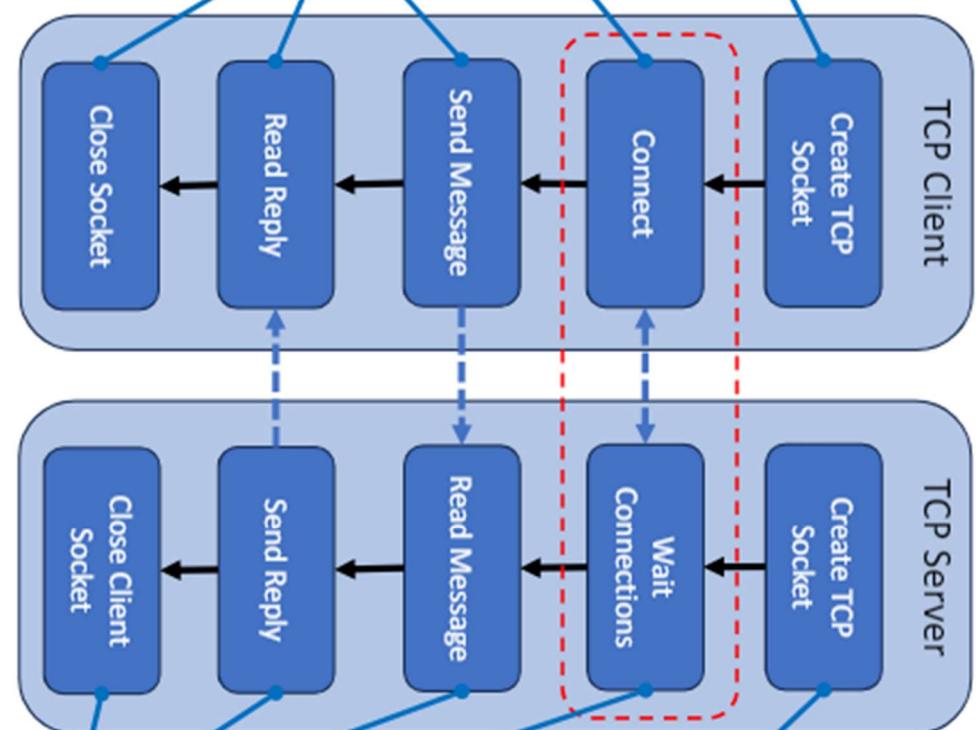
```
// client-side [includes omitted]
int main() {
    int sockfd, status;
    int opt = 1;
    char buffer[1024];
    const char *hello = "Hello from client";
    struct sockaddr_in servaddr;
    struct sockaddr_in clisaddr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n;
    if ((status = connect(sockfd, (struct sockaddr *)&servaddr,
        sizeof(servaddr))) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    send(sockfd, hello, strlen(hello), 0);
    std::cout << "Hello message sent." << std::endl;
    if (read(sockfd, buffer, 1024) == -1) {
        std::cout << "Received \\" << buffer << "\\n";
    }
    close(sockfd);
}
```



```
// server-side [includes omitted]
int main() {
    int sockfd, new_socket;
    int opt = 1;
    char buffer[1024];
    const char *hello = "Hello from server";
    struct sockaddr_in servaddr, clisaddr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
        sizeof(opt)) != 0) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(sockfd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    int addrlen = sizeof(clisaddr);
    if ((new_socket = accept(sockfd, (struct sockaddr *)&clisaddr,
        &addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    int n;
    if (read(new_socket, buffer, 1024) == -1) {
        std::cout << "Received \\" << buffer << "\\n";
    }
    send(new_socket, hello, strlen(hello), 0);
    std::cout << "Hello message sent." << std::endl;
    close(new_socket);
    close(sockfd);
}

return 0;
}
```

Note: to talk to another computer use
inet_addr() instead of INADDR_ANY (e.g.,
inet_addr("192.168.1.10"))

COMANDI UDP SOCKET

Per la connessione UDP è necessario che sia il client che il server aprano una socket.

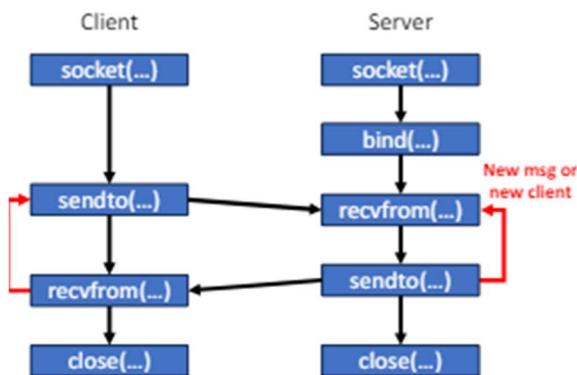
Il server deve collegare la socket ad una specifica porta.

Successivamente i messaggi verranno scambiati utilizzando le funzioni sendto e recvfrom.

- Qui l'IP e la porta del client vengono recuperate attraverso la funzione recvfrom
- È possibile mandare in loop le funzioni sendto e recvfrom per mantenere attiva la comunicazione

Quando la comunicazione è terminata, entrambi gli hosts chiudono la socket.

- È da notare che se è soltanto il client a chiudere la socket, il server non accetterà mai altri clients



I COMANDI TCP SOCKET

Per la connessione TCP vi è bisogno che sia il client che il server aprano una socket.

Il server deve collegare la socket ad una specifica porta, mettersi in ascolto ed eventualmente accettare una nuova connessione. Il client deve connettersi al server.

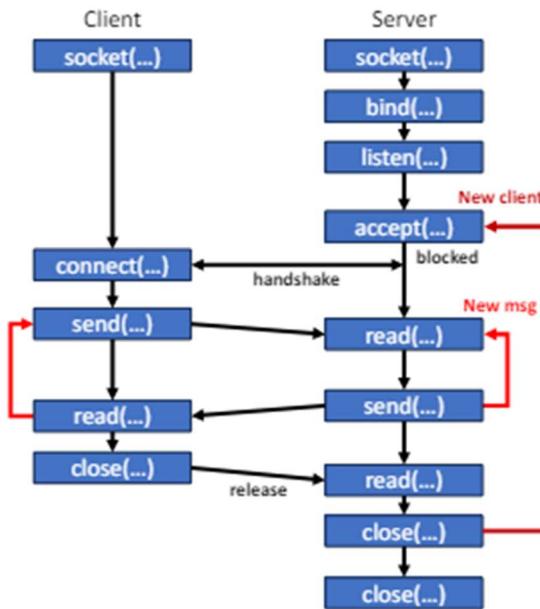
Successivamente è possibile scambiare messaggi usando le funzioni send e read.

- Qui l'IP e la porta del client vengono recuperate dalla socket nuovamente creata
- È possibile mandare in loop le funzioni send o read per mantenere attiva la comunicazione.

Quando la comunicazione è terminata, entrambi gli host chiudono la socket.

- È importante sapere che il server dovrebbe eseguire un read addizionale prima di chiudere la socket client-specific (rilascio della connessione) altrimenti bisogna attendere circa 4 minuti prima di riutilizzare la porta.

Il server dovrebbe inoltre tornare alla funzione accept per accogliere un nuovo client.



TRADUZIONE DNS (C/C++)

Come appena mostrato, si possono creare sockets tra i due hosts, ma è necessario conoscere l'indirizzo IP e la porta del server.

In internet di solito gli hosts sono identificati tramite gli hostnames piuttosto che con gli indirizzi IP.

I Berkeley sockets lavorano solo con gli indirizzi IP.

- Ciò non sorprende in quanto i sockets lavorano a livello di trasporto, mentre gli hostnames lavorano a livello di applicazione.

Se è necessario connettersi con un host tramite un hostname, è necessario utilizzare la traduzione DNS ed ottenere l'indirizzo IP associato da passare al socket.

C/C++ offre alcune funzioni e strutture che eseguono tali traduzioni per noi, che vengono usate per contattare i servers DNS e per recuperare gli indirizzi.

In particolare, si può usare la funzione `gethostbyname()` per contattare i servers DNS.

La suddetta funzione non restituisce solo l'indirizzo IP ma restituisce una struttura `hostent` che contiene alcune informazioni sull'host.

- Nome canonico
- Possibili aliases
- Uno o più indirizzi

La struttura hostent è definita come segue:

```
#include <netdb.h>

struct hostent {
    char * h_name;           // original name of the host (canonical)
    char ** h_aliases;       // list of aliases (terminated by a NULL pointer)
    int h_addrtype;          // family of the address (typically AF_INET)
    int h_length;            // length of the address (typically 4 bytes)
    char ** h_addr_list;     // list of addresses (terminated by a NULL pointer)
};

// for compatibility reasons
#define h_addr h_addr_list[0]
```

Un DNS può fornire una lista di indirizzi (i quali potrebbero essere ordinati in modo casuale se la query è inviata ad un server DNS round-robin).

- È possibile ottenere l'indirizzo del primo elemento della riga mediante `h_addr_list[0]` oppure `h_addr`.

Gli indirizzi restituiti possono essere assegnati alla struttura `in_addr` utilizzata dalle sockets.

La funzione `gethostbyname()` è definita come segue:

```
#include <netdb.h>

struct hostent *host_info = gethostbyname(const char *name);
```

dove:

- Name: è una stringa che contiene l'hostname che deve essere tradotto
- Host_info: è un puntatore alla struttura hostent che contiene le informazioni sull'host. Il suo valore è pari a NULL se vi è un errore.

È importante sapere che vi è anche una funzione simile chiamata `gethostbyaddr()` che restituisce la struttura hostent per uno specifico indirizzo IP.

```
#include <iostream>
#include <cstring>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>
```

- Include librerie per permettere la traduzione DNS.

```

int main(int argc, char **argv){
    struct hostent *server_info;
    char *server_name;

    if(argc < 2){
        std::cout << "No hostname specified" << std::endl;
        return 0;
    }
    else {
        server_name = argv[1];
    }

    server_info = gethostbyname(server_name);
    if (server_name == NULL){
        std::cout << "could Not resolve hostname " << server_name << ":" << std::endl;
        return 0;
    }
}

```

- Inizializzazione
- Ottenere hostname come un argomento
- Invocare DNS e verificare se una risposta è stata ricevuta.

```

//plot output
std::cout << "Canonical name:" << std::endl;
std::cout << "\t" << server_info->h_name << std::endl;

std::cout << "Aliases:" << std::endl;
int i = 0;
while(server_info->h_aliases[i] != NULL){
    std::cout << "\t" << server_info->h_aliases[i] << std::endl;
    i++;
}

std::cout << "IPs:" << std::endl;
i = 0;
while(server_info->h_addr_list[i] != NULL){
    std::cout << "\t" << inet_ntoa( *( (struct in_addr *) server_info->h_addr_list[i] ) ) << std::endl;
    i++;
}

return 0;
}

```

- Tracciamento dell'output dalla struttura hostent:
 - Nome canonico
 - Lista di aliases
 - Lista di IP

ESEMPIO DI SOCKET HTTP

Si può creare un codice C++ implementando da zero una richiesta http (HEAD) per una pagina web.

La pagina web che controlleremo è la pagina dal sito web UNINA “cenni storici”:

<http://www.unina.it/chi-siamo/cenni-storici>

```
#include <iostream>
#include <cstring>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>
```

- Include librerie per permettere la connessione TCP al web server.

```
int main(){
    int socket_desc;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];

    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc < 0){
        std::cout << "failed to create socket" << std::endl;
        return 0;
    }
```

- Inizializzazione delle variabili e creazione della socket TCP. Per una richiesta http 3 elementi sono specificati:
 - o L'hostname del webserver
 - o Il numero di porta
 - o La risorsa che deve essere recuperata (il percorso dell'oggetto)

```
server = gethostbyname("www.unina.it");
if (server == NULL){
    std::cout << "could Not resolve hostname :(" 
    << std::endl;
    close(socket_desc);
    return 0;
}

bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(80);
bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);

if (connect(socket_desc, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0){
    std::cout << "connection failed :(" << std::endl;
    close(socket_desc);
    return 0;
}
```

- Connessione al web server. Qui utilizziamo la seguente funzione:
server = gethostbyname(host.c_str()); che invoca DNS e ottiene l'indirizzo IP del server dall'hostname (posto nella struttura hostent chiamata server)
- L'IP dalla struttura ritornata viene copiato nella struttura serv_addr per la seguente funzione di connessione.

```
const char *request = "HEAD /chi-siamo/cenni-storici HTTP/1.1\r\nHost: www.unina.it\r\nConnection: close\r\n\r\n";  
  
if (send(socket_desc, request, strlen(request), 0) < 0){  
    std::cout << "failed to send request..." << std::endl;  
    close(socket_desc);  
    return 0;  
}  
std::cout<<"message sent:"<<std::endl;  
std::cout<<request<<std::endl;  
  
int n = recv(socket_desc, buffer, sizeof(buffer), 0);
```

- Scambio di messaggi con il web server. La richiesta http è definita nella stringa di richiesta.
- Qui si crea una connessione non persistente in quanto si vuole solo controllare una singola pagina.

```
std::cout<<"received "<<n<<" bytes:"<<std::endl;  
std::cout<<buffer;  
  
close(socket_desc);  
  
return 0;
```

- Chiudere la socket e tracciare la pagina ricevuta (HTML).

5. LIVELLO DI TRASPORTO

5.1 DALL'APPLICAZIONE AL TRASPORTO

Un protocollo a livello di trasporto per lo più fornisce una comunicazione logica tra i processi applicativi eseguiti su differenti hosts.

- Dal punto di vista dell'applicazione, gli hosts che eseguono i processi sembrano direttamente connessi come se fossero sulla stessa macchina (anche se sono su parti opposte del pianeta).

Il livello di trasporto converte i messaggi del livello applicativo in uno o più pacchetti di livello di trasporto chiamati segmenti o datagrammi (gli ultimi sono principalmente usati per i pacchetti UDP)

- Se i messaggi di applicazione sono suddivisi in chunks più piccoli (segmenti o datagrammi), ogni chunk è fornito con un header a livello di trasporto.
- I segmenti vengono passati uno per uno al livello network per poi essere trasmessi.

5.2 RESPONSABILITA' DEL LIVELLO DI TRASPORTO

I protocolli a livello di trasporto (UDP e TCP) hanno quattro principali responsabilità:

- 1. Process-to-process delivery: i messaggi vengono consegnati indipendentemente da dove si trovano questi processi. È da notare che ciò è differente dall' host-to-host delivery, il quale è di responsabilità del protocollo IP di livello inferiore.
- 2. Integrity checking: includendo i campi di rilevamento degli errori negli headers dei segmenti.
- 3. Reliable data transfer: assicurandosi che i dati vengano consegnati dal processo di invio al processo di ricezione, correttamente ed in ordine.
- 4. Congestion/flow control: impedisce alla connessione di inondare i dispositivi di rete (links o routers) con una quantità eccessiva di traffico. Questo servizio è utile per migliorare le prestazioni ed è molto beneficio per l'intero network di internet.

In particolare, UDP (più veloce ma più semplice) fornisce solo i primi due servizi, mentre il servizio TCP fornisce tutti i quattro.

5.2.1. PROCESS-TO-PROCESS DELIVERY

A livello applicativo, i diversi processi possono accedere alla network allo stesso tempo.

Un processo può anche essere connesso a multipli processi allo stesso tempo:

- Per esempio proxy servers e mail servers possono essere connessi a molteplici processi remoto.

Per risolvere questi problemi, si utilizzano le sockets per il process-to-process delivery:

- Anziché consegnare il messaggio direttamente ad uno specifico processo, si usano sockets come end-points dai quali i processi ottengono i messaggi.

- Questo approccio in qualche modo disaccoppia il numero di processi dal numero delle connessioni.

Dal momento che vi sono molteplici sockets nell'host ricevente, vi è bisogno di:

- Un identificatore univoco per ogni socket (porta) che deve essere associato ai segmenti ricevuti.
- Un processo di multiplexing/demultiplexing sul mittente/destinatario.

Demultiplexing: è il processo di reindirizzare un segmento alla socket giusta a seconda dell'identificatore associato.

Multiplexing: è il processo di creare segmenti da differenti processi, assegnando loro il giusto identificatore.

È importante notare che il problema di reindirizzare messaggi a molteplici sorgenti è abbastanza comune nelle network di computer, e multiplexing e demultiplexing sono anche presenti in altri strati (non solo in quello di trasporto).

5.3 PORTE

Gli identificatori delle sockets sono chiamati porte di sorgente e porte di destinazione (o semplicemente porte)

- Una porta è un numero a 16 bit che va da 0 a 65535.
- I numeri di porta che vanno da 0 a 1023 sono definiti numeri di porta ben noti e sono riservate.

Quando si sviluppa una nuova applicazione network, si devono assegnare i numeri di porta alle sockets (e quindi alle applicazioni) di conseguenza.

Port	Usage
20	File Transfer Protocol (FTP) Data Transfer
21	File Transfer Protocol (FTP) Command Control
22	Secure Shell (SSH)
23	Telnet - Remote login service, unencrypted text messages
25	Simple Mail Transfer Protocol (SMTP) E-mail Routing
53	Domain Name System (DNS) service
80	Hypertext Transfer Protocol (HTTP) used in World Wide Web
110	Post Office Protocol (POP3) used by e-mail clients to retrieve e-mail from a server
119	Network News Transfer Protocol (NNTP)
123	Network Time Protocol (NTP)
143	Internet Mail Access Protocol (IMAP) Management of Digital Mail
161	Simple Network Management Protocol (SNMP)
443	HTTP Secure (HTTPS) HTTP over TLS/SSL

5.3.1 PORTE: NMAP

Per verificare l'utilizzo di una porta (e anche altro) sulle macchine Linux si può usare il comando nmap.

Nmap (Network Mapper) è un utility di scansione di rete che sostanzialmente è stata creata per mappare (scoprire) gli hosts e i servizi all'interno della network.

Può essere utilizzato per sondare le porte degli hosts per rilevare quelle aperte (su cui una applicazione è in ascolto) e possibilmente i servizi associati.

- Usage:

- Port-scan of a given target:
 - \$ sudo nmap [target address]
- Probe port to determine services:
 - \$ sudo nmap -sV [target address]
- Check first N top used ports:
 - \$ sudo -top-port [N] [target address]

```
hargalaten@hargalaten-Lenovo-850-80:~$ nmap -top-port 10 www.google.com
Starting Nmap 7.80 ( https://nmap.org ) at 2023-10-13 15:27 CEST
Nmap scan report for www.google.com (142.250.180.132)
Host is up (0.021s latency).
Other addresses for www.google.com (not scanned): 2a00:1450:4002:809::2004
DNS record for 142.250.180.132: mil04s43-in-f4.1e100.net

PORT      STATE      SERVICE
21/tcp    filtered  ftp
22/tcp    filtered  ssh
23/tcp    filtered  telnet
25/tcp    filtered  sntp
80/tcp    open       http
110/tcp   filtered  pop3
139/tcp   filtered  netbios-ssn
443/tcp   open       https
445/tcp   filtered  microsoft-ds
3389/tcp  filtered  ms-wbt-server

Nmap done: 1 IP address (1 host up) scanned in 1.33 seconds
hargalaten@hargalaten-Lenovo-850-80:~$
```

Note: state can be open or closed, filtered or unfiltered (which means unable to be scanned).

5.4 MULTIPLEXING E DEMULTIPLEXING

Assumiamo che abbiamo un processo nell'host A (porta 19157) che vuole inviare un messaggio al processo (porta 46428) nell'host B.

Multiplexing:

- Il livello trasporto nell'host A crea un segmento o datagramma che include i dati dell'applicazione, il numero della porta della sorgente (19157) e il numero della porta di destinazione (46428).
- Il livello trasporto poi passa il segmento o datagramma risultante al livello network che lo incapsula in un datagramma IP, che farà del suo meglio per cercare di consegnare il segmento all'host ricevente.

Demultiplexing:

- Se il segmento o datagramma arriva all'host B, il livello trasporto lo riceve e lo decapsula.
- Il livello trasporto poi passa il segmento o datagramma alla socket appropriata esaminando il numero di porta del destinatario del segmento.

5.4.1 MULTIPLEXING/DEMULTIPLEXING IN UDP (C/C++)

In un client UDP in C++, la socket viene creata come segue:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

La socket tipicamente non è collegata ad una specifica porta sorgente, è il sistema operativo ad occuparsi di selezionarne una libera, ma possiamo sempre collegarla ad una specifica porta.

L'indirizzo o porta di destinazione è settata passando una struttura sockaddr_in adatta alla funzione sendto:

```
servaddr.sin_port = htons(19157);
servaddr.sin_addr.s_addr = inet_addr( address_of_B );
...
sendto(sockfd, (const char *)msg, strlen(msg), 0,
        (const struct sockaddr *)&destaddr,
        sizeof(destaddr));
```

In un server UDP in C++, la socket è creata e collegata come segue:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
...
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(19157);
bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));
```

La porta/indirizzo di destinazione viene recuperata attraverso la struttura sockaddr_in dalla funzione recvfrom ed usata nel messaggio di risposta:

```
recvfrom(sockfd, (char *)msg, 1024, 0,
          ( struct sockaddr * ) &cliaddr, &len);
...
sendto(sockfd, (const char *)msg, strlen(msg), 0,
        (const struct sockaddr * ) &cliaddr, sizeof(cliaddr));
```

La risposta verrà inviata indietro qualunque sia il client.

Si può dire che una socket UDP può essere identificata da due principali elementi:

- 1. Indirizzo IP di destinazione
- 2. Porta di destinazione

Infatti, è possibile utilizzare la medesima socket per inviare un messaggio di ritorno alle molteplici porte o indirizzi sorgente.

Nell'esempio, la struttura cliaddr viene riempita dalla funzione recvfrom, di modo tale che è possibile utilizzarla come indirizzo di destinazione per la successiva funzione sendto, indipendentemente da chi è l'host.

5.4.2. MULTIPLEXING/DEMULTIPLEXING IN TCP (C/C++)

Nella comunicazione TCP l'applicazione server dispone di un welcoming socket, che attende richieste di stabilimento di connessione dai clients su uno specifico numero di porta (19157 come prima)

Il client TCP crea una socket ed invia un segmento di richiesta di stabilimento di connessione all'host specificato nella struttura sockaddr_in (servaddr in questo caso):

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
...
servaddr.sin_port = htons(19157);
servaddr.sin_addr.s_addr = inet_addr( address_of_B );
...
connect(sockfd, (struct sockaddr*)&servaddr,
        sizeof(servaddr));
```

Una richiesta di stabilimento di connessione è solo un segmento TCP con un numero di porta di destinazione (in questo caso 19157) ed uno speciale bit di stabilimento di connessione settato in un header TCP (SYN = 1).

Quando il server riceve il segmento di richiesta di connessione in ingresso (con destinazione porta 19157), esso localizza il processo del server che è in attesa di accettare la connessione. Poi un nuovo socket viene creato:

```
new_socket = accept(sockfd,
                     (struct sockaddr*)&cliaddr, (socklen_t*)&addrlen)
```

Il livello di trasporto dal lato del server riempie la struttura (cliaddr) usando i numeri di porta e gli indirizzi dal segmento in ingresso e crea una nuova socket (new_socket)

Tutti i futuri segmenti aventi questa specifica porta di sorgente, specifico indirizzo IP sorgente, porta di destinazione, indirizzo IP di destinazione sono reindirizzati (demultiplexed) a new_socket.

Si può dire una socket TCP può essere identificata da quattro elementi:

- 1. Indirizzo IP della sorgente
- 2. La porta della sorgente
- 3. Indirizzo IP del destinatario
- 4. La porta del destinatario

A causa dell'iniziale handshake la connessione TCP intreccia i processi da entrambi i lati della socket.

Soltanto una sorgente e una destinazione useranno quella socket, se il lato client o il lato server delle comunicazioni viene interrotto, la socket viene chiusa per entrambi i lati (ciò non accade per UDP).

5.4.3 MULTIPLEXING/DEMULITPLEXING ESEMPIO HTTP

È possibile avere molteplici processi da una macchina in comunicazione con molteplici processi (o lo stesso processo) di un'altra macchina.

In questo esempio un host C inizializza due sessione http con il server B, e un host A inizializza una sessione http con il server B. Gli hosts A e C e il server B ognuno ha il proprio indirizzo IP univoco. L'host C assegna due differenti numeri di porta sorgente (26145 e 7532) alle sue due connessioni http.

L'host A può assegnare una porta sorgente numero 26145 alla propria connessione http (non essendo a conoscenza di C).

Questo non è un problema, siccome le due connessioni presentano indirizzi sorgenti IP differenti.

In questo esempio, il server apre un nuovo processo (o un nuovo thread) per ciascuna connessione in ingresso assegnandogli la socket specifica (new_socket).

Questo è un processo piuttosto comune ma aprire troppi processi o threads potrebbe impattare le prestazioni del server.

5.5 UDP

Come già spiegato, UDP è utile per connessioni semplici e rapide. Principalmente svolge due compiti:

- 1. Process-to-process delivery (porte, multiplexing, demultiplexing)
- 2. Integrity checking (controllo degli errori)

UDP è connectionless: non vi è alcun tipo di handshaking tra le entità mittente e ricevente a livello di trasporto prima di inviare il segmento.

UDP non fornisce alcuna garanzia riguardo la consegna dei messaggi (non stabilisce un data transfer affidabile) e non ha alcun controllo del flusso e congestione.

UDP è un protocollo minimalista che aggiunge giusto l'essenziale riguardo al protocollo IP del livello più basso.

Perché usare UDP al posto di TCP?

- Controllo a livello applicativo: UDP è più diretto quindi l'applicazione ha un maggiore controllo sulla trasmissione. Questo può essere utile per esempio nel caso delle applicazioni in real-time dove la velocità di invio o i ritardi sono cruciali e l'eventuale perdita dei dati tollerata, pertanto un minor controllo è preferibile.
- Stabilimento di connessione rapida: non vi è alcuna handshake, quindi non vi è alcun ritardo nello stabilire una connessione. Ciò è buono per esempio nel caso della DNS per non generare ulteriori ritardi.

- Nessun connection state: non ci sono parametri per il controllo della congestione, né numeri di sequenza né numeri di acknowledgment. Un server può tipicamente supportare molti più clients attivi quando l'applicazione viene eseguita mediante UDP piuttosto che mediante TCP.
- Sovraccarico minimo dei pacchetti: il segmento TCP aggiunge 20 bytes di header line in ogni segmento, mentre UDP ne aggiunge solo 8 bytes.

5.5.1. ESEMPIO DNS

Un esempio di applicazione che utilizza connessione UDP è DNS. Il client DNS funziona come segue:

- Nel livello applicazione: quando un host vuole effettuare una query al server DNS, esso costruisce un messaggio query DNS e passa il messaggio ad UDP.
- Nel livello trasporto: senza eseguire alcune handshake con il server DNS, UDP aggiunge campi header al messaggio e passa il segmento risultante al livello network.
- Nel livello network: il segmento UDP è incapsulato in un datagramma IP ed inviato al server DNS (attraverso il livello di accesso)
- Il client a questo punto attende una risposta alla sua query. Se la risposta non viene ricevuta (possibilmente perché la suddetta rete ha perso la query o la risposta), potrebbe attuare differenti approcci:
 - Reinviare la query
 - Inviare la query ad un altro name server
 - Informare l'applicazione invocante che la risposta non è stata ricevuta

5.5.2. UDP: UTILIZZO

Le applicazioni che hanno bisogno di una data transfer affidabile quali ad esempio e-mail, accesso a terminali remoti o il web, utilizzano TCP.

In questi casi non è concessa perdita di pacchetto, per esempio le mails devono essere completamente inviate e non possono avere elementi mancanti.

SNMP utilizza UDP per la gestione dei dispositivi. Le applicazioni di gestione del network devono spesso essere eseguite quando la network è in uno stato di stress (precisamente quando il data transfer affidabile è difficile da attuare).

Le applicazioni di multimedia come streaming, video conference ecc spesso utilizzano sia UDP che TCP perché una piccola quantità di perdita di pacchetto è tollerata.

In generale, le applicazioni real-time reagiscono molto male al controllo di congestione di TCP.

Service	Application Protocol	Transport Protocol
E-mails	SMTP	TCP
Remote terminal	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Name translation	DNS	UDP
Network device management	SNMP	UDP
Multimedia streaming	Proprietary	UDP or TCP
Internet telephony	Proprietary	UDP or TCP

Dall’altro lato, eseguire applicazioni multimedia mediante UDP è problematico perché non viene eseguito alcun controllo di congestione.

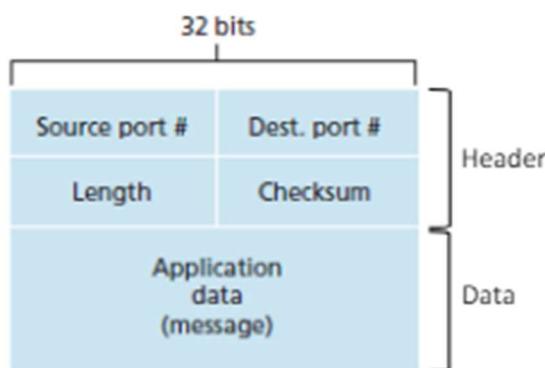
Se tutti volessero egoisticamente iniziare ad eseguire lo streaming di video a velocità elevata senza utilizzare alcun tipo di controllo della congestione, i dispositivi network potrebbero andare in sovraccarico causando una maggiore perdita dei pacchetti UDP.

Tassi di perdita elevati indotti da mittenti UDP senza controllo potrebbero far sì che anche i mittenti di TCP abbiano un drammatico calo delle loro velocità.

5.5.3. UDP: FORMATO DEL SEGMENTO

Il segmento UDP (o datagramma per essere più precisi) è composto da 5 elementi:

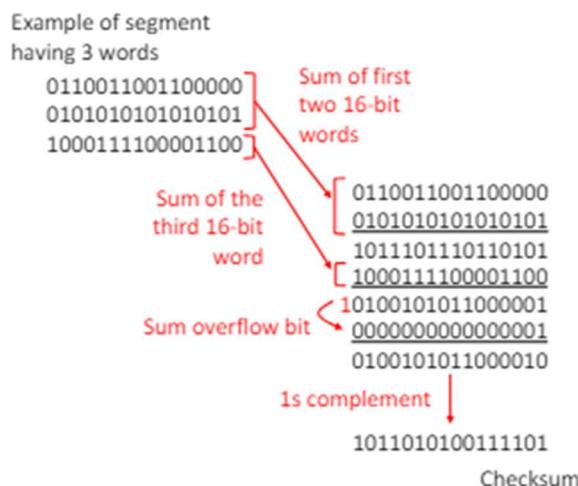
- UDP header da 64 bits che include informazioni relative al protocollo UDP ed è composto a sua volta da 4 elementi da 16 bits ciascuno:
 - o 1. Source port: numero della porta del processo mittente
 - o 2. Destination port: numero della porta del processo destinatario
 - o 3. Length: lunghezza dell’intero datagramma (header+data)
 - o 4. Checksum: usato dal mittente per verificare se il messaggio è intatto
- 5. Data (N bits) che è il messaggio concreto proveniente dal livello applicativo.



5.5.4. UDP: CHECKSUM

Il controllo dell'integrità è eseguito mediante checksum. Il checksum UDP è un file di 16 bites usato per l'individuazione degli errori come segue:

- UDP dal lato del mittente esegue il complemento a 1 della somma di tutte le parole a 16-bit presenti nel datagramma, con ogni bit di overflow che viene sommato ad esso.
- Questo risultato è posto nel campo checksum del datagramma UDP.
- Quando il ricevente somma tutti i bits all'interno del messaggio compreso il checksum, la somma deve essere 16 volte uno.
- Se anche solo un bit è pari a 0, sappiamo che si sono verificati degli errori nel pacchetto.



5.6 DATA TRANSFER AFFIDABILE

Il controllo degli errori (ad esempio attraverso checksum) permette al ricevente di comprendere almeno se il messaggio è corrotto, ma questo non è abbastanza. Il data transfer affidabile è ancora uno dei principali problemi della networking.

- Il problema non è solo indirizzato a livello di trasporto, ma anche a livello link e spesso a livello applicativo.

In un canale affidabile tre elementi devono essere garantiti:

- 1. Nessuno dei bits trasmessi devono essere corrotti (capovolti da 0 a 1 o viceversa)
- 2. Nessuno dei bits trasmessi deve essere perso o ripetuto.
- 3. Tutti i bits sono consegnati nell'esatto ordine in cui vengono inviati.

In generale, dobbiamo assumere che il livello inferiore network non è affidabile.

- Questa è un'assunzione realistica. Per esempio TCP è un protocollo di data transfer affidabile che è implementato al di sopra di un'inaffidabile livello network end-to-end.

5.6.1. DATA TRANSFER AFFIDABILE: CORRUZIONE DEI PACCHETTI E LO STOP-AND-WAIT

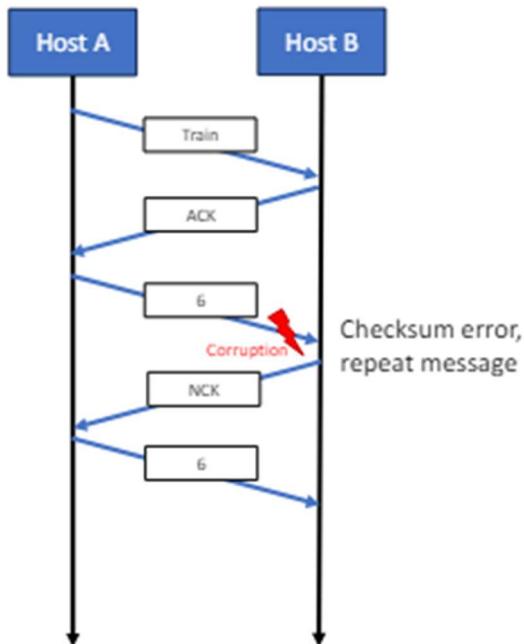
Assumiamo di avere un canale dove i bits possono essere soltanto corrotti e non persi.

- La corruzione dei bits è un problema ricorrente, che è principalmente dovuto alle componenti fisiche di una network attraverso cui il pacchetto è trasmesso, propagato o inserito in buffer più e più volte durante la comunicazione.

Stop-and-wait: un primo approccio richiede che ogni messaggio debba essere noto prima di mandarne uno nuovo:

- Positive acknowledgments (ACK) significa che il messaggio è stato ricevuto intatto.
- Negative acknowledgments (NCK) significa che si è verificato un errore quindi il messaggio deve essere ripetuto.

In una computer network, i protocolli basati sulla ritrasmissione sono anche conosciuti come protocolli ARQ (Automatic Repeat reQuest).



Ci sono un paio di problemi con questo approccio:

- 1. Cosa accade se il messaggio ACK/NCK è esso stesso corrotto?
- 2. Come può l'host B essere sicuro che un messaggio è una ripetizione piuttosto che un nuovo messaggio?

Una possibile soluzione è il sequence number: aggiunge un nuovo campo alle headers dei pacchetti che specifica l'ordine in cui i pacchetti dovrebbero essere ricevuti.

In un approccio stop-and-wait vi è sempre un messaggio sulla linea.

Non vi è bisogno di specificare l'intera sequenza dei pacchetti, ma è necessario soltanto differenziare tra il pacchetto corrente e quello precedente.

Perciò in questo caso vi è soltanto bisogno di aggiungere un bit ($s = 0/1$) agli headers dei pacchetti.

5.6.2. DATA TRANSFER AFFIDABILE: PERDITA DI PACCHETTO E LO STOP-AND-WAIT

Assumiamo di avere un canale dove i pacchetti possono anche essere persi.

- Ciò è anche abbastanza ragionevole in quanto i dispositivi network possono avere buffer in sovraccarico nel caso di traffico intenso.

Qui vi è un chiaro problema con l'approccio stop-and-wait: il loop. Se un messaggio è perso, gli hosts non ne invieranno mai uno nuovo.

- Da notare che questo accade sia se viene perso un messaggio sia se viene perso un acknowledgment, in entrambi i casi l'host A non sarà portato ad inviare un nuovo messaggio.

Timeout: una soluzione semplice ma efficiente è quella di aggiungere un timeout al lato del mittente che una volta scaduto permette all'host di riprovare.

- Un simile approccio può essere usato nei DNS su base UDP.

Questo funziona sia nel caso di perdita di pacchetto che in caso di perdita di acknowledgment.

- In caso di doppio il destinatario deve soltanto scartare il pacchetto.

La sfida qui è come stimare un timeout adatto (dipende dal RTT)

- Un timeout troppo lungo fa sì che la comunicazione rallenti.
- Un timeout troppo corto fa sì che i pacchetti si sovrappongano.

Si potrebbe dire che un timeout ragionevole debba essere in qualche modo più lungo del RTT.

Il timeout salva lo stop-and-wait dal loop, ma le prestazioni sono piuttosto pessime.

5.6.3 DATA TRANSFER AFFIDABILE: PRESTAZIONE DELLO STOP-AND-WAIT

Consideriamo il caso idealizzato di due host localizzati su coste opposte degli Stati Uniti.

La velocità della luce RTT tra questi due end systems è approssimativamente 30 millisecondi.

Assumiamo di avere:

- Un canale con una velocità di trasmissione (R) di 1 Gbps (10^9 bits per secondo)
- Una dimensione del pacchetto (L) di 1000 bytes (8000 bits)

Il tempo (t) necessario per trasmettere il pacchetto al canale è:

$$t = \frac{L}{R} = \frac{8000}{10^9} = 0.000008 \text{ (8 microseconds)}$$

Nel protocollo stop-and-wait:

- Il mittente comincia ad inviare un pacchetto in tempo t_0 ,
- L'ultimo bit entra nel canale a tempo $t_0 + 8$ microsecondi,
- Il pacchetto impiega 0.015 secondi per raggiungere il destinatario,
- L'ultimo bit viene ricevuto al tempo $t = \text{RTT}/2 + L/R = 0.015008$ secondi,
- Assumendo che i pacchetti ACK sono estremamente piccoli (il loro tempo di trasmissione può essere trascurato) l'ACK torna indietro al mittente in tempo $t = \text{RTT} + L/R = 0.030008$ secondi.

In 0.030008 secondi del tempo totale di trasmissione, il mittente stava in attesa quasi tutto il tempo (99.973% del tempo).

5.6.4 DATA TRANSFER AFFIDABILE: PRESTAZIONE DEL PIPELINING

Pipelining: invece dello stop-and-wait, al mittente è permesso di inviare pacchetti multipli senza aspettare gli acknowledgments.

Negli approcci pipelining (esempio 3 pacchetti invece di 1):

- Il mittente inizia inviando i 3 pacchetti al tempo t_0 ,
- L'ultimo bit dell'ultimo pacchetto entra nel canale al tempo $t_0 + 0.000024$ secondi,
- I pacchetti impiegano 0.015 secondi a raggiungere il destinatario,
- L'ultimo bit è ricevuto al tempo $t = \text{RTT}/2 + L/R = 0.015024$ secondi,
- Assumendo che i pacchetti ACK sono estremamente piccoli (il loro tempo di trasmissione può essere trascurato) l'ACK torna indietro al mittente al tempo $t = \text{RTT} + L/R = 0.030024$ secondi

Nei 0.030024 secondi il mittente stava aspettando 0.035% del tempo in meno (99.920% invece di 99.973%).

Negli approcci pipelining:

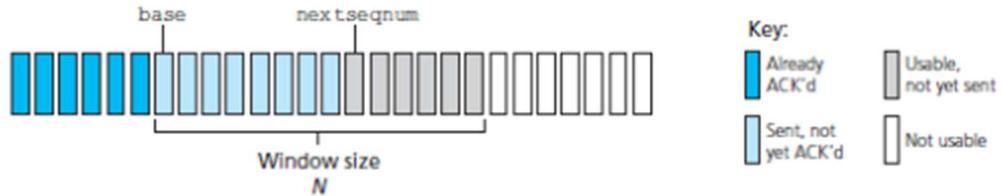
- Il range dei numeri di sequenza deve essere incrementato siccome ciascun pacchetto in transito (senza contare le ritrasmissioni) deve avere un unico numero di sequenza e possono esserci multipli pacchetti unacknowledged in transito.
- Saranno necessari buffers per immagazzinare i pacchetti in arrivo siccome non si sa se i pacchetti sono corretti o ci sono dei buchi nella trasmissione.

Vi sono due protocolli di base che utilizzano il pipelining:

- Go-Back-N
- Selective Repeat

5.6.4.1. GO-BACK-N

Il protocollo Go-Back-N (GBN): al mittente è permesso inviare pacchetti multipli senza attendere un acknowledgment ma è costretto ad avere non più di N pacchetti non acknowledged.



Base: è il numero di sequenza del più vecchio pacchetto non acknowledged.

Nextseqnum: è il più piccolo numero di sequenza non utilizzato (il prossimo ad essere inviato)

Abbiamo quindi che:

- Pacchetti da [0, base-1] sono stati trasmessi e acknowledged
- Pacchetti da [base, nextseqnum-1] sono stati inviati ma non ancora acknowledged
- Pacchetti da [nextseqnum, base+N-1] possono essere inviati
- Pacchetti da [base+N, +inf] non possono essere utilizzati fino a quando non viene ricevuto un nuovo acknowledgment.

Nel protocollo GBN il ruolo del destinatario è piuttosto semplice.

Il destinatario deve semplicemente scartare i pacchetti non in ordine (che siano danneggiati o meno) e consegnare a livello superiore (quello applicativo) solo i pacchetti in ordine.

- I pacchetti scartati (non acknowledged) saranno eventualmente ritrasmessi dal mittente.

Con questo approccio i pacchetti buoni sono anch'essi scartati ma il processo generale è piuttosto semplice:

- Il mittente deve mantenere gli indici della finestra
- Il destinatario ha soltanto bisogno di mantenere il numero di sequenza del nuovo pacchetto in ordine.

Certamente lo svantaggio di gettare via un pacchetto ricevuto correttamente è quello di doverlo rimandare di nuovo.

Questa è una reazione a catena, se si perdono pacchetti a causa delle difficoltà della network molti pacchetti correttamente ricevuti ma non in ordine potrebbero essere scartati forzando il mittente a rimandarli.

La ritrasmessione stessa potrebbe essere persa o danneggiata e così potrebbero essere richiesti persino ulteriori ritrasmessioni.

5.6.4.2. SELECTIVE REPEAT

Nonostante GNB sia più efficiente dello stop-and-wait, potrebbe soffrire di problemi di prestazione.

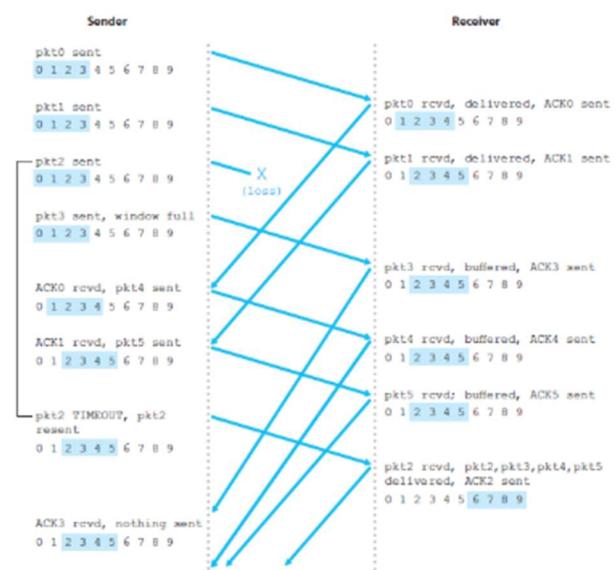
Considerando la larga dimensione della finestra, l'errore di un singolo pacchetto potrebbe far sì che GBN ritrasmetta un ingente numero di pacchetti, molti non necessari.

Nei protocolli selective-repeat (SR) il mittente ritrasmette soltanto quei pacchetti che sono probabilmente stati ricevuti in errore (persi o corrotti):

- Come in GBN i pacchetti individuali sono acknowledged
- Una finestra di dimensione N è nuovamente utilizzata per limitare il numero dei pacchetti non acknowledged
- A differenza di GBN, i pacchetti non in ordine vengono inseriti in dei buffers e acknowledged dal destinatario.

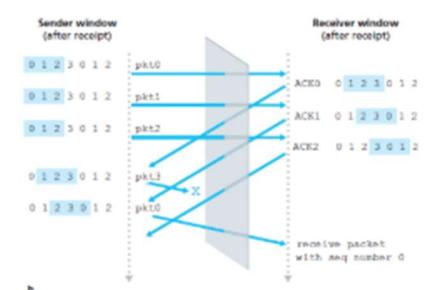
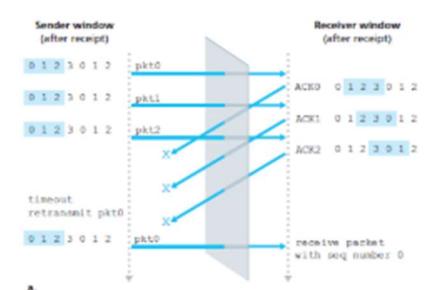
- Example of SR operation in the presence of lost packets: the receiver initially buffers pkt3, pkt4, and pkt5, while waiting for pkt2 (lost) to be retransmitted.

- Despite the previous example, here we avoid to resend pkt3, pkt4, pkt5, in so avoiding additional transmission loss or errors.



- One issue in SR is that the window size is related to the sequence number, **and sequence number is finite**.

- In this example we have four packets, **a max sequence number of 3** and a window size of 3.
- Let's assume packets 0 to 2 are correctly received and acknowledged, hence receiver's window moves to 6th packet (i.e., to packets 3, 0, 1):
 - **Case a:** the ACKs for the first three packets are lost and the sender retransmits these packets. *Is packet 0 new or old?*
 - **Case b:** the ACKs are received, packets 3 and 0 are sent but packet 3 is lost. *Is packet 0 new or old?*
- To avoid this issue, **the sequence number must be at least 2 times the window size**.



5.7 TCP

TCP rispetto a UDP è:

- Connection-oriented: c'è una fase di handshake prima della trasmissione che assicura che i due processi (mittente e destinatario) siano connessi.
- Affidabile: sono implementati il rilevamento degli errori, le ritrasmissioni, gli acknowledgments, i timers e i numeri di sequenza.
- Consapevole della congestione e del flusso: vi è una regolamentazione della velocità di trasmissione che dipende dalla prestazione del destinatario (flusso) e dalla prestazione del network (congestione)

L'essere connection-oriented rende TCP full-duplex e point-to-point:

- Full-duplex: se l'host A è connesso con B allora l'host B è connesso con A
- Point-to-point: singolo mittente e singolo destinatario.
Nota: UDP permette il multicasting, ci sono modi per trasferire dati a multipli host.

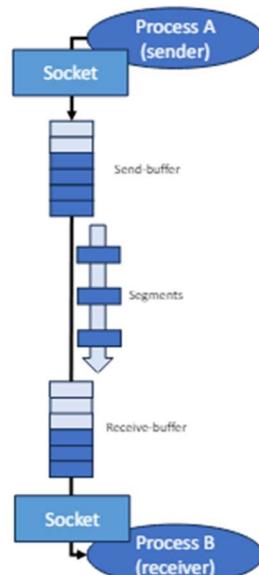
5.7.1. TCP: BUFFERS

Nella connessione TCP le operazioni di inviare e ricevere fanno molto affidamento sui buffers per essere eseguite.

I buffers permettono di disaccoppiare parzialmente i tempi di trasmissione da:

- Ritardi dell'applicazione
- Ritardi del sistema operativo nelle operazioni di multiplexing e demultiplexing dei pacchetti
- Ritardi della network (oscillazioni della prestazione della network)

Vi è anche il limite alla velocità di trasmissione, così che messaggi lunghi potrebbero essere frammentati in messaggi più piccoli che potrebbero essere collezionati in buffers (dal lato del mittente).



5.7.2 DIMENSIONE MASSIMA DEL SEGMENTO

La quantità di dati all'interno di un segmento è limitata dalla Maximum Segment Size (MSS):

$$MSS = MTU - \text{header_size}$$

dove:

- La Maximum Transmission Unit (MTU) è la massima lunghezza di un frame accettabile dal livello link
- Header_size è la header size combinata delle headers di TCP e IP.

All'invio: i segmenti sono creati dai dati dell'applicazione e passati giù al livello network, dove essi vengono separatamente incapsulati all'interno dei datagrammi IP del livello network per essere poi inviati.

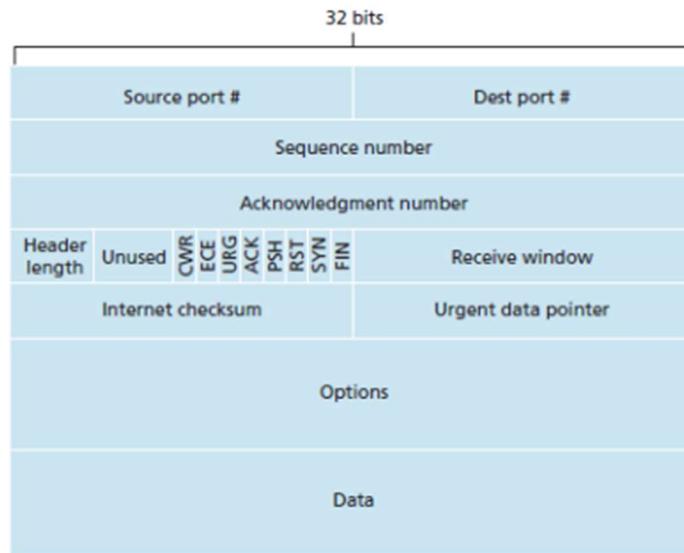
Alla ricezione: i dati sono posizionati nel buffer di ricezione, l'applicazione recupera dati dal buffer quando è pronta.

5.7.3 SEGMENTO DI TCP

Il segmento TCP consiste di campi header e di un campo data, l'ultimo contiene alcuni dati di applicazione al massimo di dimensione MSS.

L'header contiene diversi campi:

- Il numero di sequenza (32-bit) per il trasferimento dati affidabile
- Il numero di acknowledgment (32-bit) per il trasferimento dati affidabile
- Una finestra di ricezione (16-bit) usata per indicare il numero di bytes che un destinatario è in grado di accettare (per il controllo del flusso)
- La lunghezza dell'header (4-bit) che specifica il numero delle parole a 32-bit contenute nell'header. Nota che l'header TCP è variabile a causa del campo opzioni.
- Il campo opzioni (K-bit) usato per i processi opzionali come negoziare l'MSS, il time-stamping ecc
- Le flags (6-bit) includono:
 - o ACK bit indica che questo segmento è un pacchetto ACK (quindi il campo del numero di acknowledgment è in uso).
 - o I bits di RST, SYN, FIN usati per il setup della connessione e il teardown
 - o I bits di CWR e ECE usati in esplicite notifiche di congestione
 - o Il bit di PSH dice al destinatario di passare i dati all'applicazione immediatamente
 - o Il bit di URG indica che il segmento contiene dati urgenti
- Checksum (16-bit): per il controllo di integrità
- Il campo del puntatore dei dati urgenti (16-bit): indica il luogo dell'ultimo byte della parte urgente dei dati.



5.7.3.1 TCP: NUMERI DI SEQUENZA E DI ACKNOWLEDGMENT

I numeri di sequenza e i numeri di acknowledgment sono strettamente correlati.

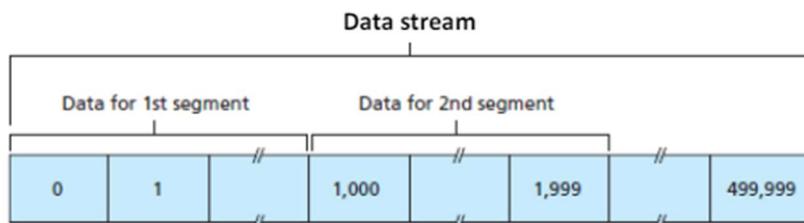
I numeri di sequenza sono usati non solo per il trasferimento dei dati affidabile ma anche per gestire la segmentazione.

- Se un grande flusso di dati è trasmesso dall'host A all'host B, vi è bisogno di suddividerlo in pezzi più piccoli a seconda dell'MSS.

Il numero di acknowledgment è calcolato a partire dai numeri di sequenza. È convenzionalmente il prossimo pezzo del flusso di dati che ci si aspetterebbe.

Si può notare che TCP è un protocollo di trasporto general-purpose, non ha informazioni circa i dati da trasmettere. Dal punto di vista del TCP i dati sono soltanto un flusso ordinato di bytes.

Il numero di sequenza per un segmento risulta poi essere il numero del flusso di byte del primo byte di dati presenti nel segmento.



Nell'esempio, si assume un flusso di dati di 500000 bytes e un MSS di 1000 bytes.

Il TCP costruisce 500 segmenti dove il primo segmento ha il numero di sequenza 0, il secondo segmento ha il numero di sequenza 1000, il terzo segmento ha il numero di sequenza 2000, e così via.

Si assuma che questo flusso di dati venga passato da un host A ad un host B.

Per semplicità, si trascura l'interazione precedente (i dati scambiati dallo stabilimento della connessione), così che si inizia dal numero di sequenza 0:

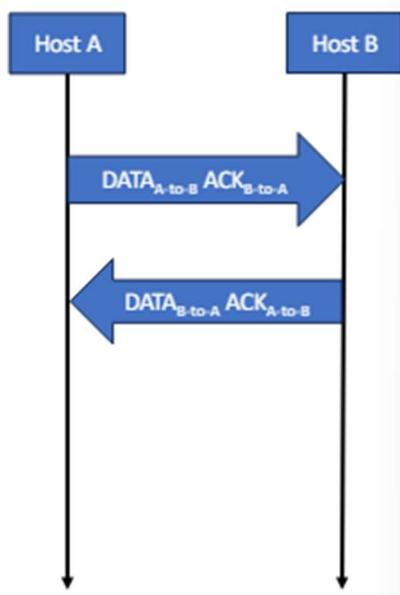
- Il destinatario otterrà un primo segmento di 1000 bytes, ad esempio dal numero di sequenza 0 al numero di sequenza 999
- Il destinatario farà l'acknowledgment della trasmissione settando un numero di acknowledgment di 1000 (il prossimo byte previsto nel flusso)
- Il destinatario otterrà il prossimo segmento di 1000 bytes, ad esempio dal numero di sequenza 1000 al numero di sequenza 1999 e così via...

Cosa accade se il destinatario sta inviando dati in ritorno?

- Ricorda: la comunicazione TCP è full-duplex, se due hosts A e B comunicano ci sono anche due flussi di dati da considerare per un'affidabile trasferimento dei dati: da A a B e da B ad A.

In questo caso si possono usare i numeri di sequenza e i numeri di acknowledgment allo stesso tempo:

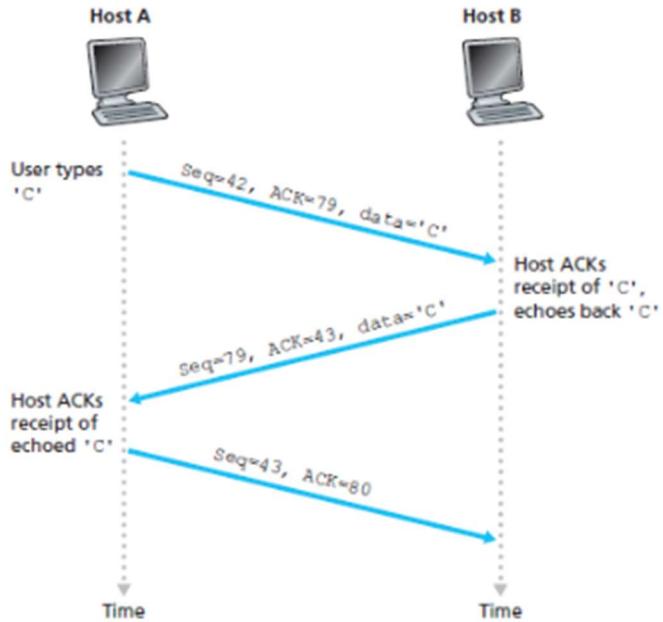
- I segmenti da A hanno i numeri di sequenza correlati al flusso da A a B e i numeri di acknowledgment correlati al flusso da B a A
- I segmenti da B hanno i numeri di sequenza correlati al flusso da B a A e i numeri di acknowledgment correlati al flusso da A a B.



Questo è un esempio semplificato di due host che si inviano dati l'un l'altro.

- Si considerano due messaggi di solo 1 byte (char) da A a B e viceversa
- Nello specifico, l'host A trasmette una "c" che viene inviata indietro da B

Qui questi segmenti forniscono ACK e DATA allo stesso tempo (il flag ACK è settato ad 1).



1. Da A a B:

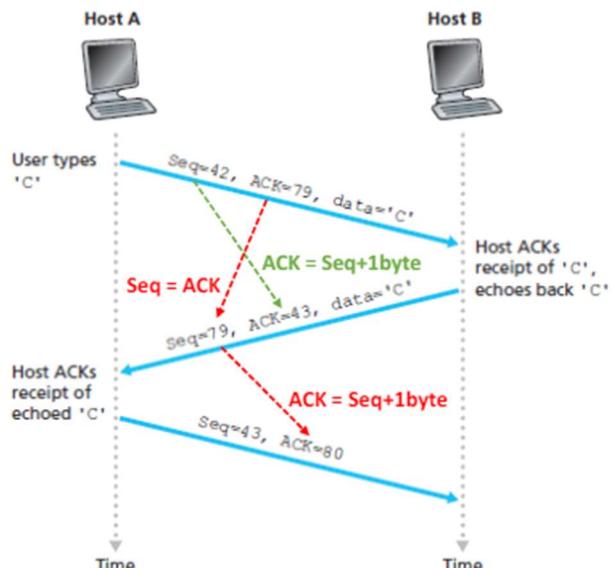
- Il numero ACK come numero di sequenza del prossimo pacchetto previsto nel flusso da B ad A (byte 79)
- Il numero di sequenza come la posizione del singolo byte che viene trasmesso (byte 42)

2. Da B ad A:

- Il numero ACK come il numero di sequenza del prossimo pacchetto previsto nel flusso da A a B (il numero di sequenza + 1 byte del char "c")
- Il numero di sequenza come la posizione del byte che si trasmette (byte 79)

3. Da A a B:

- Il segmento puramente ACK è inviato, avente il numero di sequenza 43 (come previsto da B) e il numero di ack 80 (prossimo previsto).



A-to-B info are in **green**.
B-to-A info are in **red**.

5.7.4. TIMEOUT DI RITRASMISSIONE

Precedentemente abbiamo enfatizzato che il meccanismo affidabile di trasferimento di dati TCP fa un uso estensivo dei timeouts.

I timeouts possono essere usati in combinazione con ACKs per capire (o almeno provare) se i segmenti erano stati persi o potrebbero essere ritrasmessi.

- Si nota che l'ACK sbagliato può essere ricevuto per differenti ragioni (non solo quando è perso)
- Per esempio, se i segmenti arrivano nell'ordine sbagliato alla destinazione viene inviato un ACK differente.

La gestione dell'ACK può essere complicata. L'approccio di base seguito dai metodi pipelining (dal lato del mittente) è di ignorare gli ACKs ricevuti in modo errato:

- La ritrasmissione è eseguita solo quando il timeout è trascorso
- La stima del timeout è fondamentale.

5.7.4.1 STIMA DEL RTT

Per settare i timeouts in modo adatto è necessaria la stima di un round-trip time (RTT)

Il timeout, per esempio il tempo di attesa di un ACK di segmento, dovrebbe essere più grande del RTT altrimenti dovrebbe essere inviata una ritrasmissione non necessaria.

Il TCP stima questo valore campionando l'RTT di segmenti acknowledged con successo che non sono stati ritrasmessi.

Siccome il tempo di un RTT campionario (*samRTT*) può fluttuare (dovuto al traffico della congestione, al carico di lavoro del destinatario) la stima del RTT (*estRTT*) è data dalla exponential weighted moving average (EWMA):

$$estRTT_t = (1 - \alpha)estRTT_{t-1} + \alpha samRTT_t$$

dove α è un parametro che in genere ha valore 0.125.

Questo è un esempio di come RTT si comporta in uno scenario reale (tra Francia e USA)

È anche utile per misurare la variabilità di RTT (*devRTT*) a partire dalla differenza tra il campione (*samRTT*) e la stima (*estRTT*):

$$devRTT_t = (1 - \beta)devRTT_{t-1} + \beta |samRTT_t - estRTT_t|$$

Dove β è un parametro che in genere ha valore 0.25.

Avendo questa stima, è possibile definire un timeout di ritrasmissione per il TCP che non è né più basso né più alto dell'RTT stimato.

$$\text{timeout}_t = \text{estRTT}_t + 4\text{devRTT}_t$$

Qui la deviazione dell'RTT è usata per settare un margine responsabile ed adattabile all'RTT stimato.

- Esso cresce quando RTT oscilla, così che la finestra di timeout è più grande quando si è incerti circa il corrente RTT.

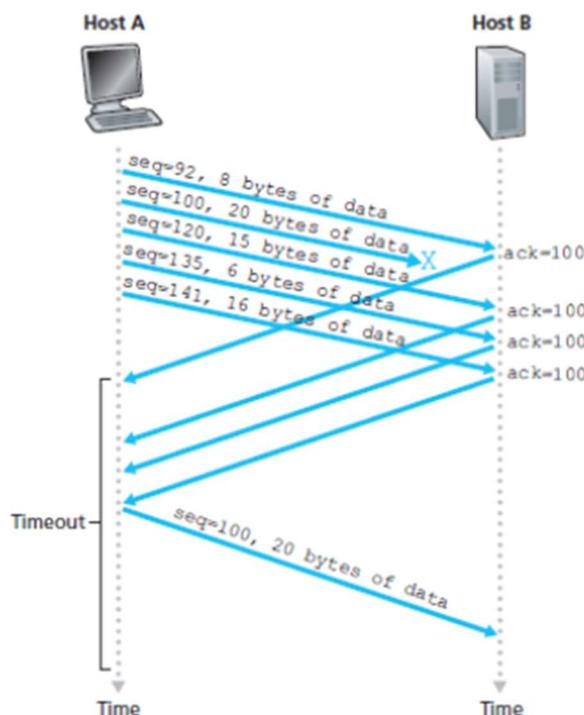
Di default, il valore iniziale dell'RTT è di 1 secondo (a $t=0$).

5.7.5 FAST RETRANSMIT

Uno dei problemi che accadono con le ritrasmissioni attivate dal timeout è che il periodo di timeout può essere relativamente lungo.

Per limitare questo problema TCP applica un meccanismo chiamato ritrasmissione veloce che utilizza un ACK sbagliato:

- Quando un destinatario TCP riceve un segmento con un numero di sequenza che è più grande di quello previsto significa che un segmento è stato mancato in modo ragionevole.
- Il destinatario riceve il vecchio ACK (ACK duplicato) avente un numero di ACK del segmento previsto.
- Se il mittente riceve N duplicati (tipicamente 3 duplicati) si assume che il segmento precedente è stato perso così che viene velocemente ritrasmesso prima della deadline.



5.7.6. TCP: PROBLEMI NELLA GESTIONE DELLA CONNESSIONE

Siccome TCP è connection-oriented i due host devono essere d'accordo sia durante le procedure di apertura che di chiusura.

Sapendo che questi messaggi potrebbero essere persi o danneggiati nella network, è abbastanza difficile per i due host trovare un accordo del genere.

Se i messaggi vengono persi durante la trasmissione, un'estremità della comunicazione può essere aperta o chiusa mentre l'altra no.

Per evitare (o meglio mitigare) questo problema TCP implementa una procedura chiamata handshake a tre vie in cui le richieste di apertura o chiusura della connessione devono essere riconosciute dagli host prima che una connessione possa essere stabilita o rilasciata.

Il problema dei due eserciti:

- Si immagini un esercito bianco accampato in una valle e su entrambe le colline vi sono due eserciti nemici blu.
- L'esercito bianco è più grande dei singoli eserciti blu, ma assieme i due eserciti blu sono più grandi e vinceranno soltanto se l'attacco è simultaneo.
- Per sincronizzare i loro attacchi, gli eserciti blu devono inviare messaggeri attraverso la valle i quali però potrebbero essere catturati (comunicazione non affidabile)
- Esiste un protocollo che permette agli eserciti blu di vincere?

Si suppone che il comandante dell'esercito blu 1 invii un messaggio con scritto. "Propongo di attaccare oggi, va bene?"

Adesso si supponga che il messaggio arrivi, che il comandante dell'esercito blu 2 accetti e che la sua risposta arrivi con successo all'esercito blu 1.

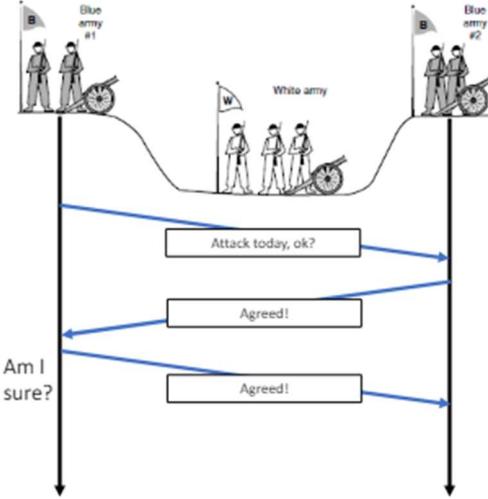
Si verificherà l'attacco? probabilmente no, perché il comandante 2 non sa che la sua risposta è arrivata, perché se non è arrivata l'esercito 1 non attaccherà.

Facciamo sì che questa sia una handshake a tre vie: il primo comandante (della proposta originale) deve rendere noto che è a conoscenza della risposta.

Assumendo che nessun messaggio sia andato perso, l'esercito blu 2 otterrà l'acknowledgment, ma il comandante dell'esercito blu 1 ora esiterà (se lui non sa che il suo acknowledgment è andato in porto).

Facciamo sì che questo diventi un handshake a quattro vie, ma neanche questo aiuta. Infatti è possibile provare che nessun protocollo esistente funzioni.

- L'handshake a tre vie non è perfetto ma in generale è sufficiente!



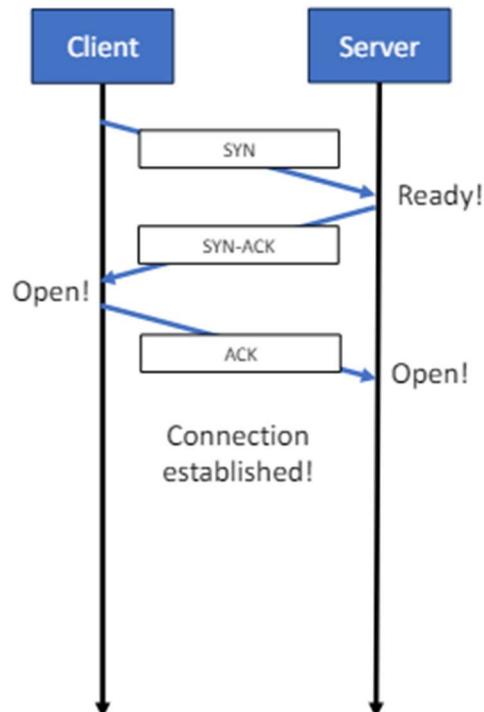
5.7.7 TCP: STABILIMENTO DELLA CONNESSIONE

In TCP un handshake a tre vie viene eseguito per stabilire la connessione:

- Il client invia una richiesta di connessione (segmento SYN) al server
- Il server risponde con uno speciale acknowledgment (segmento SYN-ACK)
- Il client invia indietro un acknowledgment finale (segmento ACK)

Lo stabilimento della connessione TCP è una procedura delicata che può anche aggiungere ritardi significativi:

- C'è in genere un timeout (di 30 – 60 secondi) per completare l'handshake, dopo che la procedura è abortita.
- Alcuni attacchi (ad esempio inondazione SYN) accadono qui.



Dettagli della procedura handshake a tre vie:

- 1. Il client invia un segmento SYN aventure:
 - Nessun dato del livello applicativo (nessun payload)
 - Il bit SYN è settato ad 1
 - Un numero di sequenza iniziale casuale (client_isn) come numero di sequenza
- 2. Quando il suddetto segmento SYN (si spera) arriva, il server alloca variabili e buffers TCP e manda indietro un segmento SYNACK aventure:
 - Nessun dato del livello applicativo (nessun payload)
 - Il bit SYN è settato ad 1
 - Il numero di acknowledgment è settato a client_isn+1
 - Un numero di sequenza iniziale casuale (server_isn) come numero di sequenza
- 3. Quando il segmento SYNACK (si spera) arriva, il client alloca anch'esso variabili e buffers per la connessione ed invia al server il segmento finale ACK aventure:
 - Possibilmente dati del livello applicativo
 - Il bit SYN settato a 0 (la connessione è stabilita)
 - Il numero di acknowledgment settato a server_isn+1

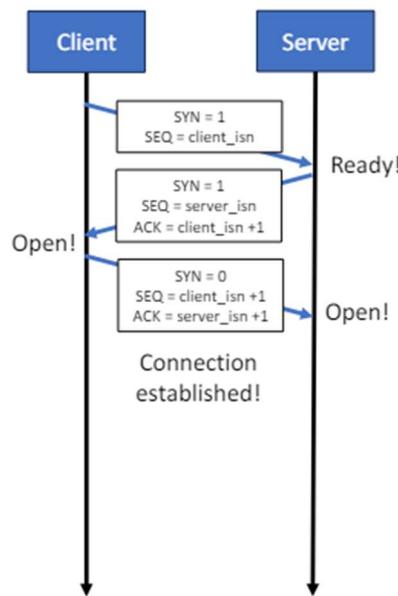
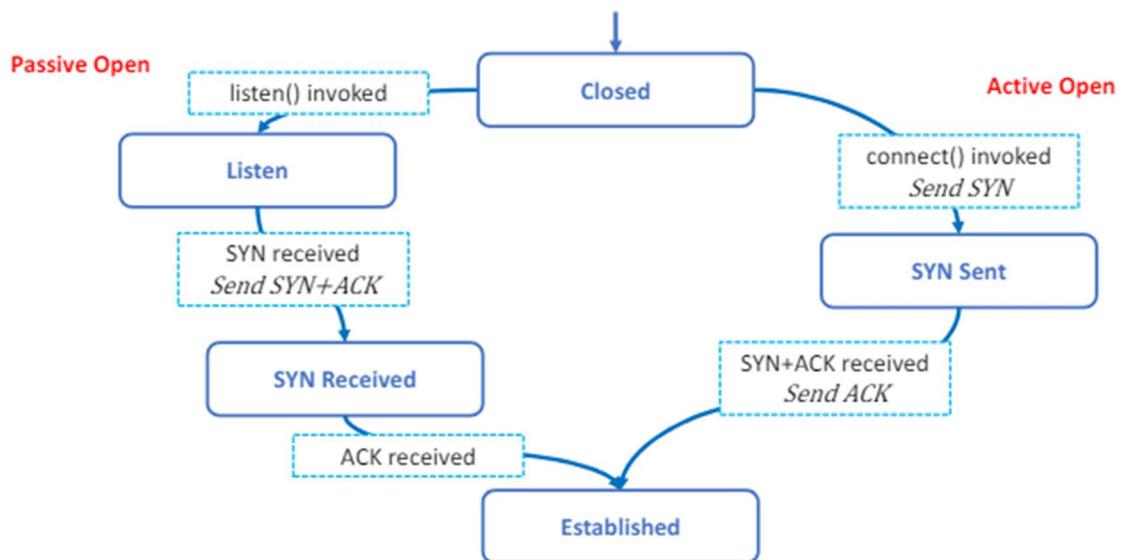


Diagramma di stato semplificato per lo stabilimento della connessione TCP.



5.7.8 TCP: IL RILASCIO DELLA CONNESSIONE

Il rilascio della connessione TCP (teardown) può essere eseguita da entrambi gli hosts.

Si assume che il client stia chiudendo la connessione, l'handshake a tre vie viene eseguito come segue:

- 1. Il client invia uno speciale segmento shutdown (segmento FIN) al server avente il bit FIN settato a 1.
- 2. Quando il server riceve il segmento, esso manda indietro un segmento di acknowledgment/shutdown avente ACK a 1 e bit FIN settato a 1.
- Da notare che ACK e FIN possono essere inviati nel medesimo segmento o in due separati.
- 3. Alla fine, il client rende noto al server di aver ricevuto il suo segmento di shutdown.

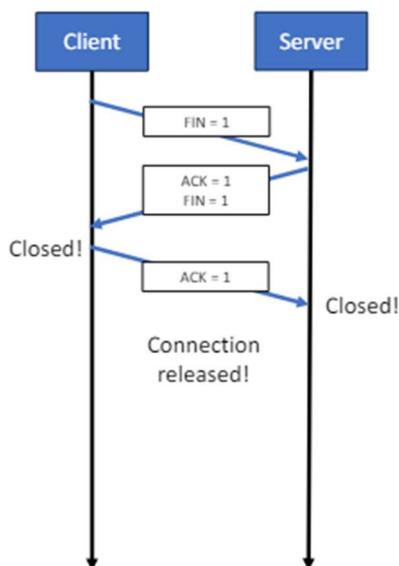
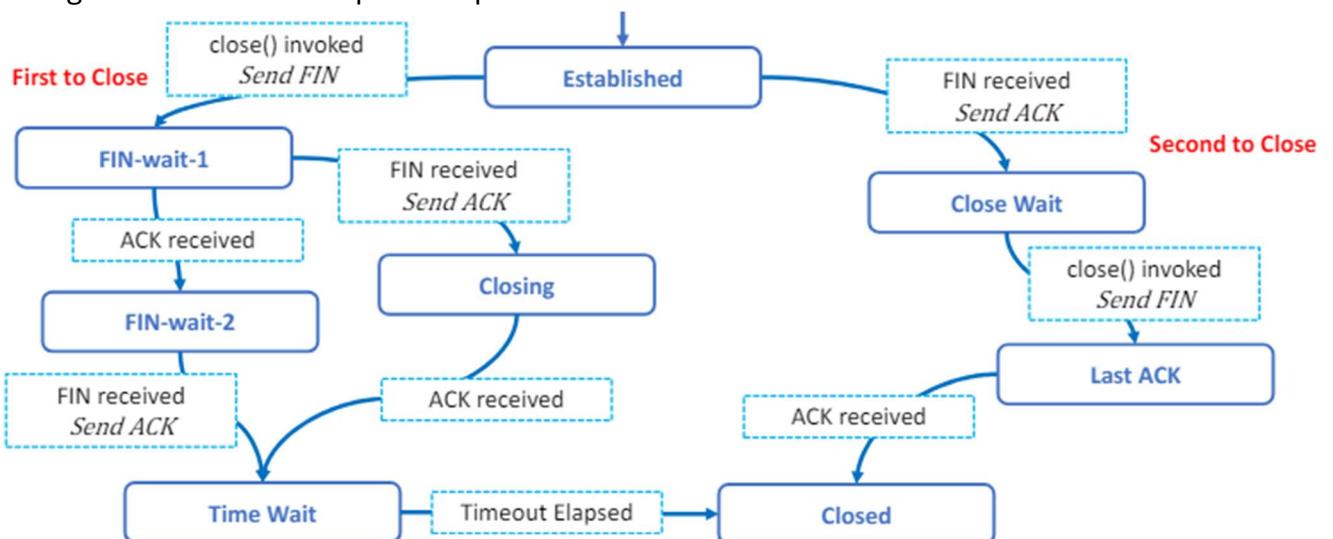


Diagramma di stato semplificato per il rilascio della connessione TCP.



Dato che l'handshake a tre vie non è perfetto, TCP fa affidamento su timers per eventualmente chiudere le connessioni o inviare nuovamente richieste.

5.7.9. IL PROBLEMA DELLA CONGESTIONE E DEL CONTROLLO DEL FLUSSO

Caratteristiche aggiuntive del TCP rispetto a UDP sono controllo della congestione e del flusso.

Si è menzionato che la perdita dei pacchetti è spesso risultato di un sovraccarico dei buffers:

- Dal buffer del destinatario, se l'applicazione ricevente non è abbastanza veloce nel leggere i dati
- Dai dispositivi network (ad esempio i routers) se i nodi sono congestionati.

Se i pacchetti vengono persi, gli hosts sono costretti a far affidamento sulla ritrasmissione e sui timeouts che impattano drasticamente sulle prestazioni della network.

5.7.9.1 CONTROLLO DEL FLUSSO

Quando la connessione TCP riceve bytes che sono corretti e in sequenza, esso piazza i dati nel buffer di ricezione. L'applicazione ricevente leggerà i dati dal buffer, ma non necessariamente nell'istante in cui i dati arrivano.

Se l'applicazione è relativamente lenta nel leggere i dati, il mittente potrebbe facilmente mandare in sovraccarico il buffer del destinatario inviando troppi dati troppo velocemente.

Il controllo del flusso di TCP è un servizio di speed-matching: fa in modo di associare (ridurre) la velocità a cui il mittente invia alla velocità con cui l'applicazione ricevente legge.

- Nota: il segmento TCP permette al ricevente di comunicare al mittente quanto spazio è rimasto disponibile nel buffer mediante il campo finestra di ricezione (receive window field).

Si assume per semplicità che il ricevente TCP scarti i segmenti non in ordine così che tutti i segmenti nel buffer sono ordinati.

Dal lato del ricevente (host B) abbiamo che:

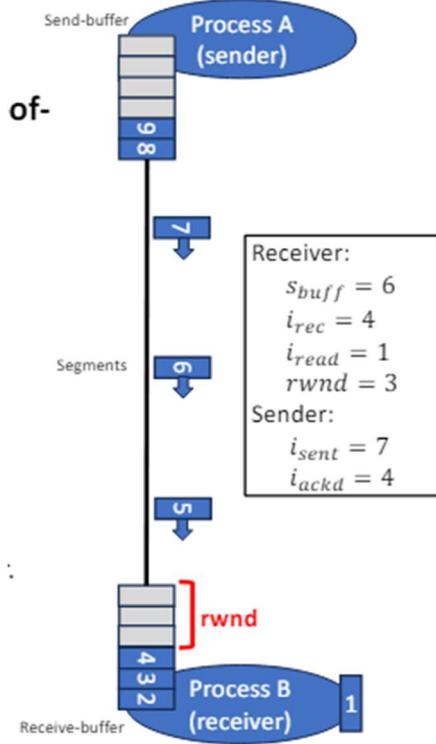
- s_{buff} è la dimensione del buffer ricevente (espresso in bytes)
- i_{read} è l'ultimo byte del flusso recuperato dall'applicazione
- i_{rec} è l'ultimo byte del flusso ad essere ricevuto
- possiamo definire la dimensione della finestra di ricezione ($rwnd$) come:

$$rwnd = s_{buff} - (i_{rec} - i_{read})$$

Dal lato del mittente (host A) abbiamo:

- i_{sent} è l'ultimo byte del flusso ad essere inviato
- i_{ackd} è l'ultimo byte del flusso che è stato acknowledged dal ricevente
- conoscendo la finestra di ricezione, il mittente può garantire in qualsiasi momento che:

$$i_{sent} - i_{ackd} \leq rwnd$$



5.7.9.2 PROBLEMA DELLA CONGESTIONE

Il problema del controllo della congestione è simile al controllo del flusso, ma riguarda l’infrastruttura della network.

Esempio: i due hosts (A e B) hanno una connessione che condivide un singolo router e un singolo collegamento in uscita con capacità R tra la sorgente e la destinazione.

Il router ha buffers che gli permettono di immagazzinare pacchetti quando la velocità di arrivo del pacchetto eccede la capacità di uscita del collegamento fisico.

Si assume per comodità che entrambe le applicazioni (in A e in B) stiano inviando dati alla connessione alla stessa velocità media di λ_{in} misurata in bytes/secondo.

Se $\lambda_{in} \leq R/2$, tutto ciò che viene inviato dal mittente viene ricevuto dal destinatario con un ritardo finito.

Se $\lambda_{in} > R/2$, il link raggiunge la sua piena capacità (R) e i pacchetti in eccesso sono immagazzinati nel buffer.

Man mano che si eccede la massima capacità i pacchetti saranno accumulati nel buffer del router aspettando il loro turno di essere inviati nel link fisico.

Siccome il buffer è finito, i pacchetti accumulati saranno eventualmente scartati e gli host forzati a ritrasmetterli.

Neppure se il buffer fosse infinito funzionerebbe, i pacchetti non verrebbero persi ma il ritardo aumenterebbe costantemente: se si eccede la capacità all’infinito, il ritardo raggiungerà l’infinito (il che vuol dire che i pacchetti è come se fossero persi).

Ci sono principalmente due approcci per controllare la congestione:

- 1. Controllo di congestione end-to-end: questo è l'approccio standard del TCP in cui la congestione è dedotta dagli end systems basandosi soltanto sull'osservazione del comportamento della network. La perdita di un segmento TCP (dovuto ad un timeout o alla ricezione di 3 ACK duplicati) è considerata come un indice di congestione della network e TCP diminuisce la velocità di invio di conseguenza.
- 2. Controllo della congestione network-assisted: questo è un approccio recente e opzionale entro cui il livello di trasporto lavora in sinergia con il livello network. I routers forniscono feedback esplicito al mittente e/o ricevente riguardo lo stato di congestione della network. Il feedback può anche essere un semplice bit che indica la congestione ad un collegamento, tuttavia è possibile sviluppare anche feedback più sofisticati.

TCP per lo più si basa sul controllo della congestione end-to-end.

L'approccio è quello di avere ciascun mittente che adatta la propria velocità di invio a seconda della congestione della network percepita.

Ecco tre principali problemi che vanno presi in considerazione:

- 1. Rate regulation: come fa un mittente TCP a regolare la velocità a cui invia traffico all'interno della connessione?
- 2. Congestion detection: come fa un mittente TCP a percepire la congestione sul percorso tra sé stesso e la destinazione.
- 3. Rate adjustment: quale algoritmo il mittente dovrebbe usare per cambiare la sua velocità di invio in funzione della congestione end-to-end percepita?

TCP CONTROLLO DELLA CONGESTIONE: RATE REGULATION

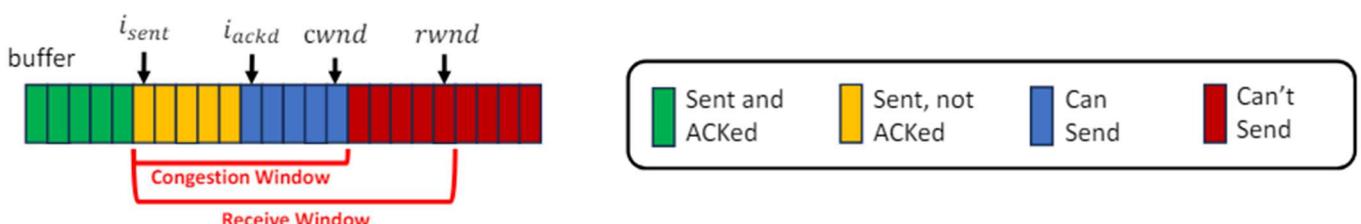
È importante ricordare che nel controllo del flusso di TCP la velocità di invio è regolata considerando la dimensione del buffer sul lato del ricevente (receiver window):

$$i_{sent} - i_{ackd} \leq rwnd$$

Nel controllo della congestione TCP il mittente tiene anche traccia della finestra di congestione (*cwnd*):

$$i_{sent} - i_{ackd} \leq \min(rwnd, cwnd)$$

Pertanto la velocità è regolata incrementando o decrementando la finestra di congestione *cwnd*.



TCP CONTROLLO DELLA CONGESTIONE: CONGESTION DETECTION

Un mittente TCP percepisce che vi è una congestione sul percorso tra sé stesso e la destinazione in due modi verificando gli eventi di perdita.

Un evento di perdita accade quando o si verifica un timeout o vengono ricevuti 3 ACKs duplicati.

- Entrambi gli eventi significano che un pacchetto precedente non è arrivato a destinazione probabilmente a causa di un sovraccarico dei dispositivi della rete.

Se gli eventi di perdita non si verificano, TCP assumerà che la network non è congestionata pertanto la congestion window può essere incrementata.

D'altro canto, se gli eventi di perdita si verificano, la finestra di congestione deve essere decrementata.

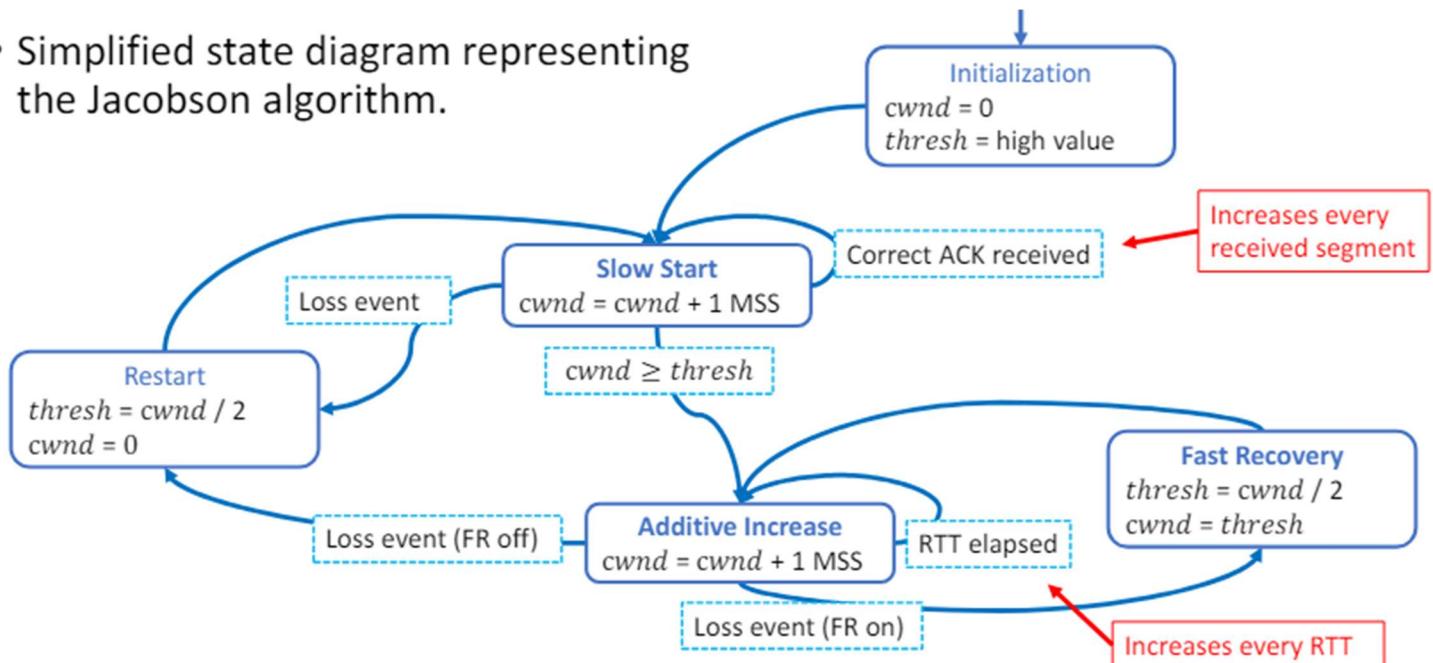
TCP CONTROLLO DELLA CONGESTIONE: RATE ADJUSTMENT

Il rate adjustment della TCP è poi eseguito attraverso un sondaggio della bandwidth: la velocità viene incrementata fino a quando gli ACKs arrivano correttamente (sondaggio della network), quando la congestione viene individuata (eventi di perdita) la velocità viene decrementata. Tale processo è continuamente ripetuto.

Il TCP usa l'algoritmo Jacobson congestion-control per regolare la velocità. Esso include 3 fasi:

- 1. Slow start: inizia con un incremento della velocità di 1 MSS/RTT esponenzialmente.
- 2. Additive increase: incrementa la velocità linearmente
- 3. Fast recovery (opzionale): dimezza la velocità invece di effettuare lo slow-start nuovamente e procede direttamente con l'additive increase.

- Simplified state diagram representing the Jacobson algorithm.



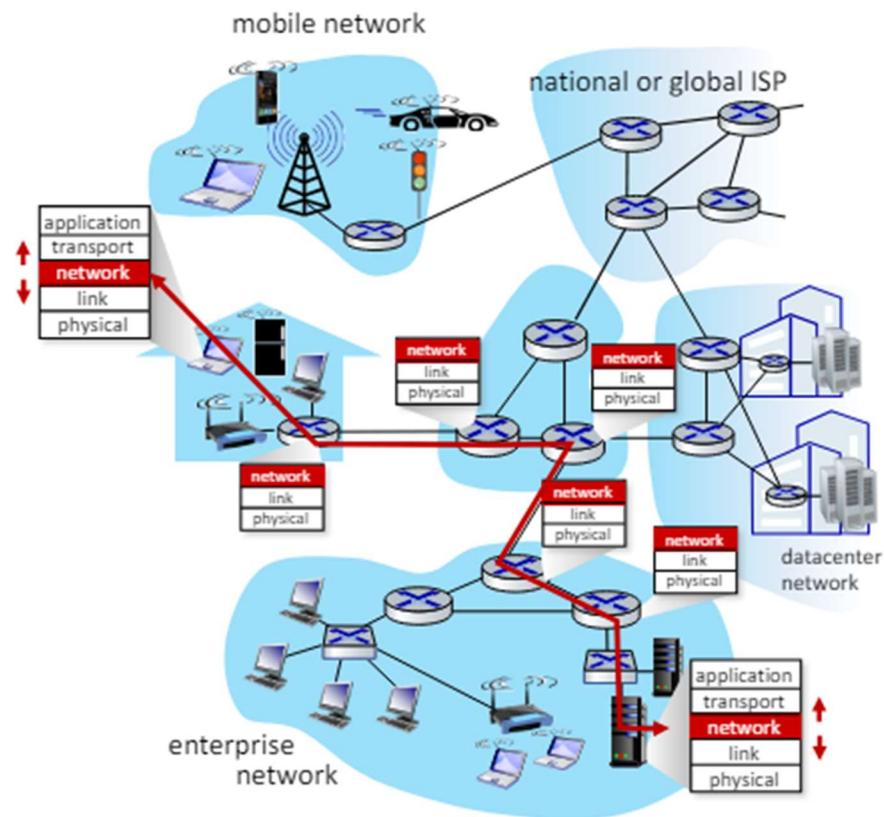
6 LIVELLO NETWORK

Il lavoro del livello network:

- Nell'host mittente: prendere segmenti dal livello di trasporto, incapsulare ciascun segmento in un datagramma ed inviare i datagrammi nella network.
- Nell'host destinatario: ricevere i datagrammi dalla network, estrarre i segmenti del livello trasporto dai datagrammi, e consegnare i segmenti su al livello di trasporto.

All'interno della network vi sono nodi che inoltrano i datagrammi ai nodi adiacenti (routers) fino al raggiungimento dell'host di destinazione.

- I routers sono mostrati con uno stack protocollo troncato. Potenzialmente i routers non eseguono protocoli a livello applicativo e a livello di trasporto.



6.1 IL LIVELLO NETWORK IN INTERNET

Il livello network è principalmente responsabile della delivery host-to-host.

Quali servizi potrebbero essere offerti con la delivery?

- Guaranteed delivery: un pacchetto inviato ad un host sorgente arriverà eventualmente all'host destinatario.
- Guaranteed delivery con ritardo limitato: i pacchetti saranno consegnati e all'interno di uno specifico limite di ritardo host-to-host.
- In-order packet delivery: I pacchetti arriveranno a destinazione nell'ordine in cui sono stati inviati.

- Guaranteed minimal bandwidth: la possibilità di specificare una minima velocità di bit tale da, se la velocità di invio dell'host è compresa in essa, allora tutti i pacchetti verranno eventualmente consegnati all'host di destinazione.
- Security: crittografia/decriptazione di tutti i datagrammi alla sorgente o destinazione.

In realtà, nessuno di questi servizi viene offerto dalle networks.

Al contrario, il livello network di internet fornisce un unico servizio, il cosiddetto servizio best-effort.

Il servizio best-effort: la rete fa del suo meglio per consegnare un pacchetto dalla sua sorgente alla sua destinazione.

Nonostante la sua semplicità, il modello best-effort service, in combinazione con una buona bandwidth ha dato prova di essere adeguato.

6.2 ROUTERS

Il ruolo principale del livello network è di eseguire l'host-to-host delivery.

Il processo è eseguito dai routers (i nodi della network) che sono speciali nodi aventi molteplici link di ingresso e di uscita. I routers forniscono due funzioni al livello di network:

- 1. Forwarding: quando un pacchetto arriva al link di input di un router, il router deve muovere il pacchetto verso un appropriato link di output. È inoltre possibile:
 - Impedire ad un pacchetto di uscire da un router (ad esempio se il pacchetto viene originato da un mittente dannoso, oppure se il pacchetto è destinato ad un host destinatario proibito.)
 - Duplicare un pacchetto e inviarlo a molteplici link di uscita
- 2. Routing: decidere il percorso da far prendere ai pacchetti mentre viaggiano dalla sorgente al destinatario. Gli algoritmi che calcolano questi percorsi sono conosciuti con il nome di algoritmi di routing.

I routers possono essere molto differenti a seconda delle loro funzioni:

- Utilizzo domestico o business
- Connessioni cablate o wireless
- Edge routers: un router che distribuisce pacchetti di dati tra uno o più networks (ad esempio connettere una network con un ISP)
- Core routers: usati per distribuire i pacchetti all'interno della medesima network piuttosto che attraverso multiple network.
 - Sono spesso utilizzati come backbone di internet e il loro lavoro è quello di trasferire grosse quantità di dati.

Forwarding (piano dei dati) si riferisce all'azione di un router locale di trasferire un pacchetto da un'interfaccia link input ad un'appropriata interfaccia link output.

- Questa è un'operazione veloce e pertanto tipicamente implementata nell'hardware.

Routing (piano di controllo) si riferisce ad un processo network-wide che determina i percorsi end-to-end che i pacchetti percorrono da sorgente a destinazione.

- Questo è un processo più lento, ed è spesso implementato mediante software.

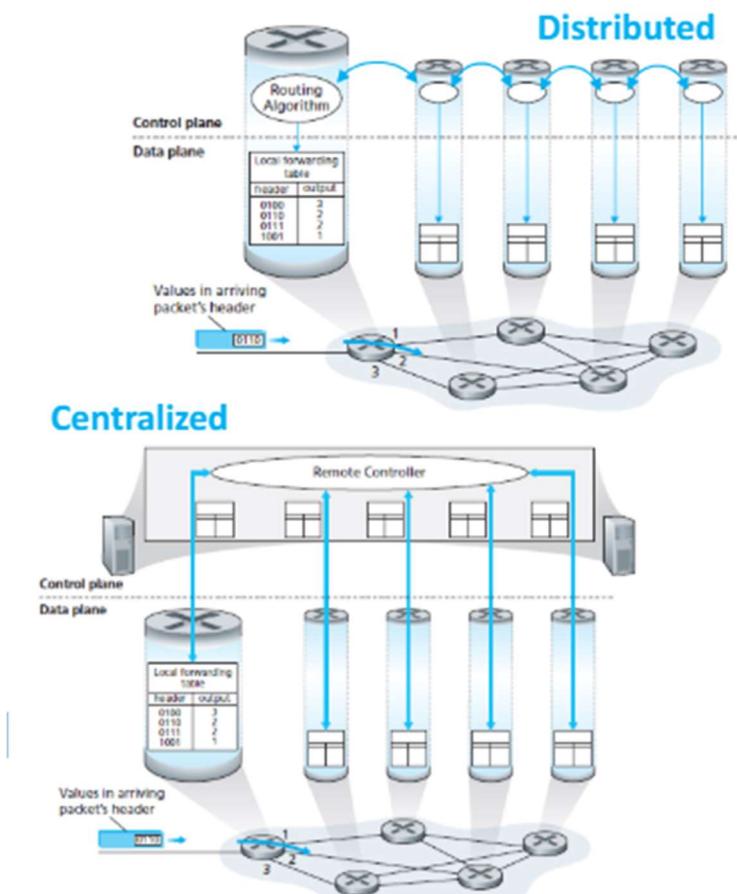
Dal piano dei dati abbiamo la forwarding table: una tabella che specifica a quali link output un pacchetto dovrebbe essere inoltrato per far sì che raggiunga la destinazione.

- Un router inoltra un pacchetto esaminando il valore di uno o più campi nell'header dei pacchetti in arrivo.
- Il valore immagazzinato nella sezione della forwarding table indica l'interfaccia link di uscita alla quale il pacchetto deve essere inoltrato.

Siccome molteplici routers possono essere incontrati prima di raggiungere la destinazione, il contenuto della forwarding table deve essere determinato collezionando informazioni dai diversi router.

Questa funzionalità può essere eseguita in due modi:

- Decentralizzato (o distribuito): possiamo avere ogni router dotato con un componente di routing che comunica con il componente di routing degli altri routers (questo è l'approccio principale).
- Centralizzato: possiamo avere un controller remoto fisicamente separato che calcola e distribuisce le forwarding tables che verranno poi usate dai routers.



Nel secondo caso, il controller remoto potrebbe essere implementato in un data center remoto e potrebbe essere gestito da un ISP o da terze parti.

Questo approccio è alla base del software-defined networking (SDN) dove la network è “software-defined” perché il controller che calcola le forwarding tables e interagisce con i routers è implementato mediante un software.

6.2.1 ROUTERS: LE PRINCIPALI COMPONENTI

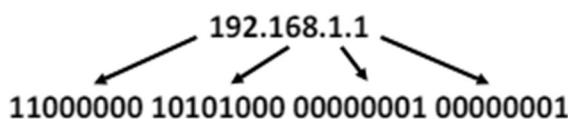
Le principali componenti di un router sono:

- Le porte di input (differenti dalle porti a livello di trasporto): sono delle interfacce fisiche di input che operano con il livello link inferiore del link connesso, il loro lavoro è:
 - o Consultare la forwarding table (lookup) e preparare la switching fabric per la porta di output da scegliere.
 - o Inoltrare pacchetti di controllo al processore di routing
 - o Il numero delle porte input può variare da dozzine a centinaia.
- Switching fabric: connette le porte input alle porte output.
- Le porte output: immagazzinano i pacchetti ricevuti dallo switching fabric e trasmettono questi pacchetti ai link in uscita.
 - o Le porte sono spesso bidirezionali, ad esempio la porta output è accoppiata con una porta input.
- Routing processor: che esegue i protocolli di routing, mantiene le informazioni di stato circa i link e computa la forwarding table.

6.2.2. ROUTERS: FORWARDING

Il ruolo della forwarding table è quello di associare indirizzi IP alle porte di output, così che i pacchetti possano essere inoltrati al giusto link di output per essere trasmessi al prossimo nodo (e possibilmente essere inoltrati nuovamente)

Un indirizzo IP è un numero a 4-bytes (32-bits) che si vede spesso nella notazione decimale, ma che può essere anche visto nella sua forma binaria:



All'interno di una forwarding table, gli IPs sono associati ai numeri di porta di output per il reindirizzamento.

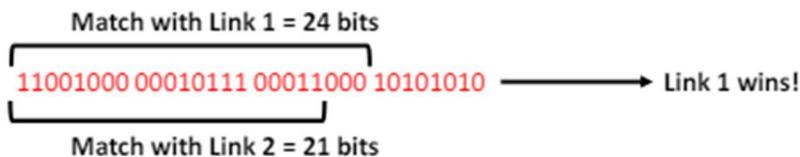
Quando un pacchetto è ricevuto dal link alla porta input una operazione lookup viene eseguita.

La porta ha una copia della forwarding table dal processore del routing (ricevuta da un bus dedicato), per evitare collo di bottiglia (bottleneck) dovuto alle continue invocazioni del processore di routing centralizzato su una base per pacchetto.

Example of a 4-ports router's table

Prefix of IP addresses	Link Interface
11001000 00010111 00010**** *****	0
11001000 00010111 000110000 **** ***	1
11001000 00010111 00011*** *** ***	2
otherwise	3

Da notare che le entries possono non essere mutualmente esclusive, in questo caso il router lo inoltra alla entry corrispondente più lunga (la regola del longest prefix matching).



Questa procedura lookup è tipicamente eseguita su hardware per essere eseguita il più in fretta possibile.

- Sono anche coinvolti embedded memories e avanzati algoritmi table-search.

Una volta che la porta di output di un pacchetto è stata determinata (in seguito ad un lookup), il pacchetto può essere inviato all'interno della switching fabric. In alcuni dispositivi, i pacchetti possono anche essere temporaneamente messi in buffer (buffered) se altre porte input stanno utilizzando la fabric.

Questa operazione a due passaggi di looking up l'indirizzo IP di destinazione (match) e forwarding (action) è chiamata match-plus-action ed è eseguita in molti dispositivi di network, quali:

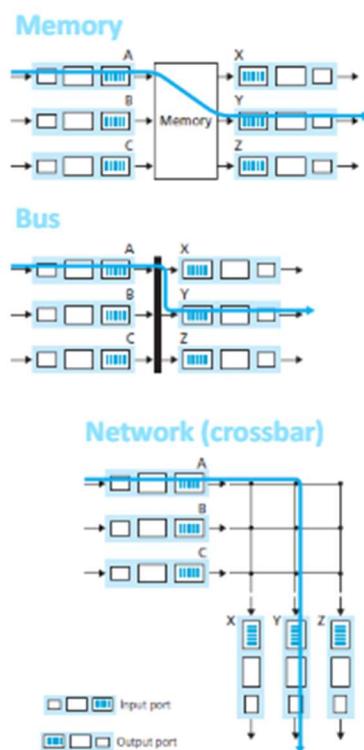
- Switches: azione simile a quella dei routers
- Firewalls: dove l'azione è di filtrare fuori specifici pacchetti in ingresso
- Network address translators (NATs) dove l'azione è quella di riscrivere il numero di porte di specifici pacchetti in ingresso prima di inoltrarli.

6.2.3. ROUTERS: SWITCHING FABRIC

In una switching fabric, lo switching può essere eseguito in 3 modi:

- 1. Switching via memory: le porte sono considerate al pari di dispositivi I/O che scrivono i pacchetti in celle di memoria e poi il processo di routing copia il messaggio sulla porta di output come specificato dalla forwarding table.
 - o Questo approccio è piuttosto lento ed era molto comune nei primi routers
- 2. Switching via bus: una porta di input trasferisce direttamente un pacchetto alla porta di output attraverso un bus condiviso, senza l'intervento di un processore di routing.
 - o Solo una porta per volta può essere servita, ma questo metodo è spesso sufficiente per i routers che operano in piccole aree locali.

- 3. Switching via interconnection network: le porte input/output sono connesse da una network aventi cross-points che possono essere aperti o chiusi e che poi rindirizzano i pacchetti.
 - o Qui molteplici pacchetti possono essere inoltrati in parallelo. Questo metodo è usato in numeri routers moderni.



6.2.4. ROUTERS: LE PORTE

Siccome lo switching richiede tempo, le porte di input e di output hanno code per immagazzinare temporaneamente i pacchetti.

L'estensione dell'accodamento non è fissa, potrebbe dipendere dal carico del traffico, dalla velocità dello switching fabric o dalla velocità della linea.

In generale, i pacchetti possono essere ricevuti/inviati più velocemente/più lentamente dello switching, pertanto essi possono accumularsi nei buffer di input/output (che potrebbero andare in sovraccarico).

6.2.5. ROUTERS: SCHEDULING DEI PACCHETTI

È ragionevole avere molteplici pacchetti che vengono inoltrati ad una singola porta output (potenzialmente provenienti da multiple porte input).

L'accesso ai pacchetti in coda dal buffer al link di output necessita di essere schedulato.

Ci sono principalmente 3 approcci:

- 1. First-come-first-served (FCFS, oppure first-in-first-out FIFO), che è approccio semplice time-based.

- 2. Priority queueing, che è basato sull'importanza dei pacchetti
- 3. Round-robin-queueing, dove sono divisi in classi (basate sulla priorità) e ogni classe viene servita a turno.

6.2.5.1. ROUTERS: PACKET SCHEDULING – FIFO

Se il link di output è impegnato (sta trasmettendo qualcosa) i pacchetti in arrivo sulla porta output devono essere posti in buffer.

Se non vi è sufficiente spazio di buffering per mantenere i pacchetti in arrivo, si dovrà far affidamento su una politica di packet-discardng.

- Una tipica politica consiste nel scartare i pacchetti recentemente arrivati (drop-tail) ma in approcci più sofisticati anche pacchetti già presenti all'interno del buffer possono essere rimossi per far spazio a quelli nuovi.

Un pacchetto è rimosso dalla coda solo se è stato completamente trasmesso attraverso il link di uscita (servito).

Nello scheduling FIFO i pacchetti sono selezionati per la trasmissione nello stesso ordine in cui sono arrivati alla porta di output.

6.2.5.2. ROUTERS: PACKET SCHEDULING – PRIORITY

Nel priority queueing, i pacchetti che arrivano al link di output sono classificati (ad esempio attraverso numeri di porte TCP/UDP) in classi di priorità al momento dell'arrivo nella coda.

Un operatore della network può configurare la coda in modo che specifici pacchetti possano ricevere priorità sul traffico utente o sui pacchetti non in tempo reale.

Ogni classe di priorità tipicamente ha una propria coda:

- I pacchetti appartenenti alle classi maggiormente prioritizzate non vuote sono trasmessi per primi.
- La scelta tra i pacchetti nella stessa classe di priorità è tipicamente realizzata mediante l'approccio FIFO.

6.2.5.3 ROUTERS: PACKET SCHEDULING – ROUND-ROBIN

Nel round robin queueing, i pacchetti sono ancora organizzati in classi, ma le classi sono alternate piuttosto che selezionate in base alla priorità.

Una comune implementazione è chiamata weighted fair queueing (WFQ) dove i pacchetti in arrivo sono classificati e accodati nelle aree di attesa appropriate.

Ciascuna classe è poi associata ad uno specifico weight che detta la velocità con cui la classe è selezionata rispetto alle altre.

6.3 PROTOCOLLO INTERNET (IP)

Il Protocollo Internet (IP) è il protocollo del livello network usato per garantire la delivery host-to-host. Vi sono due versioni di IP correntemente in uso:

- IP version 4 (IPv4) che è la più usata e la più comune
- IP version 6 (IPv6) che è la versione più nuova, proposta per rimpiazzare la IPv4.

La più importante funzionalità fornita da IP è quella di identificare gli hosts all'interno della network (indirizzamento IP).

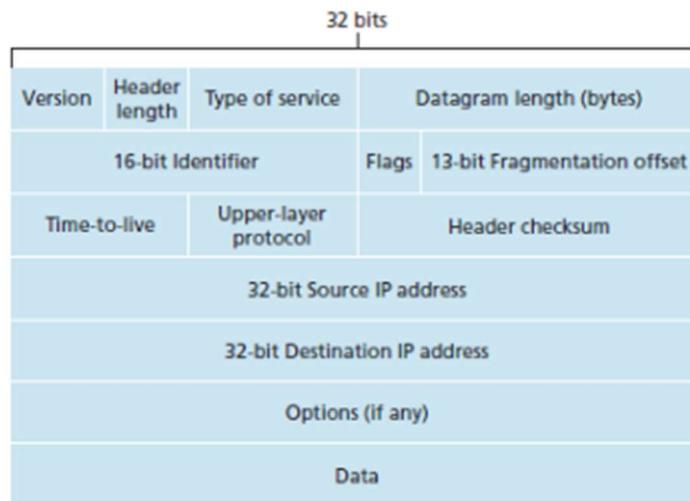
L'indirizzamento IP è il processo di assegnazione indirizzi IP ai dispositivi. L'indirizzamento è cruciale per le funzionalità della network ed è abbastanza complesso in internet (vi sono milioni o miliardi di hosts che devono essere indirizzati).

6.3.1 PROTOCOLLO INTERNET: DATAGRAMMA

Il pacchetto a livello network di internet è chiamato datagramma (come UDP). I campi chiave nel datagramma IPv4 sono:

- Il numero di versione (4 bits) che specifica la versione IP (ad esempio v4 o v6) del datagramma (i formati dipendono dalle versioni).
- La lunghezza dell'header (4 bits) è la dimensione dell'header (non fissa, le opzioni sono di dimensione variabile), usata per conoscere dove il payload inizia (no opzioni specificati indica un header di 20 bytes).
- Il tipo di servizio (TOS, 8 bits) identifica specifiche proprietà del datagramma. Tali tipi sono definiti dagli amministratori network dei routers.
- La lunghezza del datagramma (16 bits) che è la lunghezza in bytes del datagramma (header+data). I datagrammi sono raramente più grandi di 1500 bytes.
- L'identificatore (16 bits) è un numero progressivo che identifica in modo univoco un datagramma (usato nella frammentazione).
- Flags e offset della frammentazione (3+13 bits) che sono proprietà e offset del frammento (usati nella frammentazione).
- Time-to-live (TTL, 8 bits) assicura che i datagrammi non circolino per sempre nella network. Questo campo è decrementato di uno ogni volta che il datagramma è processato da un router. Se il campo TTL raggiunge 0, un router deve scartare quel datagramma.
- Il protocollo del livello superiore (8 bits) che indica il protocollo del livello di trasporto al quale il payload di questo datagramma IP dovrebbe essere passato (6 per TCP, 17 per UDP).
- Il checksum di header (16 bits) per il rilevamento degli errori. Qui il checksum è calcolato applicando il complemento ad 1 alla somma a 16 bits dell'intero header. I routers controllano questo valore e datagrammi con errori vengono di solito scartati.
 - Nota che il checksum deve essere ricalcolato e immagazzinato di nuovo per ogni router a causa dei campi TTL e opzionali che possono cambiare.

- Gli indirizzi IP di sorgente e destinazione (32+32 bits).
- Le opzioni (non fisse) permettono ad un header IP di essere esteso (funzionalità aggiuntive). Il campo opzione fornisce un certo grado di complessità (di dimensione sconosciuta), esso non è presente nel IPv6.
- Il data (non fisso) contiene l'attuale messaggio (payload) di solito nella forma di un segmento TCP/UDP a livello di trasporto.



6.3.2 PROTOCOLLO INTERNET: FRAMMENTAZIONE

Il primo problema con i datagrammi IP è che essi possono essere frammentati a livello link. Alcuni protocolli a livello link portano datagrammi di differenti dimensioni.

I datagrammi IP vengono eventualmente incapsulati nei frames del livello link per essere trasportati da nodo a nodo. Il massimo carico di dati (ad esempio la massima lunghezza di frame) che un frame a livello link può trasportare è la maximum transmission unit (MTU).

Un router che interconnette due links aventi differenti MTUs può ricevere un datagramma IP dal link di input che non è adatto al link dell'output.

Il router deve dividere il payload del datagramma IP in due o più piccoli datagrammi IP (frammenti). I frammenti vengono poi incapsulati nei frames a livello link e inoltrati tramite i collegamenti in uscita.

Quando un router frammenta un datagramma:

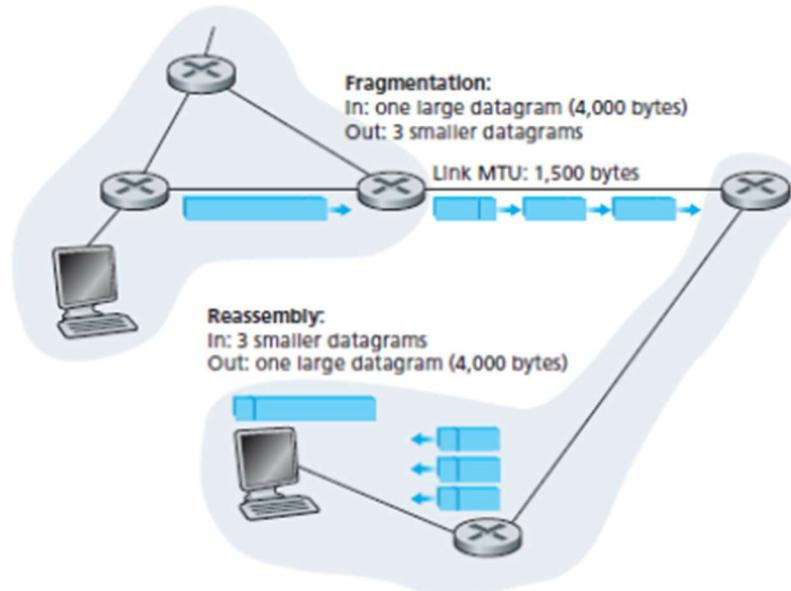
- Esso copia lo stesso identificatore, lo stesso indirizzo di sorgente e lo stesso indirizzo di destinazione nei nuovi segmenti.
- Setta il campo fragment offset di tutti i frammenti a numeri progressivi.
- Setta la flag dell'ultimo frammento a 1 (per segnalare che i frammenti sono finiti).

Chiaramente, i frammenti hanno bisogno di essere riassemblati prima che essi raggiungano il livello di trasporto alla destinazione in quanto sia TCP che UDP stanno aspettando di ricevere frammenti completi dal livello network.

I progettisti di IPv4 pensavano che riassemblare i datagrammi nei routers avrebbe introdotto significanti complicazioni nel protocollo impattando negativamente sulle prestazioni del router.

Il riassemblaggio del datagramma IPv4 viene poi eseguito dagli end systems:

- Se molteplici datagrammi aventi gli stessi indirizzi ed identificatori vengono ricevuti, significa che il datagramma originale è stato frammentato.
- L'host deve ricreare il datagramma originale dai frammenti.



6.3.3. PROTOCOLLO INTERNET: INDIRIZZAMENTO

Gli indirizzi IP sono tipicamente scritti in una notazione cosiddetta dotted-decimal, in cui ogni byte dell'indirizzo è scritto nella sua forma decimale ed è separato da un punto (dot) dagli altri bytes nell'indirizzo.

- For example:

	Dotted-decimal	Binary
IP address	193.32.216.9	11000001 00100000 11011000 00001001

Ciascun dispositivo nell'internet globale deve avere (in qualche modo) un indirizzo IP che è globalmente unico.

Poiché ogni indirizzo IP è lungo 32 bits (4 bytes) vi sono un totale di 2^{32} (quasi 4 miliardi) di possibili indirizzi IP.

Gli hosts e in generale i dispositivi network sono connessi attraverso links (cablati o wireless).

Un host di solito ha solo un singolo link nella network, mentre un dispositivo network (ad esempio router) può avere molteplici links.

Il confine tra l'host e il link fisico è chiamato interfaccia.

Poiché ogni host e router è capace di inviare e ricevere datagrammi IP, l'IP richiede che ogni interfaccia abbia il proprio indirizzo IP.

Sostanzialmente, un indirizzo IP è tecnicamente associato all'interfaccia, piuttosto che all'host o al router contenenti quell'interfaccia.

Assegnare gli indirizzi IP alle differenti interfacce non è banale. Potrebbe essere imprudente assegnare casualmente indirizzi IP per differenti ragioni, ad esempio:

- Poiché gli IPs non danno indicazioni circa la locazione, non si saprà dove trovare gli hosts.
- Si avrebbe bisogno di un enorme forwarding tables all'interno dei routers.

L'indirizzamento della network assomiglia a quello della telefonia standard: le networks sono gerarchicamente divise in sotto-networks (o sottoreti) aventi differenti prefissi.

Un indirizzo IP è poi diviso in due parti:

- 1. La prima porzione (leftmost) identifica la sottorete al quale il nodo è connesso.
- 2. La seconda porzione (rightmost) identifica la singola interfaccia.

Il numero di bits appartenenti ad ogni porzione non è fissato.

Nello specifico, per distinguere la parte di sottorete dalla parte interfaccia, un indirizzo IP è associato ad una maschera di sottorete che specifica quale bit dell'indirizzo appartiene alla parte di sottorete.

- For example:

	Dotted-decimal	Binary
IP address	193.32.216.9	11000001 00100000 11011000 00001001
Subnet Mask	255.255.255.0	11111111 11111111 11111111 00000000

Un altro modo comune per rappresentare le maschere è la slashed notation, ad esempio 193.32.216.0/24 rappresenta l'IP di una sottorete, dove /24 indica che i 24 bits della parte sinistra dell'indirizzo sono dedicati alla sottorete.

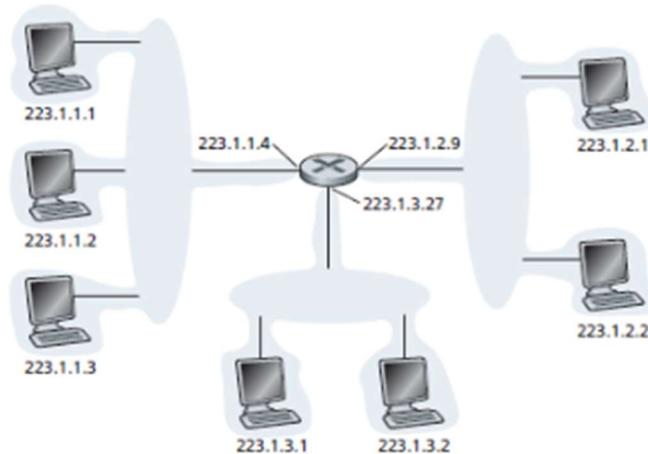
In questo esempio un router (con tre interfacce) è usato per interconnettere sette hosts. Gli hosts sono divisi in tre sottoreti (destra, sinistra, giù) ognuna delle quali è collegata ad un'interfaccia del router.

Ogni sottorete è associata ad uno specifico indirizzo, ad esempio la sottorete sinistra ha indirizzo 223.1.1.0/24, quindi tutte le interfacce in questa network hanno indirizzi IP nella forma 223.1.1.XXX:

- 223.1.1.1 (host)
- 223.1.1.2 (host)
- 223.1.1.3 (host)

- 223.1.1.4 (router)

Le altre due sottoreti hanno indirizzi 223.1.2.0/24 e 223.1.3.0/24.



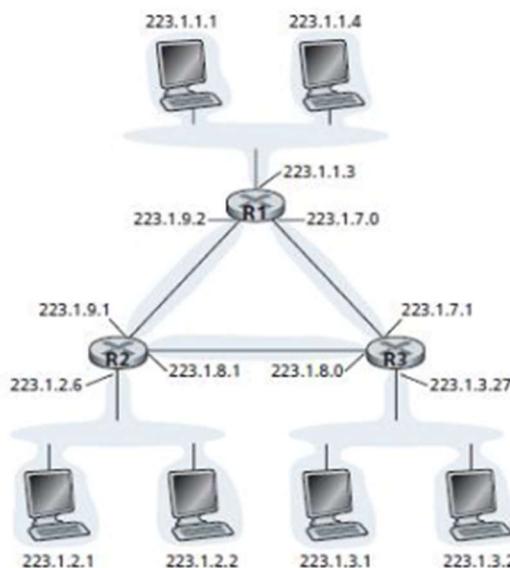
In questo esempio vi sono 3 routers che sono interconnessi attraverso un link diretto (point-to-point). Ogni router ha 3 interfacce, una per ogni link point-to-point e una per ogni link broadcast che direttamente connette il router ad una coppia di hosts.

Vi è un totale di 6 sottoreti:

- 223.1.1.0/24 (R1-hosts)
- 223.1.2.0/24 (R2-hosts)
- 223.1.3.0/24 (R3-hosts)
- 223.1.9.0/24 (R1-R2)
- 223.1.8.0/24 (R2-R3)
- 223.1.7.0/24 (R3-R1)

In questo caso i router sono come gates (gateways) che connettono differenti networks: se si potessero separare le interfacce da ogni router, si potrebbero avere 6 networks isolate.

Tipicamente, le organizzazioni medio-grandi (come compagnie o istituzioni accademiche) hanno molteplici sottoreti interconnesse.



IFCONFIG

Sulle macchine Linux possiamo controllare il nostro indirizzo usando il comando ifconfig (l'equivalente sulle macchine windows è ipconfig).

Ipconfig (configurazione interfaccia) fornisce la lista di tutte le interfacce networks della macchina assieme alla loro configurazione network (indirizzi, maschere ecc)

- Usage:

- To get the configuration:
 - \$ ifconfig

```
eth0      Link encap:Ethernet HWaddr 00:0F:20:CF:8B:42
          inet addr:217.149.127.10 Bcast:217.149.127.63 Mask:255.255.255.192
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:2472694671 errors:1 dropped:0 overruns:0 frame:0
                  TX packets:44641779 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:1761467179 (1679.8 Mb)   TX bytes:2870928587 (2737.9 Mb)
          Interrupt:28
```

Ifconfig output example
(Wikipedia)

INDIRIZZAMENTO DI INTERNET

L'idea di suddividere grandi networks in pezzi più piccoli è particolarmente importante in internet dove miliardi di dispositivi (senza menzionare le interfacce) devono essere connessi.

In internet, gli indirizzi IP devono essere attentamente assegnati per evitare alcuni problemi:

- Le forwarding tables dei routers possono diventare molto grandi
- Si potrebbero avere differenti interfacce con lo stesso indirizzo
- Potrebbero esaurirsi gli indirizzi disponibili

L'approccio è di dividere gli indirizzi internet fornendo sottoreti per le organizzazioni (quali IPs, compagnie, istituzione ecc). Vi sono due modi:

- 1. Indirizzamento di classe (più vecchio, non più usato in pratica)
- 2. Indirizzamento senza classe (in uso)

6.3.3.1 INDIRIZZAMENTO DI CLASSE

Nel classfull addressing gli indirizzi di internet erano divisi in classi basandosi su una specifica divisione:

	Format	Example	IPs per network
Class A	a.b.c.d/8	10.X.X.X	> 16 million
Class B	a.b.c.d/16	10.10.X.X	65535
Class C	a.b.c.d/24	10.10.10.X	254

Per esempio, se la tua organizzazione ha bisogno di 300 IPs, ti sarebbe stata assegnato un indirizzo di classe B assieme a tutti gli indirizzi IP in esso contenuti.

Vi è un chiaro problema con questo tipo di approccio, poiché sono necessari solo 300 IPs i restanti IPs sono sprecati!

6.3.3.2 INDIRIZZAMENTO SENZA CLASSE (CLASSLESS ADDRESSING)

Il moderno approccio è più flessibile ed è chiamato Classless InterDomain Routing (CIDR, pronunciato cider). In questo modo ad un'organizzazione potrebbero essere assegnati indirizzi network di qualsiasi forma: a.b.c.d/X

Se hai bisogno di 300 IPs ti verrà assegnato a.b.c.d/23 (ad esempio 241.115.2.0/23) che fornisce 512 IPs e soltanto 212 andrebbero sprecati.

Negli indirizzi “CIDRized”:

- La parte network dell'indirizzo è chiamata prefisso.
- Il set di IPs riservati all'organizzazione è chiamato blocco.

Nota: è possibile per i blocchi sovrapporsi, in questo caso la lunghezza (X) del prefisso può essere usata per discriminare differenti blocchi.

- Esempio: l'indirizzo IP 10.10.10.15/16 non è lo stesso blocco di 10.10.10.14/24.

Vi è la possibilità di creare sottoreti interne (sottoreti all'interno di un'organizzazione) all'interno di un blocco CIDR.

Per esempio, il blocco CIDR 241.115.0.0/16 può essere scomposto in sottoreti addizionali:

- 241.115.1.0/24 (sottorete 1)
- 241.115.2.0/24 (sottorete 2)
- 241.115.3.0/24 (sottorete 3)
- ecc

Da un punto di vista binario:



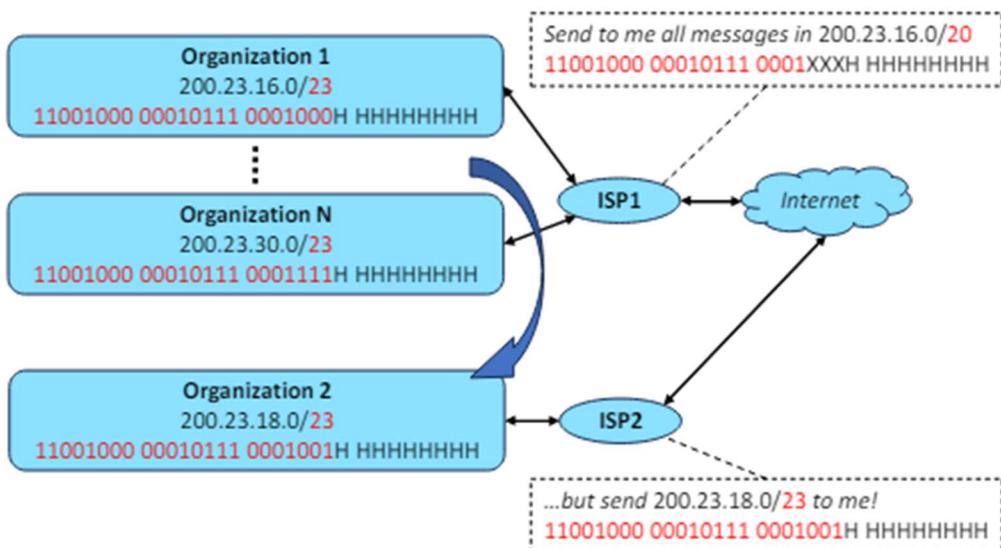
6.3.3.3 ADDRESS AGGREGATION

L'indirizzamento basato sul prefisso è molto utile per i dispositivi che connettono differenti prefissi.

È possibile per un router semplicemente ricordare i prefissi (ad esempio salvarli nella forwarding table).

Quando un messaggio con uno specifico prefisso viene ricevuto, è inoltrato ad un router più specifico, e così via.

Questo approccio è chiamato address aggregation (o riepilogo del percorso).



È più efficiente man mano che i blocchi vengono raggruppati!

6.3.3.4 OTTENERE UN BLOCCO

Per ottenere un blocco degli indirizzi IP per utilizzo all'interno della sottorete di un'organizzazione vi sono due modi:

- 1. Si può contattare un ISP, che potrebbe fornire indirizzi da un blocco più grande di indirizzi che è già stato allocato.
 - o Per esempio, l'ISP può esso stesso aver allocato un blocco di indirizzi 200.23.16.0/20, che può ulteriormente separare in sotto-blocchi di dimensione variabile a seconda della politica dell'ISP stesso.
- 2. Si può chiedere alla Internet Corporation for Assigned Names and Numbers (ICANN), che è un'autorità globale no-profit la quale ha la responsabilità di gestire lo spazio dell'indirizzo IP. ICANN gestisce anche i DNS root servers, assegnando i nomi di dominio e risolvendo le dispute sui nomi di dominio.

6.3.4 PROTOCOLLO DI INTERNET: ASSEGNAZIONE IP

All'interno di un blocco di indirizzi, gli indirizzi IP devono essere assegnati alle interfacce individuali.

Un amministratore di sistema può configurare gli indirizzi IP in due modi:

- 1. Manualmente: assegnando ad uno ad uno gli indirizzi IP agli hosts
- 2. Automaticamente: la network autonomamente assegna indirizzi IP liberi agli host in arrivo.

Il secondo approccio (il più comune) è messo in pratica usando il Dynamic Host Configuration Protocol (DHCP).

Oltre a fornire ad un host l'indirizzo IP, il DHCP fornisce ad un host le informazioni necessarie per entrare nella network, quali la maschera della sottorete, l'indirizzo del gateway di default e l'indirizzo del server locale DNS.

Il DHCP è plug-and-play, ed è solitamente usato nelle nostre case!

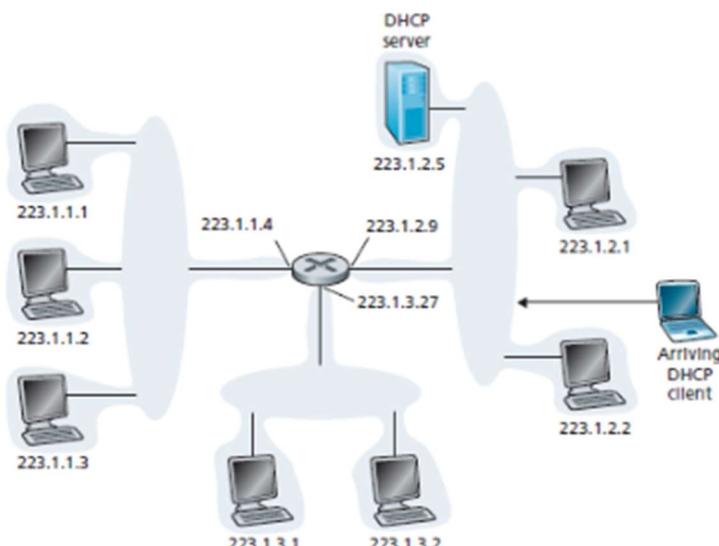
6.3.4.1 DHCP

Il DHCP è un protocollo client-server: un nuovo host in arrivo (client) si connette al server DHCP per ricevere informazioni sulla network. Il servizio DHCP potrebbe essere fornito da un computer o dal router stesso.

Guardiamo al precedente esempio delle 3 sottoreti connesse attraverso un singolo router.

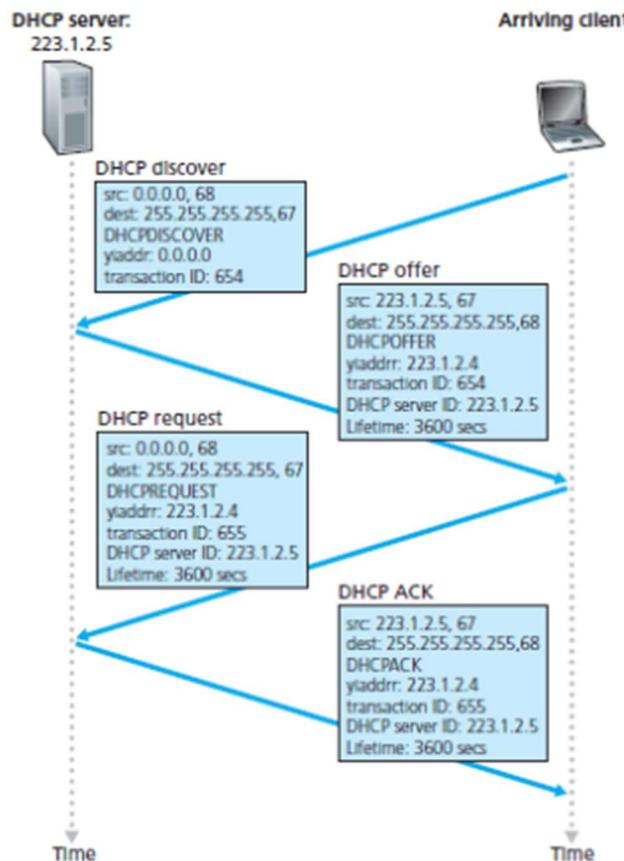
Qui assumiamo che la network a destra (233.1.2.0/24) sia dotata di un server DHCP.

Quando un nuovo host entra nella network, esso commercia l'indirizzo IP con il DHCP.



Aggiungere un nuovo host è un processo in 4 fasi:

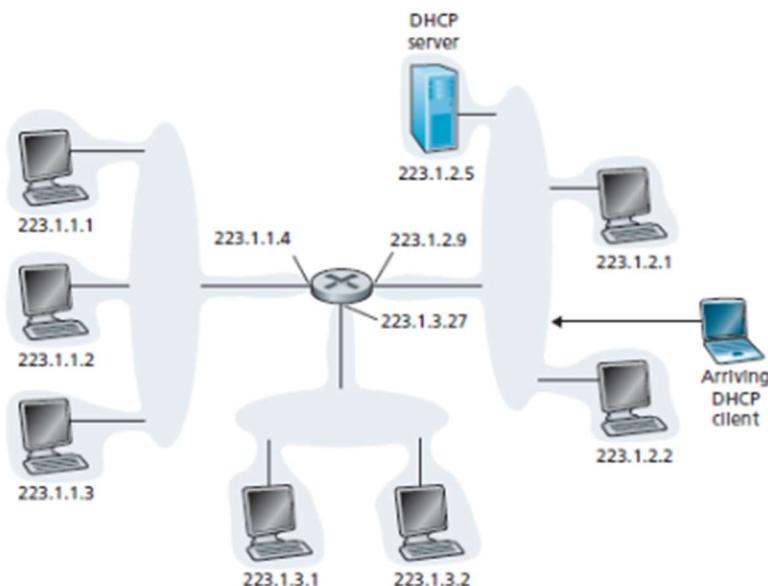
- 1. **DHCP server discovery:** il nuovo host manda in broadcast un DHCP discovery message (UDP sulla porta 67) per trovare il server DHCP.
 - IP di destinazione: 255.255.255.255 (broadcast)
 - IP di sorgente: 0.0.0.0 (questo host)
- 2. **DHCP server offer:** poiché molteplici servers DHCP possono essere presenti, quando un discovery message viene ricevuto, un server risponde con un messaggio di broadcast (sulla porta 68) offrendo una possibile configurazione della network (yiaddr file – Your Internet Address).
 - IP di destinazione: 255.255.255.255 (broadcast)
 - IP di sorgente: 233.1.2.5 (server DHCP)
- 3. **DHCP request:** il nuovo client seleziona l'offerta accettata inviando indietro il messaggio di offerta.
- 4. **DHCP ACK:** il messaggio ACK finale che conferma la transazione.



In questo caso tutto fila liscio in quanto il DHCP e l'host in arrivo sono nella stessa sottorete.

Se un server esiste ma in una differente sottorete, vi è bisogno di un agente di inoltro DHCP per inoltrare i messaggi DHCP (per esempio un router).

In questo esempio configuriamo il nostro router per farlo diventare un agente di inoltro così che anche gli hosts dalle sottoreti 223.1.1.0/24 e 223.1.3.0/24 vengano serviti dalla nostra DHCP.



ROUTE

In Linux possiamo usare il ifconfig in combinazione con il comando route per vedere la configurazione di network fornita da DHCP.

- Usage:

- See our current configuration
(IP address, mask):
 - \$ ifconfig
- See our default gateway:
 - \$ route -n

```
Lenovo-B50-80:~$ ifconfig wlp9s0
wlp9s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 100.103.0.64 netmask 255.255.0.0 broadcast 100.103.255.255
        inet6 fe80::47c4:ec38:2dc3:5d70 prefixlen 64 scopeid 0x20<link>
          ether 34:e6:ad:f0:d5:11 txqueuelen 1000 (Ethernet)
            RX packets 14398 bytes 20600768 (20.6 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 4100 bytes 451906 (451.9 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Lenovo-B50-80:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         100.103.0.1   0.0.0.0         UG    600    0        0 wlp9s0
```

6.3.5 PROTOCOLLO INTERNET: VISIBILITÀ'

In internet vi sono miliardi di dispositivi connessi che devono essere associati con indirizzi IP unici per essere raggiunti da tutti.

È davvero necessario avere tutti i dispositivi visibili a tutti?

Vi sono diversi problemi nell'avere tutti i dispositivi raggiungibili al di fuori delle reti locali:

- Gli indirizzi IP sono finiti
- Se i dispositivi sono localmente connessi è spesso irragionevole (o non desiderabile) esporli tutti su internet
- Gli amministratori locali dovrebbero essere pienamente consapevoli del sovrapporre la struttura a blocco IP per assegnare indirizzi non usati.

6.3.5.1 PROTOCOLLO INTERNET: NAT

Il servizio della Network Address Translation (NAT) permette di rimappare gli indirizzi IP di pacchetti all'interno di una network in altri differenti (network masquerading).

Ciò viene eseguito usando una traslation table che associa gli IPs o le porte locali (della LAN) agli IPs e alle porte globali (della WAN). Questo servizio può essere offerto dai routers.

La rete locale mascherata è anche chiamata network privato o regno con indirizzi privati. Gli indirizzi IP privati sono validi soltanto all'interno del network privato.

Questo è un esempio di come il router abilitato NAT sta connettendo un network ad internet.

Le quattro interfacce nel network locale hanno lo stesso indirizzo di sottorete di 10.0.0.0/24. Il router si comporta come un trasportatore di messaggi che sovrascrive gli IPs con uno specificato nella tabella:

- I pacchetti in uscita hanno i loro IPs di sorgente/porta sovrascritte con un singolo IP del lato WAN (138.76.29.7) e con una porta nuova (5001).
- I pacchetti in entrata hanno i loro IPs di destinazione/porta sovrascritte con uno specifico IP dal lato LAN dell'host (10.0.0.1) e con una porta iniziale (3345).

È importante notare che poiché gli host sono inconsapevoli del traffico degli altri hosts il NAT non può usare la porta iniziale in quanto molteplici hosts potrebbero selezionare la stessa porta simultaneamente.

Siccome la porta è di 16 bits, i NATs possono gestire più di 60.000 connessioni simultanee.

PING

Per controllare se un host è raggiungibile sulla network, si può usare il comando ping (ping è uguale sia sulle macchine Windows che sulle macchine Linux).

Ping (Packet Internet Groper) è una utility basata sul protocollo di trasporto ICMP (Internet Control Message Protocol) ed invia un pacchetto “echo request” ad un host che automaticamente risponde con un messaggio “echo reply”.

- Ping fornisce anche il tempo trascorso tra la richiesta e la risposta misurata in RTT.

• Usage:

- To check the reachability:
 - \$ ping [target address]

```
Lenovo-B50-80:~$ ping google.com
PING google.com (142.251.209.46) 56(84) bytes of data.
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=1 ttl=117 time=19.8 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=2 ttl=117 time=20.3 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=3 ttl=117 time=19.5 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=4 ttl=117 time=18.9 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=5 ttl=117 time=18.8 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=6 ttl=117 time=29.1 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=7 ttl=117 time=68.6 ms
64 bytes from mil04s51-in-f14.1e100.net (142.251.209.46): icmp_seq=8 ttl=117 time=19.6 ms
[...]
```

NMAP (PARTE 2)

Oltre alla scansione delle porte, il comando nmap può essere usato per mappare/fare lo scan ai dispositivi sulla network.

Esso fa affidamento su ping (ICMP) per scoprire gli hosts nella network.

• Usage:

- Ping-based hosts discovery:
 - \$ sudo nmap -sn [gate address]/[subnet bits]

```
Lenovo-B50-80:~$ sudo nmap -sn 100.103.0.1/16
Starting Nmap 7.80 ( https://nmap.org ) at 2023-11-03 09:41 CET
Nmap scan report for _gateway (100.103.0.1)
Host is up (0.0024s latency).
MAC Address: 10:8D:18:E5:74:80 (Cisco Systems)
Nmap scan report for 100.103.0.3
Host is up (0.43s latency).
MAC Address: 6C:20:56:2C:03:2F (Cisco Systems)
Nmap scan report for 100.103.0.11
Host is up (0.0031s latency).
MAC Address: 2A:FC:45:97:87:28 (Unknown)
Nmap scan report for 100.103.0.16
Host is up (0.0046s latency).
```

6.3.6 INTERNET PROTOCOLLO: INDIRIZZI IP RISERVATI

Siccome gli amministratori locali dovrebbero essere liberi di organizzare un network privato senza interferenze, secondo la convezione vi sono alcuni blocchi IP riservati, ad esempio quelli di ISPs e ICANN non possono essere assegnati a nessuna organizzazione. Tra gli esempi più famosi abbiamo:

- 255.255.255.255 che indica broadcast
- 192.168.0.0 – 192.168.255.255 per network private
- Ecc ecc

La necessità di riservare gli ISPs è data dal fatto che gli indirizzi privati potrebbero essere gli stessi di quelli pubblici (non privati), così che il routing all'interno della network potrebbe essere ambigua.

6.3.7 INTERNET PROTOCOLLO: ESEMPIO DI SETUP DI LAN

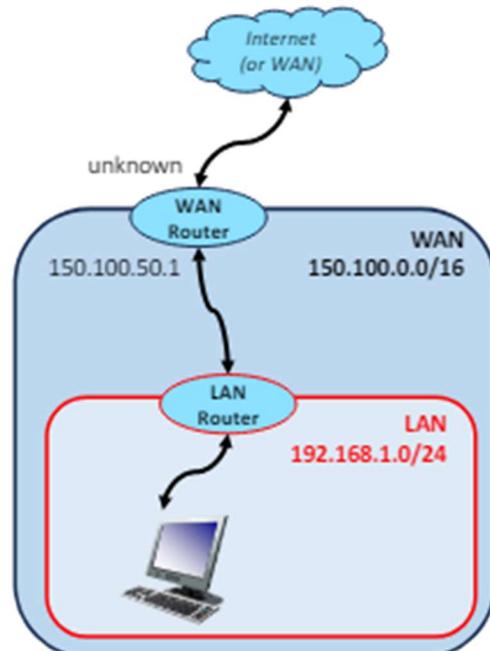
Vediamo un esempio in cui vogliamo creare una nuova sottorete locale (LAN) in una network preesistente (WAN).

L'amministratore della network della WAN ci fornisce le seguenti informazioni:

- L'IP della WAN: 150.100.0.0/16
- L'IP del gateway: 150.100.50.1
- Un IP libero per la nostra network: 150.100.50.10

Dobbiamo adesso configurare il nostro router e i nostri dispositivi per settare la nuova LAN

- Consideriamo un router abilitato per la NAT
- Decidiamo che l'IP della nuova LAN sarà 192.168.1.0/24



Il nostro router locale ha due interfacce, una per il lato WAN e una per il lato LAN.

Iniziamo configurando i settaggi dal lato WAN del nostro router locale.

Sul router dobbiamo specificare i seguenti parametri:

- L'indirizzo IP del router nella WAN
- La sottorete della WAN
- Il gateway
- Uno o più DNS

È importante notare che possiamo anche specificare un IP dinamico se la WAN ha un servizio DHCP.

Internet Connection

Set up an internet connection with the service information provided by your ISP (internet service provider).

Internet Connection Type: **Static IP**

Select this type if your ISP provides specific IP parameters.

IP Address: **150.100.50.10**

Subnet Mask: **255.255.0.0**

Default Gateway: **150.100.50.1**

Primary DNS: **8.8.8.8**

Secondary DNS: **4.4.4.4** (Optional)

Adesso configuriamo i settaggi dal lato della LAN.

Dobbiamo dare un IP all'interfaccia del router dal lato LAN:

- Siccome la nostra network sarà indirizzata come 192.168.1.0/24, daremo 192.168.1.1/24 al router.

In questo esempio, il nostro router fornisce anche il servizio DHCP, possiamo quindi specificare:

- Un pool di indirizzi (per gli host DHCP)
- Lease time (timeout dell'assegnamento dell'IP, dopo il quale l'IP può essere riutilizzato)
- Gateway e DNS che devono essere comunicati agli hosts

LAN

View and configure LAN settings.

MAC Address: **48-22-54-16-18-7D**

IP Address: **192.168.1.1**

Subnet Mask: **255.255.255.0**

DHCP Server

Dynamically assign IP addresses to the devices connected to the router.

DHCP Server: Enable

IP Address Pool: 192.168.1.100 - 192.168.1.250

Address Lease Time: 120 minutes

Default Gateway: 192.168.1.1 (Optional)

Primary DNS: (Optional)

Secondary DNS: (Optional)

Questo è un semplice esempio di come configurare questa connessione ad un nuovo host.

Ci sono differenti metodi da scegliere:

- Automatico (DHCP): usa DHCP per configurare automaticamente la connessione
- Manuale: settare la connessione settando manualmente la configurazione.

In un settaggio manuale vi è bisogno di conoscere la configurazione della network (e gli IPs disponibili).

Si devono specificare le principali informazioni per il setup della connessione:

- L'indirizzo IP dell'host (statico/fisso)
- La maschera di sottorete
- Il gateway
- DNS (uno o più)

New Profile Add

Identity IPv4 IPv6 Security

IPv4 Method Automatic (DHCP) Manual Link-Local Only Disable Shared to other computers

Addresses

Address	Netmask	Gateway	
192.168.1.10	255.255.255.0	192.168.1.1	

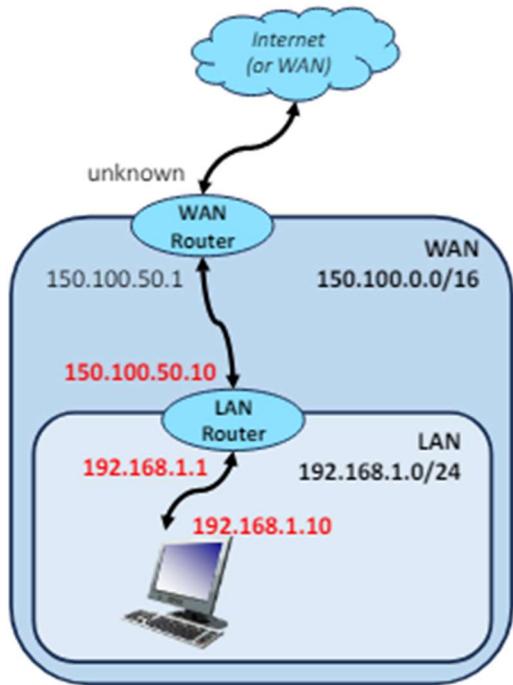
DNS Automatic
 8.8.8.8

Separate IP addresses with commas

In quella LAN così creata è possibile connettere nuovi hosts alla LAN specificando un indirizzo IP manuale (al di fuori del pool di DHCP) o un indirizzo IP automatico (all'interno del pool di DHCP).

Gli hosts connessi invieranno i loro pacchetti al router locale (inconsapevole della WAN esterna).

Abbiamo il router locale configurato in modo tale da inoltrare i pacchetti al router della WAN (e possibilmente ad Internet).



6.3.8 PROTOCOLLO DI INTERNET – IPv6

Negli primi anni 90, l'Internet Engineering Task Force (IETF) che è una ONG fondata nel 1986, iniziò uno sforzo per sviluppare un successore del protocollo IPv4 che potesse affrontare la carenza di indirizzi IPv4 a 32-bit.

IPv6 fu designato per assicurare più indirizzi disponibili, ma era anche un'opportunità di aggiornare altri aspetti di IPv4, basati sull'esperienza operazionale accumulata.

Non è certo quando esattamente tutti gli IPv4 disponibili sarebbero stati esauriti (fu ipotizzato il 2008 o il 2018, ma abbiamo ancora alcuni indirizzi IPs disponibili).

Ma cosa si può dire riguardo IPv5? Fu proposto nel 1979 ed era basato sullo sperimentale Internet Stream Protocol (ST). Faceva ancora affidamento sugli indirizzi a 32bit, così fu soppresso a causa della carenza di indirizzi.

IPv6 ha i seguenti vantaggi:

- Capacità di indirizzamento estese: dai 32 ai 128 bits, che assicura che il mondo non esaurisca mai gli indirizzi IP.

- Un header di una lunghezza fissa di 40 byte: alcuni dei campi di IPv4 sono stati scartati o resi opzionali
- Flow labeling: i pacchetti da alcune applicazioni (ad esempio audio/video streaming) possono essere raggruppati in un flusso unico aventi un'identificazione specifica.

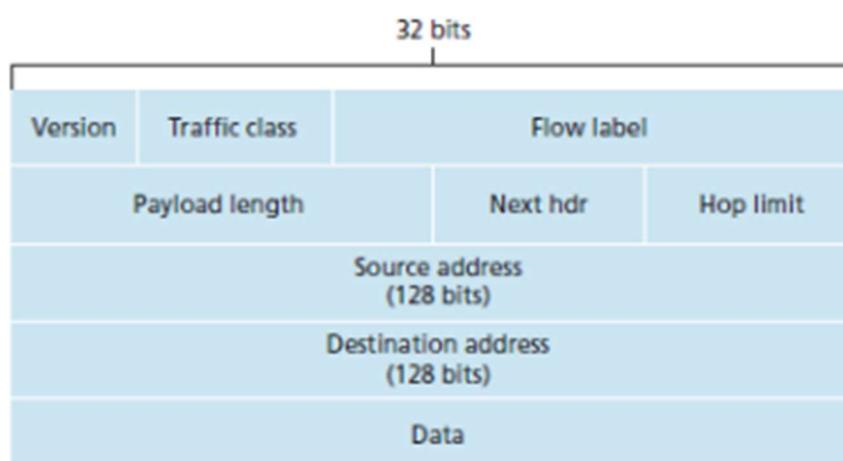
6.3.8.1 IL DATAGRAMMA IPv6

Il datagramma IPv6 è composto dai seguenti campi:

- Versione (4 bits): identifica il numero di versione IP. Non è una sorpresa che IPv6 porta un valore di 6 in questo campo.
- Classe di traffico (8 bits): come il campo TOS in IPv4, può essere usato per dare priorità ai datagrammi
- Flow label (20 bits): identifica un flusso di datagrammi
- La lunghezza del payload (16 bits): il numero di bytes del payload (esclusi l'header di 40 bytes)
- Il next header (8 bits): identifica il protocollo del livello superiore a cui i contenuti di questo datagramma (campo data) verranno consegnati (ad esempio a TCP o UDP). È simile al campo protocollo dell'header di IPv4.
- Hop limit (8 bits): specifica il time-to-live come nel campo TTL di IPv4
- Gli indirizzi di sorgente e destinazione (128+128 bits): gli indirizzi di 128 bit di IPv6
- Data (variabile): la porzione di payload del datagramma

Ciò che è mancante rispetto a IPv4 è:

- I campi relativi alla frammentazione: IPv6 non permette la frammentazione e il riassemblaggio sui routers intermedi, ma solo sugli hosts:
 - o Se un datagramma IPv6 ricevuto da un router è troppo grande per essere inoltrato, il router semplicemente fa cadere il datagramma e manda un errore di messaggio di "Pacchetto troppo grande" indietro al mittente. Il mittente può poi reinviare il messaggio, usando una dimensione di datagramma IP più piccola.
- Il checksum dell'header: esso richiede tempo sui routers, ed è ridondante come il checksum tipicamente eseguito dai protocolli a livello link sull'intero pacchetto.
- Le opzioni: non fanno più parte dell'header standard dell'IP, alcune opzioni possono essere specificate nel campo next header (assieme agli identificatori di TCP/UDP)



Vi è un grande problema con IPv6: non è compatibile con le versioni precedenti. I sistemi IPv4 sono incapaci di gestire i datagrammi IPv6.

Come internet adotterà IPv6?

- L'approccio flag day: un giorno tutte le macchine internet saranno spente ed effettueranno l'aggiornamento da IPv4 a IPv6. Un simile approccio è stato usato quando TCP fu introdotto ma fu un disastro (persino per le network piccole del tempo). Un day flag che coinvolge miliardi di dispositivi è persino maggiormente impensabile oggigiorno.
- L'approccio tunneling: l'uso di specifici routers (tunnels) incaricati di mappare i datagrammi IPv4 nei datagrammi IPv6 e viceversa in un modo quasi trasparente. Questo è probabilmente l'approccio più realistico e già usato nella pratica.

Mentre l'adozione di IPv6 fu inizialmente lenta, adesso sta accelerando. Google riporta che circa il 45% dei clients (nel 2023) quando accedono ai servizi Google usano IPv6.

6.4 ROUTING

Alla ricezione di un pacchetto, un router può eseguire 3 azioni:

- 1. Inoltrare il pacchetto così com'è ad un neighbor adatto
- 2. Inoltrare un pacchetto modificato/rimpiazzato (ad esempio frammentazione, mascheramento, ecc)
- 3. Scartare il pacchetto (ad esempio se scaduto)

I routers sono dotati di forwarding tables che specificano la strategia di inoltro locale (nel vicinato del router). Il modo in cui queste tabelle sono create può essere centralizzato o decentralizzato.

Per decidere dove i pacchetti dovrebbero essere inoltrati i routers fanno uso di algoritmi di routing, che determina buoni percorsi (ad esempio sequenza di percorsi) dai mittenti ai destinatari.

Tipicamente, un “buon” percorso è quello che ha il costo minimo (più corto o più veloce ecc), nel mondo reale vi sono anche dei problemi di politica da prendere in considerazione (ad esempio le regole che specificano se un'organizzazione può parlare con un'altra ecc)

6.4.1 ROUTING: FORMULAZIONE DEL PROBLEMA

Possiamo formalizzare la nostra network come un grafo $G = (N, E)$ dove:

- N è un set di nodi (i routers della nostra network)
- $E \subseteq N \times N$ è un set di archi (links fisici) rappresentati come una coppia di nodi

Un arco ha anche un valore che rappresenta il suo costo, che può riflettere la lunghezza fisica del link, la velocità del link, o il costo monetario associato al link.

Possiamo formalizzare un costo generico come una funzione $c: N \times N \rightarrow \mathbb{R}$ così che, data una generica coppia di nodi $(x, y) \in N \times N$:

- $\forall x \in N, c(x, x) = 0$
- Se $(x, y) \notin E$, allora $c(x, y) = \infty$
- Se $(x, y) \in E$, allora anche $(y, x) \in E$ (grafo non orientato) e $c(x, y) = c(y, x)$

Seguendo questa formulazione, un percorso π in G è una sequenza di nodi $\pi = (x_1, x_2, \dots, x_n)$ tale che tutti i nodi adiacenti siano collegati, formalmente:

$$\forall x_i, x_{i+1} \in \pi, (x_i, x_{i+1}) \in E$$

Il costo generale del percorso è pari alla somma dei costi dei link che lo compongono:

$$c(\pi) = \sum_{i=1}^{n-1} c(x_i, x_{i+1})$$

Indicando con $P(x, y)$ il set di tutti i possibili percorsi che connettono i nodi x e y , possiamo definire con $\pi^* \in P(x, y)$ il miglior percorso avente costo minimo:

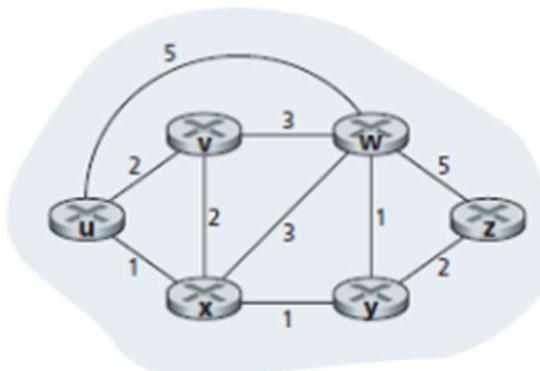
$$\forall \pi \in P(x, y), c(\pi^*) \leq c(\pi)$$

Nel seguente esempio abbiamo un grafo G composto da 6 nodi:

- $N = \{u, v, w, x, y, z\}$
- $E = \{(u, w), (u, v), (w, x), \dots\}$
- $c(u, w) = 5, c(u, v) = 2, c(u, x) = 1, \dots$

Per esempio, il percorso migliore π^* tra il nodo x e z è:

- $\pi^* = (x, y, z)$
- $c(x, y) = 1, c(y, z) = 2$
- $c(\pi^*) = 3$



Gli algoritmi che sono in grado di trovare il percorso ottimale π^* tra due nodi sono chiamati algoritmi shortest path.

Nel caso delle computer networks, ci interessa trovare il percorso ottimale non solo tra due nodi ma tra tutti le possibili coppie di nodi.

Un algoritmo di routing converge quando viene trovato nella network il percorso più breve tra tutte le coppie di nodi.

Nel campo delle network, ci sono 2 famiglie di algoritmi che vengono tipicamente utilizzati:

- Distance-Vector.
- Link-State.

6.4.2 ROUTING: ALGORITMO DISTANCE-VECTOR

L'algoritmo Distance-Vector (DV) è un approccio decentralizzato che sfrutta la conoscenza locale circa la network combinata con la comunicazione dei nodi per trovare i percorsi più brevi.

- È basato sull'algoritmo Bellman-Ford che è usato per computare i percorsi

Qui ogni nodo riceve alcune informazioni da uno o più dei suoi neighbors direttamente collegati, esegue un calcolo e poi distribuisce i risultato del suo calcolo ai suoi neighbors.

Questo algoritmo è asincrono in quanto non richiede che tutti i nodi siano coordinati l'uno con l'altro.

Chiamiamo $d^*(x, y)$ il costo (distanza) del miglior percorso dal nodo x al nodo y . Possiamo applicare l'applicazione Bellman:

$$d^*(x, y) = \min_v \{c(x, v) + d^*(v, y)\}$$

Dove v è un neighbor di x . Questo è abbastanza intuitivo, se v è il primo nodo del miglior percorso, allora il resto dei nodi devono essere quelli nel percorso migliore da v a y .

Se siamo capaci di trovare tale v possiamo semplicemente aggiungerla nella forwarding table e inoltrare a lui tutti i pacchetti diretti a y .

Per fare ciò, dovremmo avere una stima di $d^*(x, y)$ per tutti i neighbors (il vettore della distanza).

Il pseudocodice:

```

DistanceVector(x)
for all nodes v
  if v is a neighbor of x then
     $D_x(v) = c(x,v)$ 
  else
     $D_x(v) = \infty$ 
  for all neighbors w and destinations y
     $D_w(y) = ?$ 
    send initial  $D_x(\cdot)$  to all neighbors

repeat
  if cost of a neighbor updated then
    for all nodes y
       $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$ 
    if  $D_x(y)$  changed for any destination y then
      send updated  $D_x(\cdot)$  to all neighbors
until false //forever

```

Usiamo la data-structure $D_w(v)$ per immagazzinare i vettori di distanza da tutti i nodi w ai nodi target v .

Durante la fase di inizializzazione:

- Solo le distanze dai neighbors sono settate.

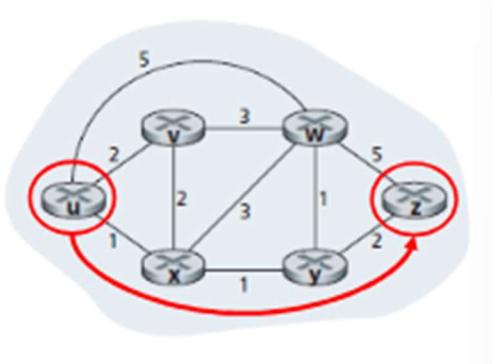
Durante la fase online:

- Le informazioni dal vicinato vengono ricevute.
- I vettori di distanza vengono aggiornati (equazione di Bellman)
- I cambi locali vengono comunicati ai nodi adiacenti.

Per chiarire questo processo ci focalizziamo soltanto su uno specifico percorso, tra u e z :

- 1. All'inizio, u conosce soltanto i suoi neighbors. Il costo a z è infinito ($D_u(z) = \infty$)
- 2. Il nodo y si sveglia e scopre un percorso migliore per arrivare a z , che è il link diretto $y-z$ con costo pari a 2 ($D_y(z) = c(y,z) = 2$). Esso comunica le "buone" informazioni a x .
- 3. Il nodo x scopre che il miglior percorso per arrivare a z adesso passa attraverso y con il costo pari a 3 ($D_x(z) = c(x,y) + D_y(z) = 1 + 2$), e inoltra le "buone" informazioni ad u .
- 4. Il nodo u riceve le informazioni. Ora vi è un percorso per arrivare a z che costa meno di infinito ed è quello che passa per x ($D_u(z) = c(u,x) + D_x(z) = 1 + 3$).

Lo stesso processo accade per tutti i nodi/percorsi.



In termini di complessità computazionale, ogni volta che il costo di un nodo viene aggiornato, tutti i nodi devono rivalutare i propri archi. Questo processo conduce tipicamente ad una complessità del caso peggiore di $O(N \cdot E)$.

Pro:

- L'algoritmo è asincrono, ciò facilita la computazione poiché i routers possono stare online e in ogni stato adattarsi ai costi aggiornati.

Contro:

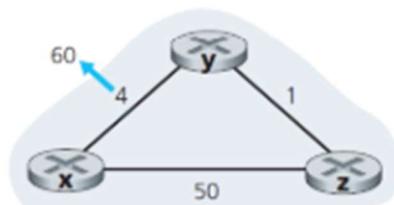
- Convergenza lenta: gli aggiornamenti si diffondono lentamente siccome tutti i nodi devono individuare il cambiamento ed inviare un aggiornamento a quelli adiacenti.
- Count-to-infinity: siccome solo i percorsi migliorati vengono comunicati, se un link o un nodo fallisce, la propagazione di questa "cattiva" notizia sarà lenta: il nodo adiacente ad un link rotto immediatamente switcherà ad un percorso alternativo, ma essi non avranno la certezza che il miglior percorso non passi attraverso il fallimento.

6.4.2.1 ROUTING: COUNT-TO-INFINITY

Consideriamo questo esempio semplificato in cui il link x-y subisce un improvviso incremento del costo.

- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

This is the **initial situation** before the increment ($c(x,y)$ is still 4).



- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

Node y detects the increment and updates distance-vector to x:

$$D_y(x) = \min\{ 60, 1 + D_z(x) \} = 1 + D_z(x) = 6,$$

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- $D_y(x) = 4,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

But vector $D_y(x)$ is used by z to compute the path to x , so also $D_z(x)$ must be updated!

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 7,$
- $D_z(y) = 1.$

- $D_y(x) = 4,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

But vector $D_z(x)$ is itself used by y to compute the path to x , so $D_y(x)$ must be updated again!

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 7,$
- $D_z(y) = 1.$

- $D_y(x) = 1 + D_z(x) = 8,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 7,$
- $D_z(y) = 1.$

- $D_y(x) = 4,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

This is a loop: the 2 vectors will be continuously updated until a stable configuration is reached.

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 5,$
- $D_z(y) = 1.$

- $D_y(x) = 1 + D_z(x) = 6,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 9,$
- $D_z(y) = 1.$

- $D_y(x) = 1 + D_z(x) = 8,$
- $D_y(z) = 1,$
- $D_x(y) = 4,$
- $D_x(z) = 4 + D_y(z) = 5,$
- $D_z(x) = 1 + D_y(x) = 7,$
- $D_z(y) = 1.$

6.4.3 ROUTING: ALGORITMO LINK-STATE

L'Algoritmo Link-State (LS) è un approccio centralizzato che sfrutta la piena conoscenza circa la network per trovare il miglior percorso.

- Gli algoritmi LS sono basati sulla variante dell'algoritmo Dijkstra.

È importante notare che tale livello di conoscenza può essere ottenuto in pratica avendo ciascun nodo che invia pacchetti link-state in broadcast, contenenti IDs e costi dei link a loro collegati, a tutti gli altri nodi della network.

La versione di base dell'algoritmo computa i percorsi più brevi da un nodo di partenza a tutti i possibili nodi di destinazione (pertanto deve essere eseguito su tutti i nodi).

Il pseudocodice:

```
LinkState(x):
    N' = {x}
    for all nodes v
        if v is a neighbor of x then
            D(v) = c(x,v)
        else
            D(v) = ∞

    repeat
        find w not in N' such that D(w) is a minimum
        add w to N'
        update D(v) for each neighbor v of w and not in N':
            D(v) = min( D(v), D(w) + c(w,v) )
    until N'= N
```

Noi usiamo la data-structure $D(v)$ per immagazzinare la distanza dal nodo corrente da tutti i possibili nodi di destinazione.

Durante la fase di inizializzazione:

- Assumiamo che tutti i costi sono conosciuti!
- Solo le distanze dai neighbors sono settate.

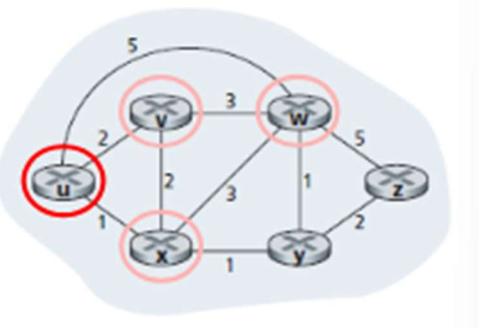
Durante la fase di costruzione:

- Viene selezionato il nodo più vicino w .
- Si esplora il vicinato di w , controllando se un percorso attraverso w è migliore di quello corrente.
- Si esce quando tutti i nodi sono stati esplorati.

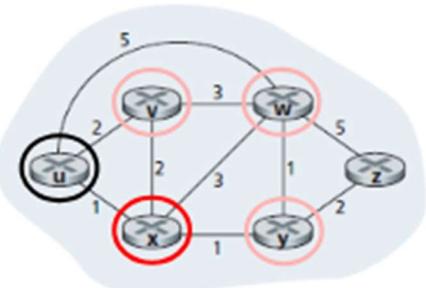
Consideriamo l'esempio precedente dei 6 nodi. Qui mostriamo come l'algoritmo LS lavora quando invocato sul nodo u :

- In questo caso, useremo una funzione aggiuntiva $p(x)$ per immagazzinare il nodo precedente a quello selezionato.
- Se vogliamo recuperare il percorso, possiamo semplicemente andare a ritroso per tutti i predecessori fino a quando il nodo corrente viene raggiunto.

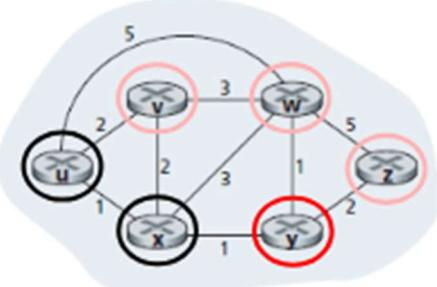
step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y		4, y	
3	$uxyw$		3, y		4, y	
4	$uxywz$				4, y	
5	$uxywz$					



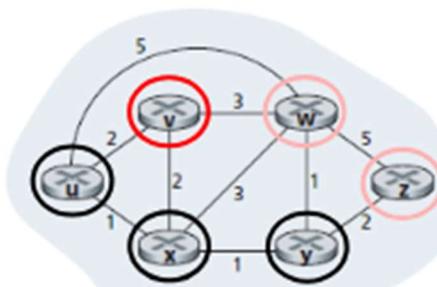
step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y		4, y	
3	$uxyw$		3, y		4, y	
4	$uxywz$				4, y	
5	$uxywz$					



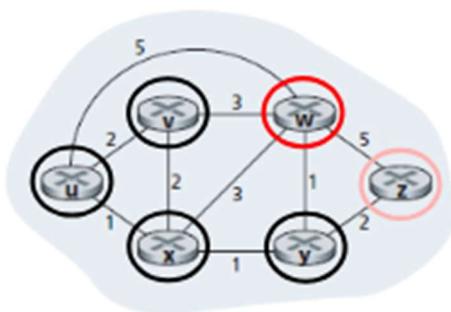
step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y		4, y	
3	$uxyw$		3, y		4, y	
4	$uxywz$				4, y	
5	$uxywz$					



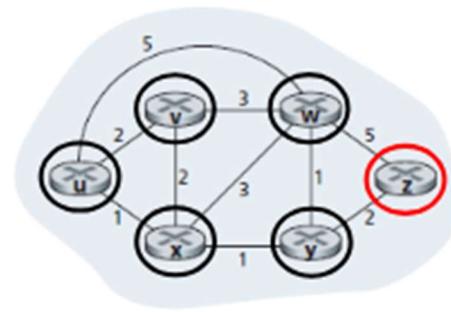
step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y		4, y	
3	$uxyw$		3, y		4, y	
4	$uxywz$				4, y	
5	$uxywz$					



step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uwy	2, u	3, y		4, y	
3	uwyv		3, y		4, y	
4	uwyvw				4, y	
5	uwyvwz					



step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uwy	2, u	3, y		4, y	
3	uwyv		3, y		4, y	
4	uwyvw				4, y	
5	uwyvwz					



In termini di complessità computazionale, ad ogni iterazione controlliamo tutti i nodi della network, eccetto il nodo sorgente, poi una nuova sorgente viene selezionata. Ciò significa che stiamo rimuovendo un nodo per ogni iterazione, così che il numero totale di interazioni sia pari a $|N|(|N| + 1)/2$ che è $O(|N|^2)$ nel caso peggiore.

Questa versione dell'algoritmo è piuttosto basilare, la sua complessità può essere ridotta introducendo strutture dati più sofisticate (ad esempio gli heaps), possiamo quindi raggiungere una migliore performance: $O(|N| + |E| \log(|E|))$.

Pro:

- Convergenza più veloce: tutte le comunicazioni sono eseguite simultaneamente.
- Nessun count-to-infinity: lo stato di tutti i link è propagato.

Contro:

- Sincrono: i nodi devono ricevere informazioni da parte dell'intera network prima di cominciare.

6.4.4. TRACEROUTE

In Linux possiamo usare traceroute per tracciare il percorso dei nostri pacchetti verso una specifica destinazione:

- Install traceroute:
 - `$ sudo apt-get install traceroute`
- Trace a route:
 - `$ traceroute ADDRESS`

Il comando traceroute restituirà gli indirizzi IP di tutti i dispositivi incontrati dall'host sorgente all'host di destinazione.

Se proviamo diverse volte a settare come target un host distante (ad esempio google.com) potremmo vedere diversi dispositivi nella lista (dovuto all'aggiornamento delle rotte).

6.4.5. DV contro LS

Gli algoritmi DV ed LS hanno approcci complementari verso la computazione del routing:

- L'algoritmo DV sfrutta le informazioni locali riguardante i neighbors
- L'algoritmo LS richiede informazioni globali riguardo tutti i nodi

Complessità del messaggio: LS è più complesso in quanto richiede comunicazione sincrona tra tutti i nodi. In DV, la comunicazione è necessaria soltanto se il percorso migliore cambia.

Velocità di convergenza: DV richiede tempo per convergere, nel frattempo potremmo imbatterci in percorsi sotto ottimali oppure loops. Gli algoritmi DV inoltre soffrono del problema del count-to-infinity.

Robustezza: LS è considerato più robusto in quanto le forwarding tables sono calcolate separatamente. Ciascun nodo riceve informazioni dagli altri e crea la sua propria tabella. In DV tutte le computazioni sono a catena, ciascun nodo dipende dalla tabella degli altri. Se una tabella non è corretta, tutte le tabelle si imbattono in un errore.

- Nel 1997 un malfunzionamento di un router da parte di una piccola ISP ha causato una reazione a catena, che ha inondato la backbone dei routers e ha costretto la disconnessione di parte di Internet per diverse ore.

Alla fine dei conti non vi è un vincitore, infatti entrambe le soluzioni sono utilizzate in Internet.

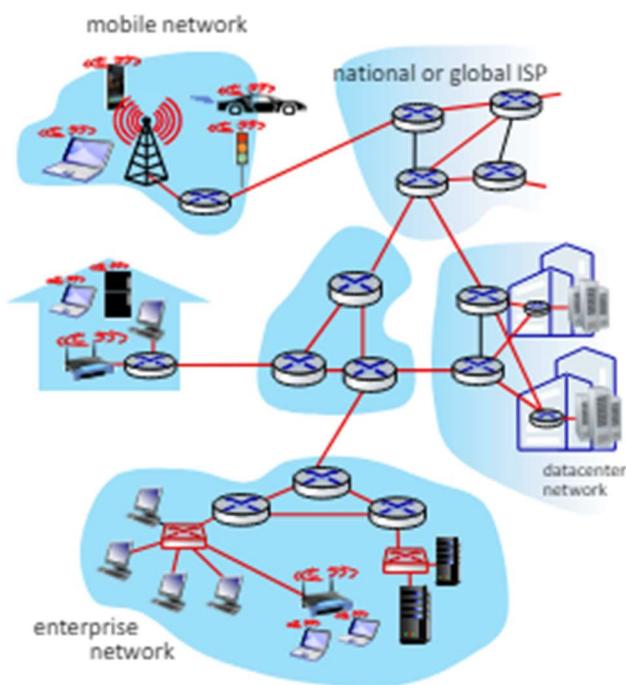
7 LIVELLO LINK

7.1 DAL LIVELLO NETWORK AL LIVELLO LINK E FISICO

Si è visto che il livello network di base fornisce il livello di comunicazione tra i due hosts (ovunque essi siano).

I livelli link e fisico forniscono la comunicazione tra due hosts connessi:

- Il livello link è la porzione dello stack dedicata alla trasmissione dei pacchetti sui collegamenti (canali di trasmissione), da un nodo all'altro della network.
- Il livello fisico è la porzione di stack che regola la struttura dei collegamenti (il mezzo di trasmissione, i connettori, i tipi di cablaggio ecc) e come i bits vengono rappresentati/trasmessi lungo il collegamento.



I livello link e fisico sono strettamente intrecciati (questi spesso sono raggruppati in un singolo livello chiamato accesso alla network).

- Alcuni protocolli comuni (ad esempio Ethernet) coprono entrambi i livelli.

Differentemente dai livelli precedenti, questi due livelli sono presenti in tutti i pezzi dell'infrastruttura della network:

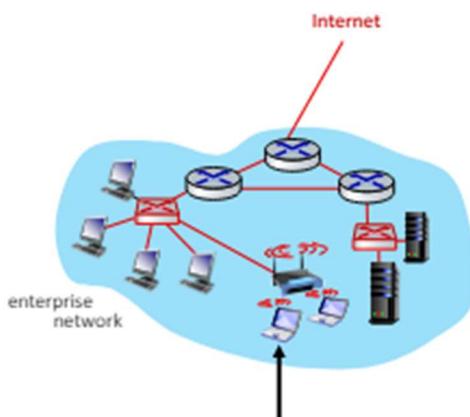
- Nodi: hosts (PC, servers, ecc), routers, switches, hubs, WIFI access point, ecc
- Links: cablati (cavi di rame, fibra ottica), wireless (radiofrequenze).

Per trasmettere un datagramma IP, lo si deve trasferire dall'host di sorgente all'host di destinazione saltando su ciascuno dei singoli dispositivi/collegamenti lungo il percorso.

Per essere trasferiti tramite un collegamento, i datagrammi vengono incapsulati nei frames del livello link a seconda dello specifico tipo di collegamento.

Questa è l'ultimo incapsulamento, i frames vengono convertiti in segnali e trasferiti sul collegamento seguendo le sue specifiche. I segnali possono essere:

- Impulsi elettrici
- Luce
- Onde radio



In order to reach Internet, packets of this host have to cross 4/5 links (wireless and wired), 1 access point, 1 switch and 2/3 routers.

7.2 SERVIZI

Il livello link può offrire fino a 4 servizi: framing, link access, reliable delivery, error detection and correction.

- 1. **Framing:** incapsulare i datagrammi nei frames del livello link avente il datagramma del livello superiore come payload e uno specifico header/tailer. La struttura del frame dipende dai protocolli a livello link/fisico.
- 2. **Link access:** definire le regole che regolano l'accesso al link attraverso i mezzi del protocollo Medium Access Control (MAC). Tali regole dipendono dall'architettura del link:
 - o Per i collegamenti point-to-point (un singolo mittente ed un singolo destinatario), il protocollo MAC è semplice (o inesistente). Il mittente può inviare un frame ogni volta che il collegamento è inattivo.
 - o Per i collegamenti broadcast vi è il cosiddetto problema dell'accesso multiplo. Qui il protocollo MAC serve a coordinare le trasmissioni dei frames di diversi nodi.
- 3. **Reliable delivery:** garantisce che i frames trasmessi lungo il collegamento siano stati ricevuti.
 - o Un servizio di consegna affidabile a livello link è spesso usato per i collegamenti che sono soggetti a tassi di errore elevati, come un collegamento wireless, con l'obiettivo di correggere un errore localmente.
 - o Esso potrebbe produrre un forte sovraccarico. Poiché altri protocolli di applicazione/trasporto (ad esempio TCP) sono affidabili, questa caratteristica non è sempre implementata.

- 4. Error detection and correction: l'hardware a livello link può essere influenzato dal problema dell'inversione dei bit (bit flipping problem). Diversi protocolli a livello link implementano strategie di controllo/correzione più sofisticate (e implementate nell'hardware) in aggiunta a quelli provenienti dai livelli superiori (TCP e checksum di IP).
 - o Poiché non vi è ulteriore incapsulamento, questi controlli sostanzialmente coprono l'intero messaggio.
 - o I controlli implementati nell'hardware possono essere molto più veloci.

7.3 IMPLEMENTAZIONE

Nei dispositivi della network (end-systems, routers ecc) le funzionalità a livello link sono implementate sia dal lato del software e (maggiormente) dal lato dell'hardware.

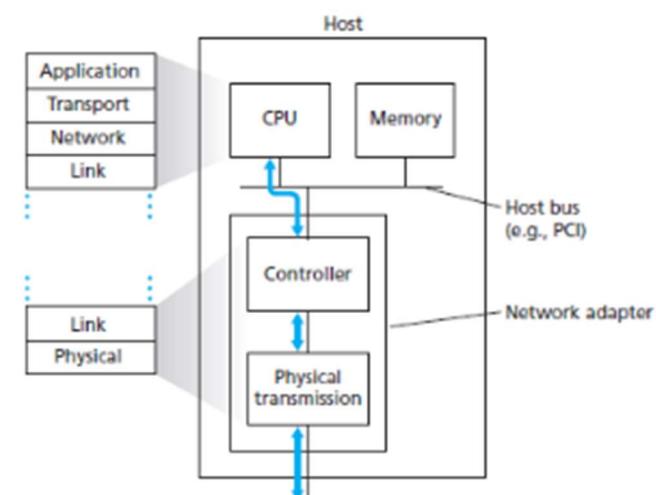
Nei computeri vi sono adattatori di network chiamati Network Interface Cards (NICs) aventi un chip specializzato (controller) per implementare le funzionalità a livello del link.

- Queste cards di solito erano separate dalla scheda madre (inseriti nei PCI slots) ma recentemente questo approccio sta cambiando (LAN-on-motherboard).

Software: esegue sulla CPU e fornisce funzionalità di alto livello quali indirizzamento, gestione dell'hardware e degli interrupt, gestione degli errori, incapsulamento/decapsulamento dei datagrammi.

- A livello link si può avere un indirizzo aggiuntivo, l'indirizzo MAC che è usato per identificare il NIC.
- I datagrammi devono essere immagazzinati nella memoria per essere usati dal protocollo del livello superiore. Essi devono essere:
 - o Recuperati dalla memoria durante l'incapsulamento (dal lato del mittente)
 - o Inseriti nella memoria durante il decapsulamento (dal lato del destinatario).

Hardware: lavora come un tipico dispositivo I/O, che converte i dati nel segnale da essere trasmesso a seconda del protocollo fisico (ad esempio ethernet, wireless ecc)



7.4 TECNOLOGIE FISICHE

Il protocollo a livello link, quindi la struttura frame risultante, dipende dalla tecnologia del collegamento fisico (livello fisico).

Qui alcune tecnologie comuni:

- Ethernet 802.3: la connessione cablata basata su impulsi elettrici su 4 coppie di cavi intrecciati di rame (più comuni) o i fotoni sui cavi della fibra ottica.
- Wifi 802.11: connessione wireless basata su radiofrequenza (2.4 GHz, 5GHz, 6GHz)
- Bluetooth: connessione wireless basata su radiofrequenza (2.4 GHz)

Le tecnologie fisiche sono continuamente in evoluzione, i loro limiti in termini di distanza/bandwidth sono spesso aggiornati.

Differenti tecnologie hanno differenti pro e contro.

Type	Signal	Distance	Bandwidth	
Wireless	Radio	30-50 m	1.3 Gbps	Security, speed
Twisted pair copper cables	Electricity	30-100 m	1-25 Gbps	Interferences
Fiber-optic cable	Light	100-200 km	1-400+ Gbps	Fragile, expansive

7.5 RILEVAMENTO DEGLI ERRORI E CORREZIONE: FORMULAZIONE DEL PROBLEMA

Nel livello link è possibile eseguire il rilevamento e la correzione degli errori a livello bit.

Si assume che D di dimensione d sia il nostro dato, per proteggerlo dagli errori di bit includiamo nel messaggio alcuni bits di correzione e di rilevamento degli errori: EDC .

L'obiettivo è di scoprire se i ricevuti D' e EDC' differiscono dall'originale D e EDC e se sì possibilmente recuperare l'iniziale D e EDC .

Nota che persino con i bits di rilevamento degli errori potrebbero esserci ancora errori di bits non rilevati, il destinatario potrebbe essere inconsapevole del fatto che le informazioni ricevute possano contenere degli errori di bit.



7.5.1 PARITY CHECK

Il parity check è forse la più semplice forma di rilevamento degli errori: includiamo solo 1 parity bit aggiuntivo al messaggio così che il numero totale degli 1 tra i $d+1$ bits del messaggio sia pari (even parity scheme) o dispari (odd parity scheme).

Message	Priority bit (even scheme)	Priority bit (odd scheme)
10101100	0	1
11010000	1	0

Il destinatario necessita soltanto di contare il numero degli 1 nei $d+1$ bits ricevuti. Se un numero dispari di bits con valore 1 viene trovato con uno even parity scheme (o viceversa), il destinatario sa che almeno un errore di bit si è verificato.

Chiaramente questo approccio funziona soltanto se si verifica un numero dispari di errori di bit, ma allora quanto spesso capita?

Se la probabilità di errori di bit è bassa e si può assumere che gli errori si verifichino indipendentemente da un bit al prossimo, la probabilità di multipli errori di bits in un pacchetto dovrebbe essere estremamente bassa, ma questo non è il caso delle computer networks.

In pratica, è stato osservato che gli errori sono spessi raggruppati insieme in bursts (per niente indipendenti l'un l'altro). Sotto le condizioni di errore burst, le probabilità di errori non individuati utilizzando una parità a singolo bit può raggiungere il 50%, il che non è per niente utile.

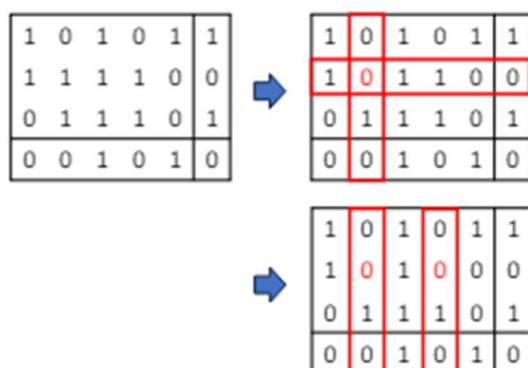
PARITY CHECK (BIDIMENSIONALE)

Un possibile modo di migliorare questo approccio è di usare molteplici parity bits.

La two-dimensional parity è una tecnica in cui il messaggio viene diviso in n righe e m colonne, ognuna delle quali avendo uno specifico parity bit.

Qui il destinatario può individuare che un errore si è verificato, e anche quale bit, quindi può tentare una correzione.

La two-dimensional parity può anche individuare (ma non correggere) qualsiasi combinazione di due errori in un pacchetto.



7.5.2 CYCLIC REDUNDANCY CHECK (CRC)

Il Cyclic Redundancy Check (CRC) è una tecnica di rilevamento errori largamente usata. Essa funziona come segue:

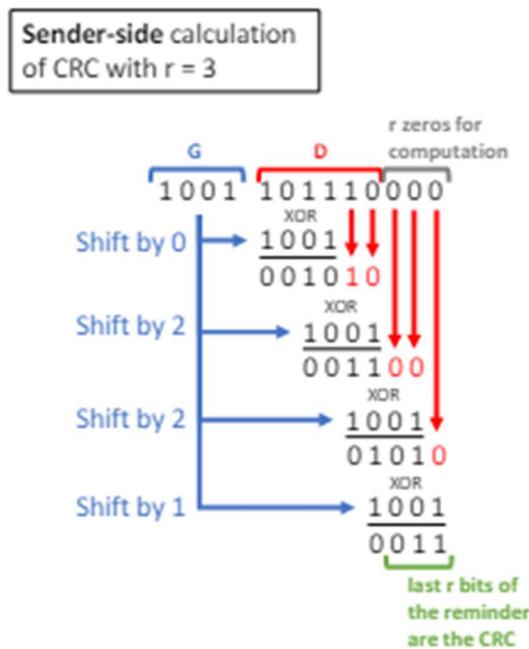
- Per inviare d bits di data D il mittente e il destinatario sono d'accordo su un pattern di $r+1$ bits chiamato generator (G), avente il bit più significativo (leftmost) ad 1.
- Per un dato pezzo di data D , il mittente sceglierà r bits addizionali (CRC bits) da essere aggiunti al messaggio così che $D+CRC$ sia il modulo 2 divisibile per G (nessun resto)
- Il destinatario divide il messaggio $D+CRC$ per G, se il resto è zero, il messaggio è corretto, altrimenti si è verificato un errore.
- Questa operazione viene implementata spostando G sopra fino al più significativo bit 1 del messaggio ed eseguendo un XOR a livello bit.

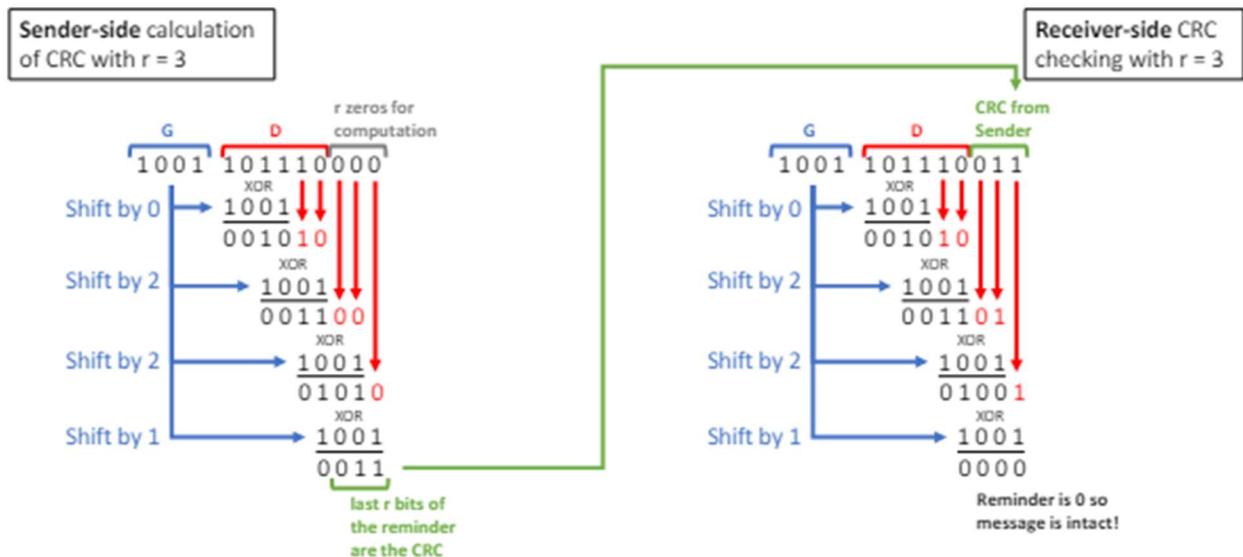
Si consideri un esempio semplice di un messaggio D a 6 bits e un CRC a 3 bits dove:

- $d = 6$
- $D = 1\ 0\ 1\ 1\ 1\ 0$
- $r = 3$
- $G = 1\ 0\ 0\ 1$

Dal lato del mittente si deve creare il CRC a partire da D e G, questo CRC è allegato al messaggio e trasmesso al destinatario.

Dal lato del destinatario si usa D, G e il CRC ricevuto per capire se il messaggio ricevuto è intatto.





Gli standardi internazionali sono stati definiti per generatori da 8, 12, 16 e 32 bits ($r=8$, $r=12$, $r=16$, $r=32$).

Il CRC-32 standard a 32 bit che è stato adottato nei protocolli IEEE in diversi livelli link usa il seguente generatore (33 bits):

$$G_{\text{CRC-32}} = 100000100110000010001110110110111$$

Ognuno degli standardi CRC può individuare con sicurezza errori di bursts minori di $r+1$ bits, mentre per errori più grandi di $r+1$ vi è una probabilità P di trovare l'errore che è:

$$P = 1 - 0.5^r$$

Quindi la probabilità di trovare l'errore aumenta con l'aumentare di r .

7.6 TIPI DI LINK

Ci sono principalmente 2 tipi di link che possono essere gestiti:

- 1. Link point-to-point: consiste in un singolo mittente ad un'estremità e in un singolo destinatario all'altra. Il protocollo point-to-point (PPP) è un esempio di come il protocollo gestisce questi links.
- 2. Link broadcast: molteplici nodi mittenti e destinatari tutti connessi allo stesso canale condiviso. Il termine broadcast è usato perché quando un nodo trasmette un frame esso viene ricevuto da tutti i nodi nel canale.

L'accesso ai link broadcast deve essere coordinato (problema degli accessi multipli) siccome comunicazioni multiple su un singolo link possono interferire l'una con l'altra.

7.6.1. COLLISIONI

Il problema principale del link broadcast è la collisione: se molteplici nodi stanno trasmettendo contemporaneamente frames sul medesimo canale, tutti questi frames si sovrapporranno diventando incomprensibili.

Questi frames in collisione vengono poi ricevuti da tutti i nodi sul canale e dismessi come errori, ma:

- Tutti i frames trasmessi sono persi
- L'intervallo di tempo è sprecato, in quanto il canale è stato usato per trasmettere dati inutili.

7.6.2. PROTOCOLLO DI ACCESSI MULTIPLI

Nelle computer networks i protocolli multiple access sono usati per regolare le trasmissioni attraverso i canali broadcast, così che:

- Le collisioni vengano gestite
- Ciascun nodo ha una chance di essere trasmesso, così che i nodi non abbiano il monopolio del link
- Connessioni stabilite non vengono interrotte

Tali protocolli sono necessari per una varietà di settaggi della network, incluse le network cablate, wireless o satellitari, dove centinaia di migliaia di nodi possono direttamente comunicare su un canale broadcast.

Il ruolo primario di un protocollo di multiple access è in qualche modo quello di evitare collisioni (vi sono dozzine di protocolli sulle differenti tecnologie a livello link). I principali approcci sono:

- Partizionamento del canale: la bandwidth è partizionata per i differenti nodi
- Random access: i nodi scommettono per l'accesso
- Taking-turns: i nodi aspettano il loro turno.

Desiderata: un protocollo ad accesso multiplo per un canale broadcast di velocità R bps dovrebbe anche fornire le seguenti caratteristiche:

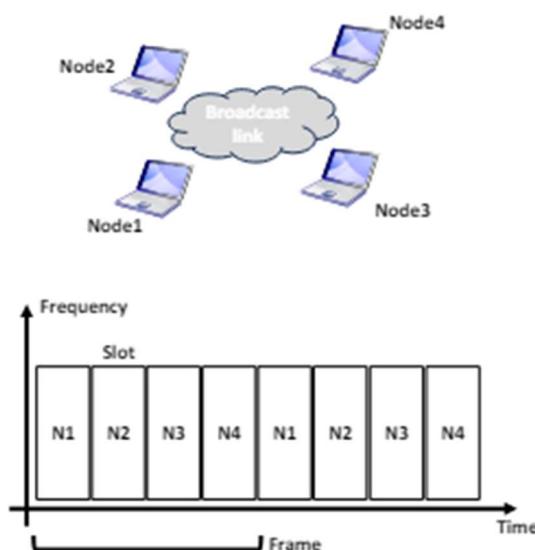
- Massimizzare l'utilizzo del canale: se M nodi hanno dati da inviare, ognuno dovrebbe avere in media un throughput di R/M bps (se $M=1$ allora il throughput dovrebbe essere R)
- Essere decentralizzato: un nodo master potrebbe essere un singolo punto di fallimento
- Essere semplice e leggero: molti frames vengono inviati, non ci deve essere alcun sovraccarico.

7.6.2.1 PROTOCOLLI PARTIZIONAMENTO CANALE: TDMA

Time division multiple access (TDMA): si consideri un canale con N nodi avente velocità di trasmissione pari a R bps, TDMA divide il tempo in time frames (steps) e in più divide ogni time frame in slots di tempo N.

Ad ogni slot di tempo viene assegnato uno degli N nodi. Ogni volta che un nodo ha un pacchetto da inviare, esso attende il time slot assegnato.

Tipicamente, le dimensioni dello slot vengono scelte di modo che un intero pacchetto possa essere trasmesso durante uno slot di tempo.

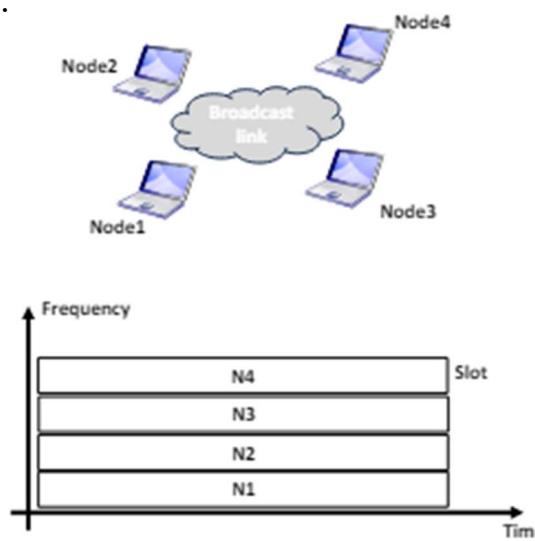


7.6.2.2. PROTOCOLLI DI PARTIZIONAMENTO CANALE: FDMA

Frequency division multiple-access (FDMA): divide il canale a velocità R bps in differenti frequenze (ciascuno con una bandwidth di R/N) e assegna a ciascuna frequenza uno degli N nodi.

FDMA e TDMA condividono pro e contro:

- Essi evitano le collisioni e dividono la bandwidth equamente tra gli N nodi.
- Un nodo è limitato ad una bandwidth di R/N , persino quando è l'unico nodo con pacchetti da inviare.

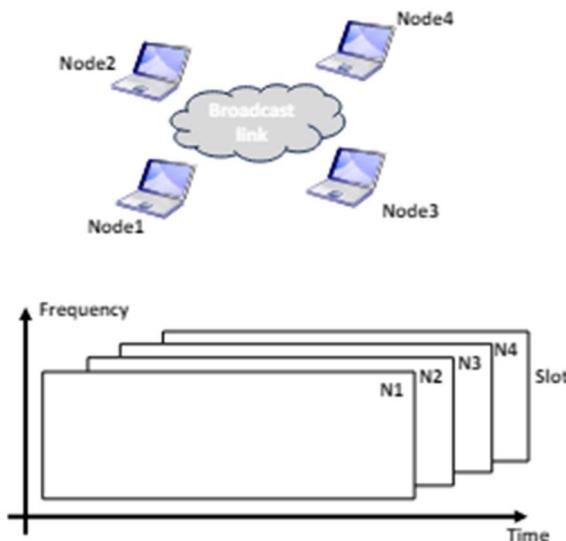


7.6.2.3 PROTOCOLLI DI PARTIZIONAMENTO CANALE: CDMA

Code divisione multiple access CDMA: assegna un differente codice ad ogni nodo che viene usato per codificare e decodificare i bits data che invia.

Se i codici vengono scelti attentamente, le networks CDMA consentono a differenti nodi di trasmettere simultaneamente senza causare interferenze.

CDMA lavora principalmente sui canali wireless; è stato usato nei sistemi militare per molto tempo (dovuto alle sue proprietà anti-jamming) e anche nella telefonia cellulare.



7.6.3. PROTOCOLLI DI RANDOM ACCESS

Nei protocolli di random access, un nodo trasmittente sempre trasmette alla massima velocità del canale (a R bps).

Se una collisione si verifica (ad esempio almeno due nodi stanno trasmettendo) tutti i nodi trasmettenti aspettano un ritardo casuale prima di tentare nuovamente la ritrasmissione.

Poiché questa selezione è eseguita in modo indipendente, è possibile per i due nodi scegliere un ritardo che è differente abbastanza da permettere a uno dei due contendenti di inviare il messaggio.

Se invece viene scelto un ritardo simile, si verifica una nuova collisione ed il processo è iterato.

7.6.3.1 PROTOCOLLI DI RANDOM ACCESS: SLOTTED ALOHA

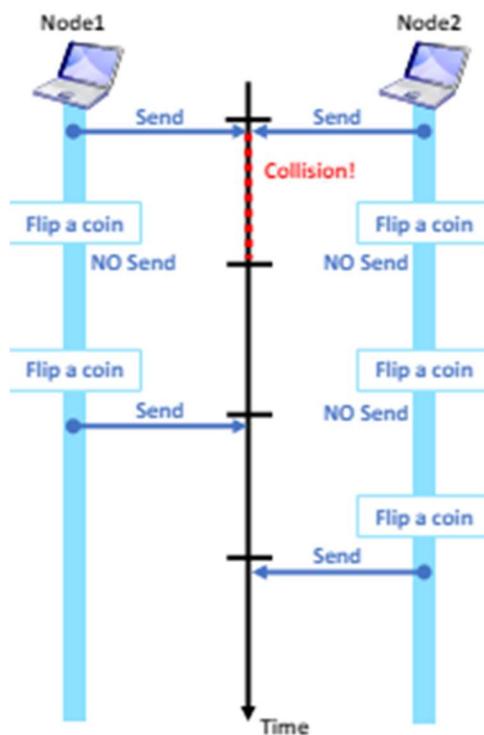
Il protocollo slotted ALOHA funziona come segue:

- Il tempo è diviso in slots (come in TDMA) dove ciascuno slot è grande abbastanza da contenere un frame.
- I nodi sono sincronizzati, così che ogni nodo trasmette frames solo all'inizio di uno slot.

- Se nessuna collisione è individuata all'interno dello slot, la comunicazione continua.
- Altrimenti, se una collisione è individuata all'interno dello slot, ogni nodo collidente ha una probabilità $p \in [0,1]$ di reinviare questo frame in ciascuno degli slots successivi, fino a quando il frame viene inviato con successo.

Caratteristiche:

- Se vi è un solo nodo, esso userà la velocità totale R del canale (nessuna collisione)
- È decentralizzata in quanto i nodi sono totalmente indipendenti dagli altri al di là della sincronizzazione.
- È semplice da implementare ed eseguire (la selezione randomica è piuttosto veloce)
- Avere multiple collisioni consecutive è improbabile.



7.6.3.2 PROTOCOLLI DI RANDOM ACCESS: CSMA

Una delle debolezze di ALOHA è che possiamo cominciare la trasmissione (producendo una collisione) anche se il canale è già occupato.

Una soluzione è quella di monitorare il canale e tentare la trasmissione solo quando il canale è inattivo.

I protocolli carrier sense multiple access (CSMA) e CSMA con collision detection (CSMA/CD) sono basati su 2 principi:

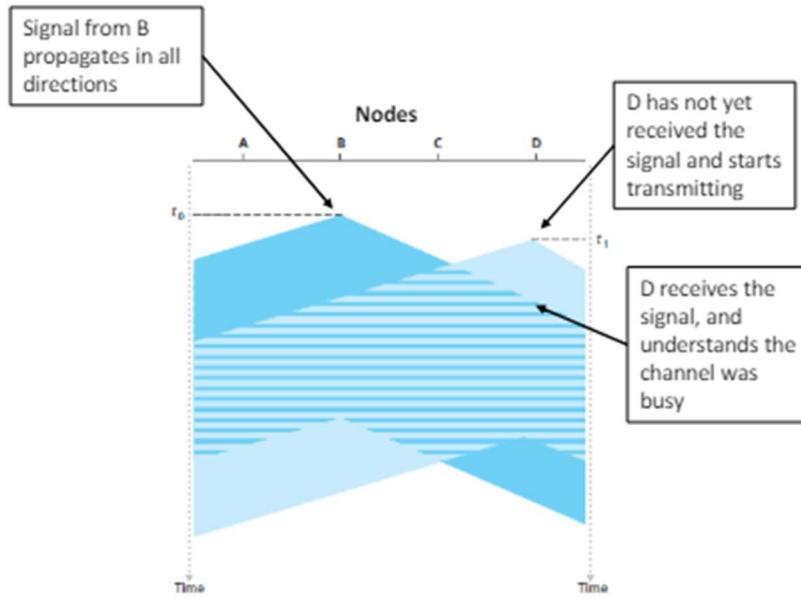
- Carrier sensing: i nodi ascoltano il canale prima di trasmettere. Se un frame proveniente da un altro nodo è correntemente in fase di trasmissione, esso aspetta fino a che nessuna trasmissione è rilevata.
- Collision detection: i nodi ascoltano il canale mentre stanno trasmettendo. Se una collisione è individuata, esso si ferma dal trasmettere e aspetta un quantità random di tempo prima di ricominciare.

Se tutti i nodi eseguono carrier sensing, perché le collisioni si verificano in primo luogo?

Questo accade a causa del ritardo nel segnale di trasmissione!

Persino se la propagazione dei segnali nel canale è tipicamente vicina alla velocità della luce, richiede tempo per raggiungere tutti gli altri nodi. Pertanto, un secondo nodo individua la trasmissione solo dopo che ha cominciato.

A causa di questo ritardo tra l'inizio della trasmissione e il rilevamento, un nodo può considerare libero un canale che in realtà è attualmente in uso, producendo una collisione.



Il CSMA puro è molto semplice:

- 1. Controlla se il canale è occupato.
- 2. Se il canale è inattivo manda un frame.

In CSMA le collisioni non vengono individuate ma sono ancora possibili, noi capiamo che un frame è andato perso soltanto poiché l'ACK non è ricevuto.

Il CSMA/CD è più evoluto (correntemente implementato in Ethernet):

- 1. Controlla se il canale è occupato

- 2. Se il canale è inattivo, invia un frame
- 3. Mentre trasmette, controlla possibili collisioni
- 4. Se una collisione viene individuata, interrompe la trasmissione e attende un periodo random $K \in \{0, \dots, 2^n - 1\}$ dove n è il numero delle collisioni rilevate sul frame corrente (backoff binario esponenziale)

Siccome il numero dei possibili periodi cresce, la probabilità di inviare con successo un frame incrementa con il numero delle collisioni.

7.6.4 TAKING-TURNS

Protocollo polling: vi è un nodo maestro che seleziona in un modo round-robin un nodo per volta a cui è concesso trasmettere (fino al massimo throughput). Questo processo è iterato ogni volta che la trasmissione si interrompe (ad esempio Bluetooth).

- Non vi sono collisioni
- Vi è un polling delay (il tempo per selezionare i nodi)
- L'approccio è centralizzato, vi è un singolo punto di fallimento

Protocollo token-passing: non vi è alcun nodo maestro, i nodi si scambiano uno speciale frame chiamato token, se un nodo riceve il token gli è concesso trasmettere, successivamente il token viene passato ad un altro nodo.

- Non vi sono collisioni
- L'approccio è decentralizzato
- Ci sono problemi se alcuni nodi dimenticano di rilasciare il token (monopolio del link)

7.6.5 INDIRIZZI MAC

A livello link, i dispositivi sono identificati mediante indirizzi MAC:

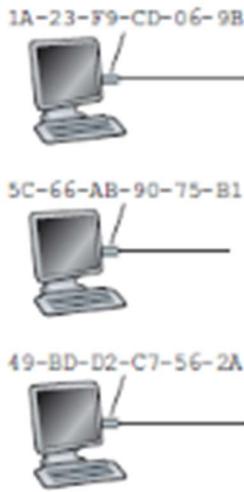
- Ogni interfaccia della network ha uno specifico indirizzo MAC (che è stato designato per essere fisso ma può essere cambiato)
- Ogni produttore ha un MAC personale.

L'indirizzo MAC (o indirizzo fisico) è un indirizzo a livello del link composto da 6 bytes (2^{48} indirizzi possibili) spesso rappresentati nella notazione esadecimale:

1A:23:F9:CD:06:9B or 1A-23-F9-CD-06-9B

Gli indirizzi MAC sono locali (mentre gli IPs sono globali):

- Tutte le interfacce sono associate ad un indirizzo MAC, ma questo è usato solo all'interno della LAN.



In una LAN, 2 interfacce (A e B) comunicano come segue:

- A include l'indirizzo MAC di B nel frame e lo trasmette.
- B riceve il frame e compara il proprio MAC con il MAC di destinazione del frame.
- Se i due MACs sono uguali, il frame viene accettato, altrimenti viene rifiutato (il resto dello stack non viene coinvolto)

Vi è anche la possibilità di inviare messaggi broadcast (che vengono accettati indipendentemente dal MAC), per le LANs che usano indirizzi a 6 byte (come ad esempio Ethernet e 802.11), l'indirizzo broadcast è una stringa di 48 1 consecutivi (che è, FF-FF-FF-FF-FF-FF nella notazione esadecimale).

In una LAN è piuttosto possibile (anche se non frequente) per un'interfaccia ricevere frames diretti ad un'altra interfaccia, il ruolo dell'indirizzo MAC è quello di filtrare i frame non desiderati senza disturbare l'host.

7.6.6 SWITCHES

Gli switches sono l'equivalente dei routers nel livello link:

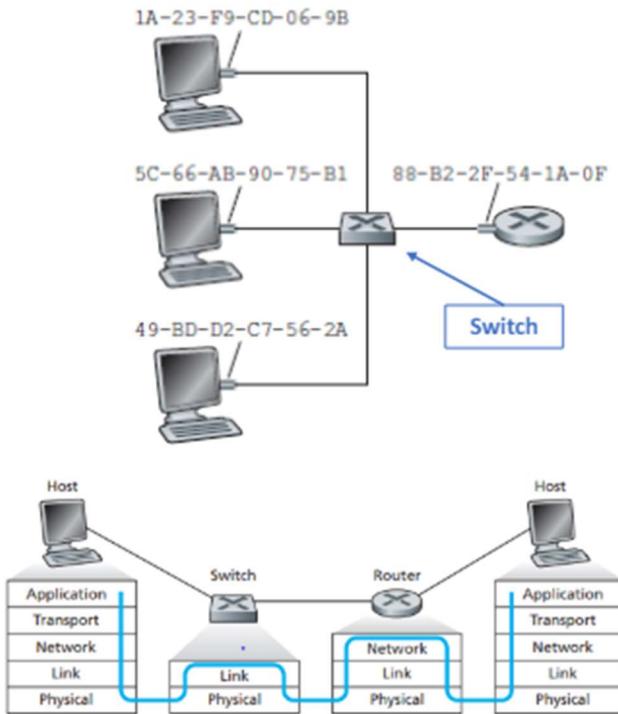
- Non vi è alcun algoritmo di routing implementato
- Vengono usati solo gli indirizzi MAC, gli indirizzi IP non sono considerati

Il ruolo dello switch è quello di ricevere i frames del livello link in ingresso e inoltrarli nei link di uscita:

- Lo switch è trasparente agli hosts e ai routers nella sottorete
- Uno switch ha anche dei buffers sulle interfacce

Gli switches hanno delle forwarding tables che associano gli indirizzi MAC alle interfacce.

- La tabella è aggiornata automaticamente e dinamicamente (self-learning) quando nuovi dispositivi vengono individuati.



SWITCHED LAN

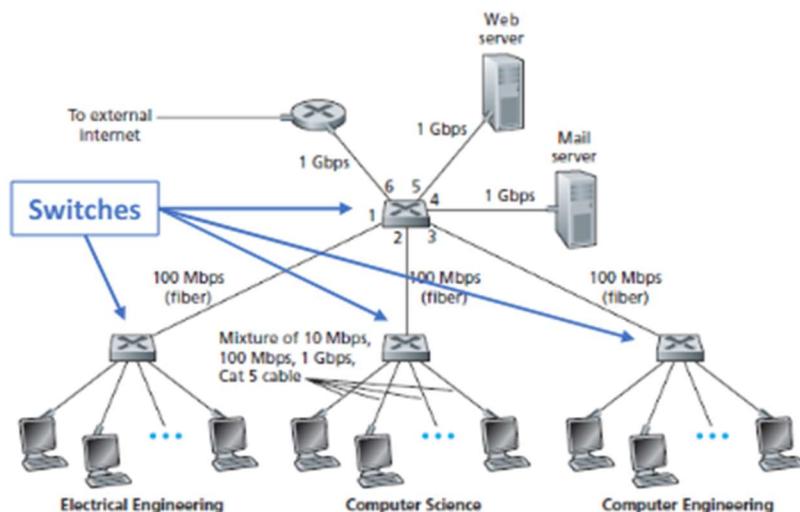
È tipico per le LANs utilizzare uno o più switches connettendo multipli dispositivi locali.

A differenza dei routers, gli switches sono più veloci e plug-and-play:

- Non vi è alcun algoritmo di routing coinvolto
- Solo due livelli dello stack vengono considerati

D'altro canto, le switched LANs sono limitate in dimensione e devono essere strutturate a forma d'albero:

- Gli indirizzi MAC sono difficili da raggruppare (le forwarding tables negli switches potrebbero crescere rapidamente)
- Non vi è alcun instradamento, i loops sono difficili da evitare



7.6.7 ADDRESS RESOLUTION PROTOCOL (ARP)

Siccome i protocolli a livello superiore lavorano con gli indirizzi IP è necessario tradurre gli IP in MAC.

L'Address Resolution Protocol (ARP) gestisce la conversione degli indirizzi IP in MAC.

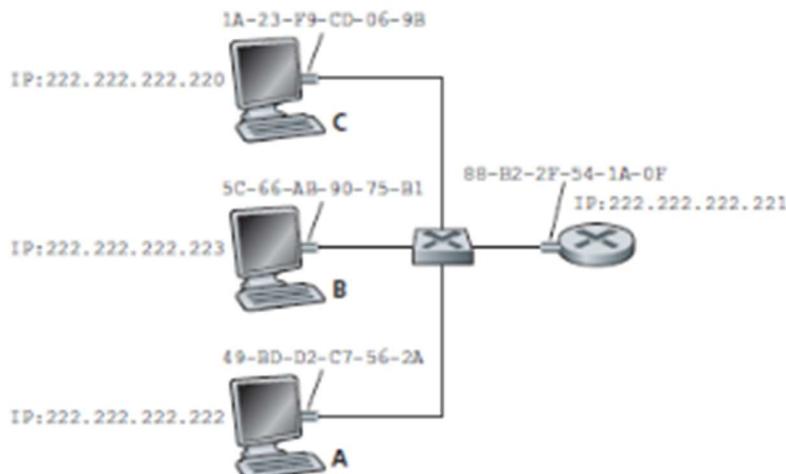
Ogni interfaccia è dotata di un modulo ARP avente un ARP table che associa ciascun IP nella LAN ad un indirizzo MAC con uno specifico time to live (TTL) valore dopo il quale la entry è ritardata (in genere 20 minuti).

Siccome gli indirizzi MAC sono locali, ARP lavora solo su reti locali (LANs).

Si assume che un host C (222.222.222.220) voglia mandare messaggi all'host A (222.222.222.222). Per fare ciò, è necessario conoscere anche il MAC associato.

Prima di inviare il messaggio, se A non è presente nella tabella un pacchetto ARP è inviato in broadcast a tutti i dispositivi della network in ricerca del giusto IP.

Tutti i nodi ricevono questo pacchetto ma solo l'IP ricercato (222.222.222.222) risponde con un messaggio diretto (no broadcast).

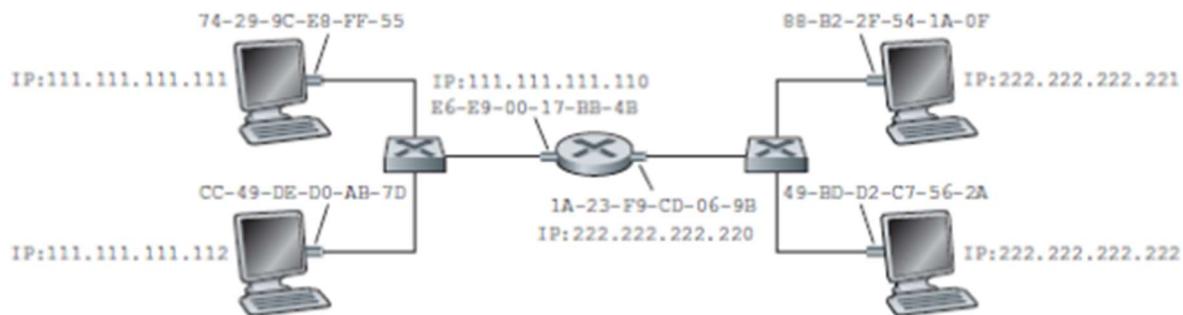


IP Address	MAC Address	TTL
222.222.222.221	BB-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-A8-90-75-81	13:52:00

Cosa succede se l'IP è al di fuori della network (non locale)?

Il router che connette le 2 network deve avere almeno 2 interfacce (2 IP, 2 MAC e 2 ARP tables) ciascuna all'interno delle specifiche sottoreti.

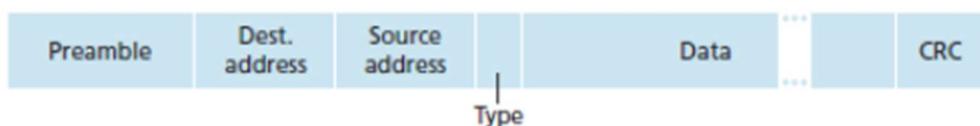
I frames diretti al di fuori della sottorete sono inviati alla prima interfaccia del router, inviati alla seconda interfaccia e diretti verso l'host corretto utilizzando la seconda ARPA table.



7.6.8 ETHERNET FRAME

L'Ethernet frame è composto dai seguenti campi:

- Data field (da 46 a 1500 bytes): contiene il datagramma IP. Il limite massimo è dato dalla maximum transmission unit (MTU) dell'Ethernet, se il datagramma eccede questa dimensione viene frammentato
- Destination address (6 bytes): contiene l'indirizzo MAC dell'adattatore di destinazione
- Source address (6 bytes): contiene l'indirizzo MAC dell'adattatore sorgente che trasmette il frame sulla LAN
- Type field (2 bytes): specifica il protocollo a livello di network usato per questo frame
- CRC (4 bytes): contiene il numero CRC
- Preamble (8 bytes): è un blocco di bits “wake up” utilizzato per sincronizzare gli orologi degli adattatori di sorgente e destinazione.



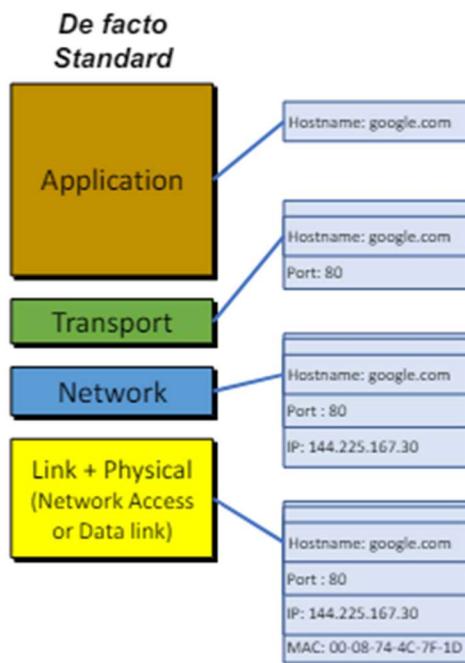
8 STACK OVERVIEW

8.1 THERE AND BACK AGAIN

Come abbiamo già visto, diversi protocolli nello stack TCP/IP collaborano nel permettere ai messaggi di viaggiare avanti ed indietro nella network.

Il protocollo specifico di ogni livello aggiunge un pezzo di informazione ai messaggi (attraverso l'incapsulamento e il decapsulamento) regolando uno o più aspetti della comunicazione (servizi offerti).

È in qualche modo difficile estrarre il quadro completo dai singoli protocolli, adesso ripasseremo l'intero processo proponendo lo scenario di richiesta di una pagina web.



Example of the different addresses/identifiers considered in each layer of the stack.

8.2 ESEMPIO DELLA PAGINA WEB

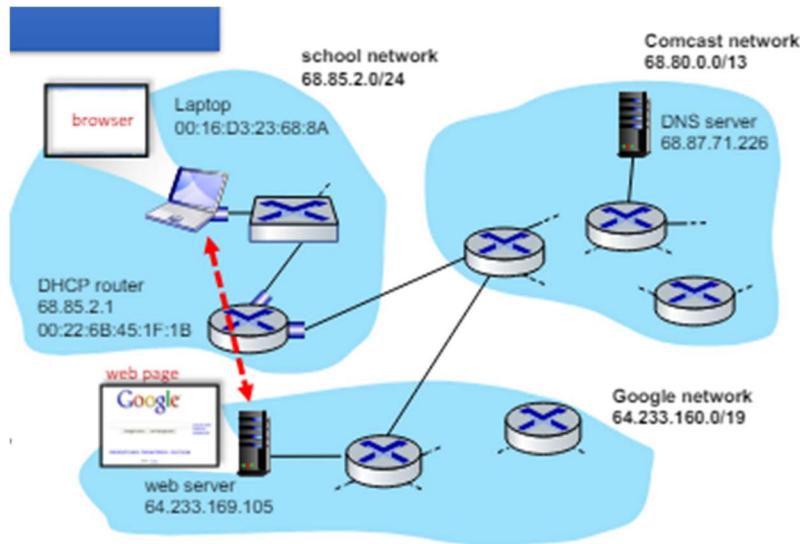
Assumiamo che un utente (Bob) voglia accedere alla pagina web google.com dal network istituzionale (scuola).

- Per fare ciò, Bob connette il laptop al network della scuola attraverso un cavo Ethernet

In questo esempio assumiamo una configurazione tipica per la network della scuola:

- Le sockets (sui muri) dell'Ethernet sono connesse ad uno switch, che è connesso al router abilitato DHCP della scuola.
- Il router della scuola è connesso ad un ISP che fornisce anche un servizio DNS.

Per semplicità, assumiamo anche che non vi è servizio NAT dal router, che tutte le connessioni sono Ethernet e i che pacchetti non vengono mai persi.



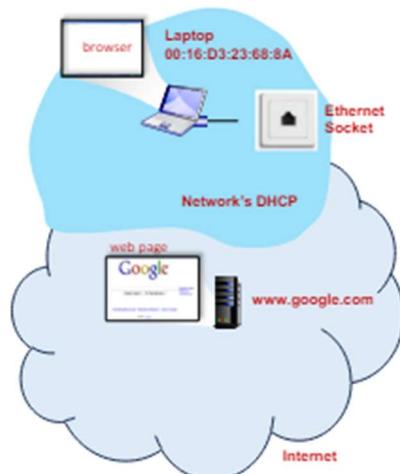
All'inizio, l'utente sa molto poco circa la topologia della network, il modo in cui la LAN è connessa ad Internet o i dispositivi coinvolti.

Ciò che Bob sa è:

- C'è un sito web www.google.com da qualche parte in Internet che Bob vuole raggiungere.
- La rete locale (LAN) è in qualche modo connessa ad Internet
- Il computer può essere connesso alla LAN attraverso un cavo Ethernet (vi è una socket sul muro)
- La LAN ha un DHCP attivo.

Una volta che il laptop è fisicamente connesso alla rete (attraverso il cavo Ethernet) esso deve:

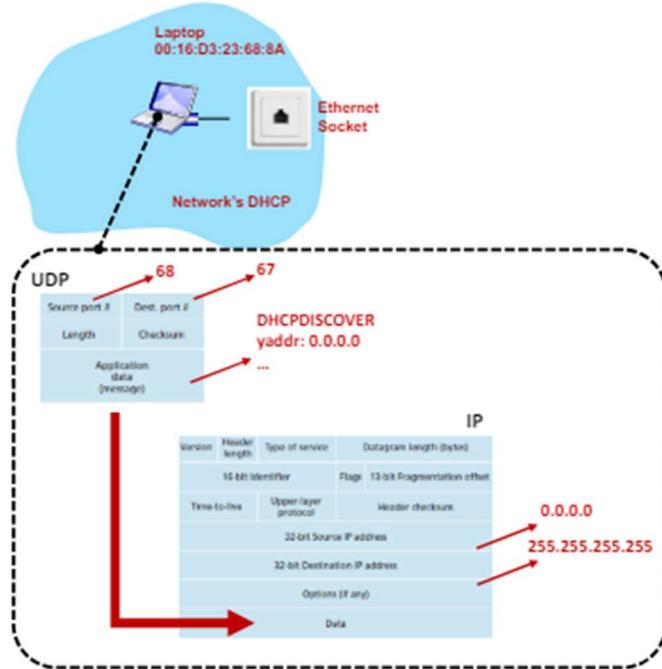
- Unirsi alla network (prendere informazioni network dal DHCP)
- Prendere l'indirizzo IP del sito web (DNS)
- Prendere la pagina web di Google (http://www.google.com).



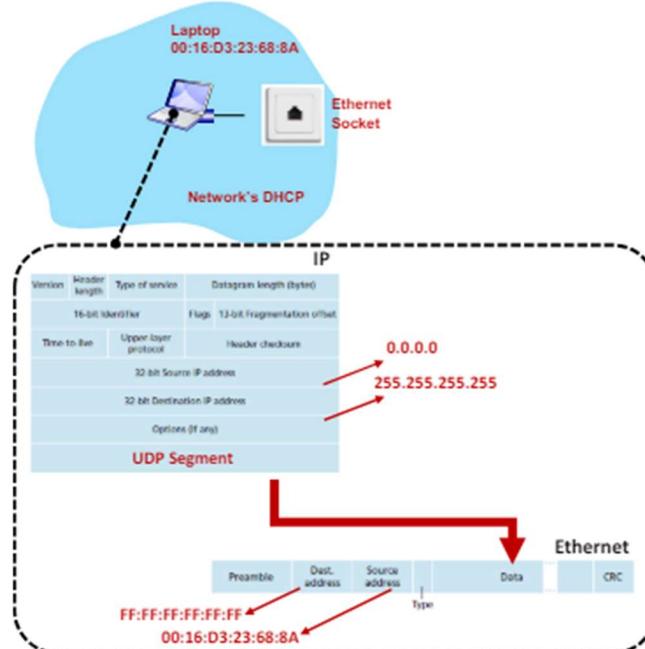
8.2.1 DHCP

Il primo step per il laptop è quello di unirsi alla network richiedendo un host IP, un gateway e DNS al DHCP.

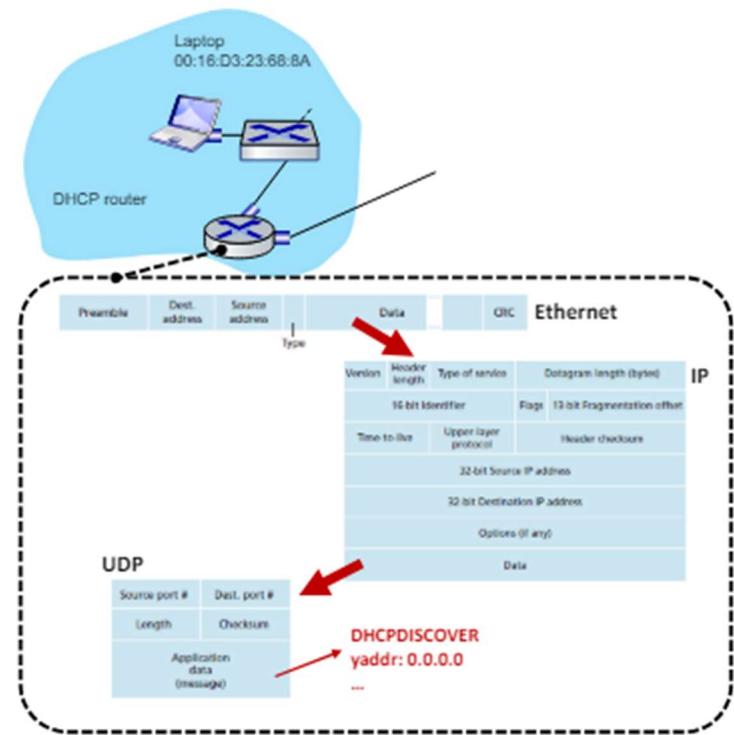
- 1. Il sistema operativo (OS) sul laptop crea un messaggio discovery DHCP e inserisce questo messaggio all'interno di un segmento UDP con porta di destinazione 67 (server DHCP) e porta di sorgente 68 (client DHCP).
 - o Il segmento UDP viene poi piazzato all'interno di un datagramma IP con indirizzo IP broadcast di destinazione (255.255.255.255) e un indirizzo di sorgente IP di 0.0.0.0 (non vi è ancora un IP host)



- 2. Il datagramma IP contenente il messaggio discovery DHCP viene poi piazzato all'interno del frame di Ethernet avente FF:FF:FF:FF:FF:FF (broadcast) come indirizzi MAC di destinazione (raggiungerà tutti i dispositivi sullo switch e si spera il DHCP) e l'indirizzo MAC del laptop 00:16:D3:23:68:8A come sorgente.

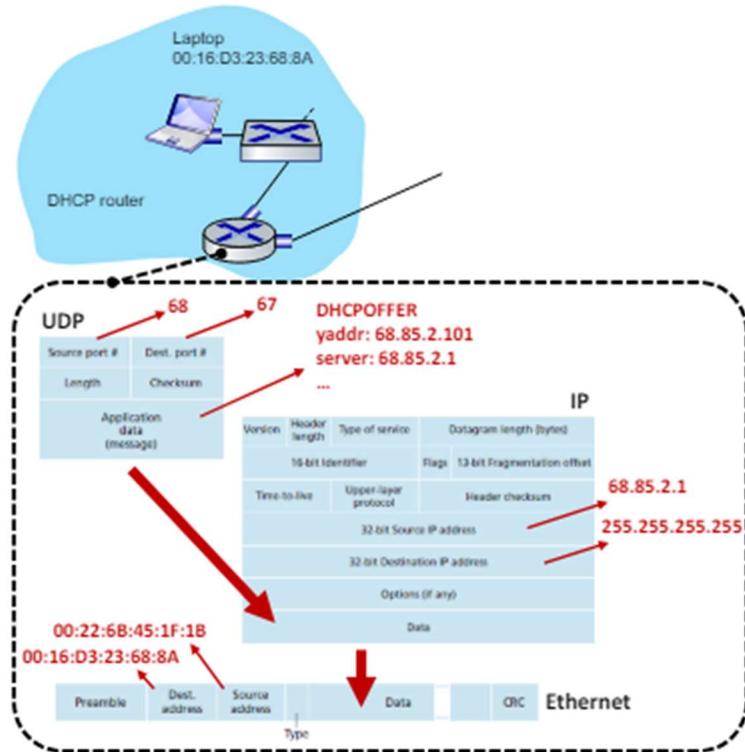


- 3. Il frame di Ethernet contenente il messaggio DHCP viene inviato allo switch di Ethernet. Lo switch trasmette il frame su tutte le porte in uscita (incluse le porte connesse al router).
- 4. Il router riceve il frame di Ethernet contenente il discovery DHCP sulla sua interfaccia (con indirizzo MAC 00:22:6B:45:1F:1B), il messaggio viene decapsulato:
 - o Il frame viene accettato perché ha come indirizzo di destinazione MAC quello broadcast (FF:FF:FF:FF:FF:FF), poi il datagramma IP viene estratto.
 - o Il datagramma viene accettato perché ha come indirizzo di destinazione IP quello broadcast (255.255.255.255), poi il segmento UDP viene estratto.
 - o Il segmento UDP viene sottoposto a demultiplexed e il payload viene ricevuto attraverso il processo DHCP sul router.



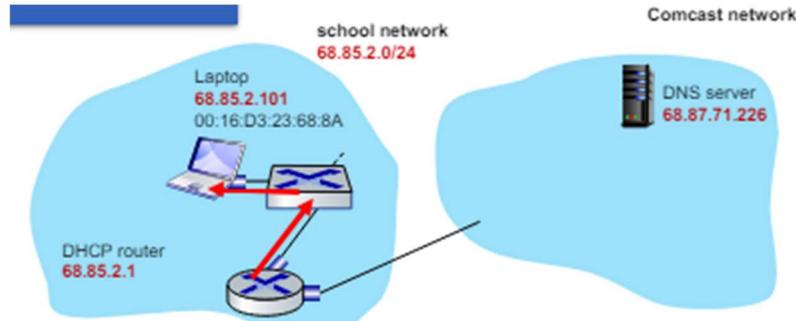
- 5. Assumiamo che il router DHCP possa allocare indirizzi IP nel blocco CIDR 68.85.2.0/24 (che è una sottorete fornita da ISP):
 - o Il server DHCP offre un indirizzo IP 68.85.2.101 al laptop
 - o Il server DHCP crea un messaggio di offerta DHCP contenente:
 - L'indirizzo IP offerto e la maschera (68.85.2.101/24)
 - L'indirizzo IP del server DNS (68.87.71.226)
 - L'indirizzo IP del gateway (68.85.2.1)

- Il messaggio viene incapsulato:
 - Un segmento UDP viene creato con porta di sorgente 67 e porta di destinazione 68.
 - Il segmento viene inserito dentro ad un datagramma IP avente broadcast come indirizzo di destinazione e 68.85.2.1 come indirizzo di sorgente
 - Il segmento viene inserito in un frame di Ethernet avente 00:22:6B:45:1F:1B come indirizzo MAC di sorgente (l'interfaccia LAN del router) e 00:16:D3:23:68:8A come indirizzo MAC di destinazione (il laptop).



- 6. Il frame Ethernet contenente l'offerta di DHCP viene inviato (unicast) dal router allo switch, che lo inoltra al laptop controllando l'indirizzo MAC di destinazione.
- 7. Il laptop di Bob riceve il frame di Ethernet con l'offerta DHCP e lo decapsula:
 - Il frame viene accettato dovuto al corretto indirizzo MAC, poi il datagramma viene estratto
 - Il datagramma IP viene accettato a causa dell'indirizzo IP di trasmissione, poi il segmento UDP viene estratto.
 - L'offerta di DHCP viene sottoposta a demultiplexing e passata al processo client DHCP del sistema operativo.
 - Il sistema operativo accetta l'offerta ed invia indietro un messaggio di richiesta DHCP (saltato per la brevità)

- Quando un messaggio DHCP ACK viene eventualmente ricevuto da un nuovo segmento UDP, il sistema operativo setta l'informazione network ricevuta (IP, gateway e DNS). Adesso il laptop si è unito alla network.

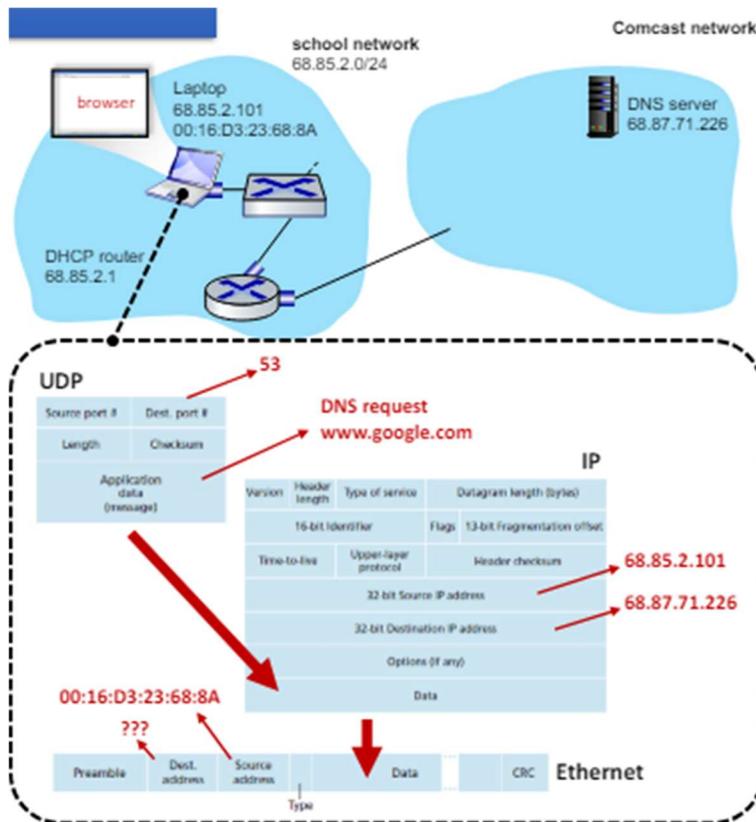


- All'inizio noi conoscevamo soltanto l'indirizzo MAC del laptop, adesso noi conosciamo:
 - L'IP del laptop
 - L'IP del router (gateway)
 - La configurazione network (maschera)
 - L'IP del DNS

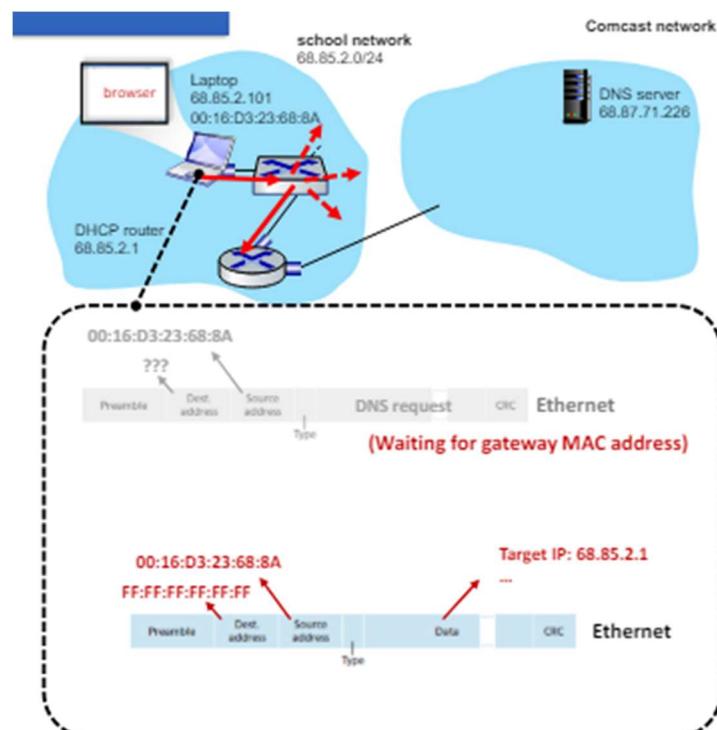
8.2.2 ARP

Adesso l'utente decide di aprire il browser e di navigare l'homepage di Google (www.google.com) , quindi l'IP del web server deve essere recuperato dal DNS:

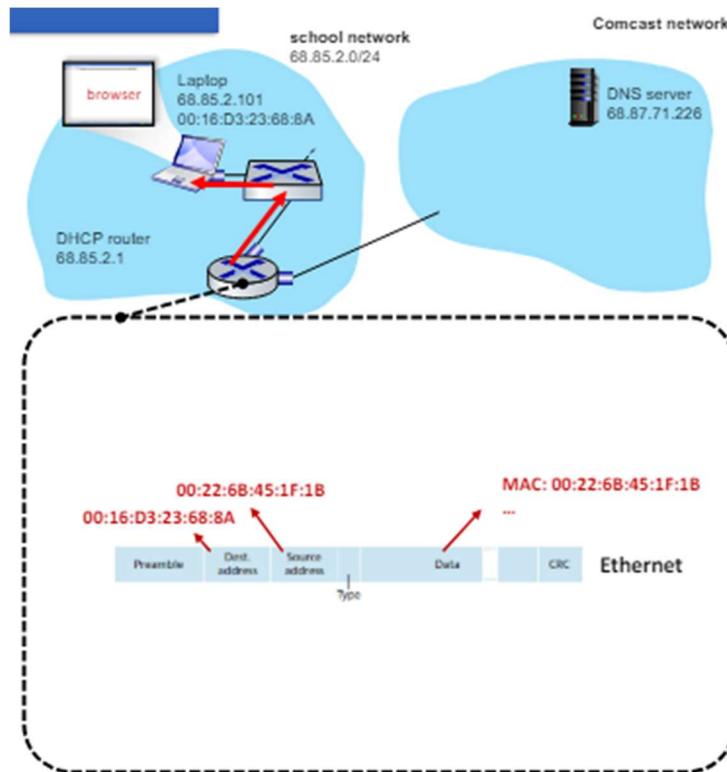
- 8. Il browser invoca una routine del sistema operativo per ottenere l'IP del server (come nella funzione `gethostbyname`):
 - Il sistema operativo crea un messaggio di query DNS per l'hostname www.google.com
 - Il messaggio DNS viene incapsulato all'interno di un segmento UDP avente 53 come porta di destinazione (server DNS)
 - Il segmento UDP è posto all'interno di un datagramma IP avente 68.87.71.226 (IP di DNS recuperato da DHCP) come indirizzo di destinazione e 68.85.2.101 come indirizzo IP di sorgente (sé stesso)
- 9. Il datagramma IP contenente la query di DNS deve essere incapsulata nel frame di Ethernet. Per fare ciò, abbiamo bisogno dell'indirizzo MAC del gateway, quindi la query ARP deve essere creata.
 - Nota che persino se noi conoscessimo l'IP del gateway da DHCP, non conosceremmo l'indirizzo MAC.



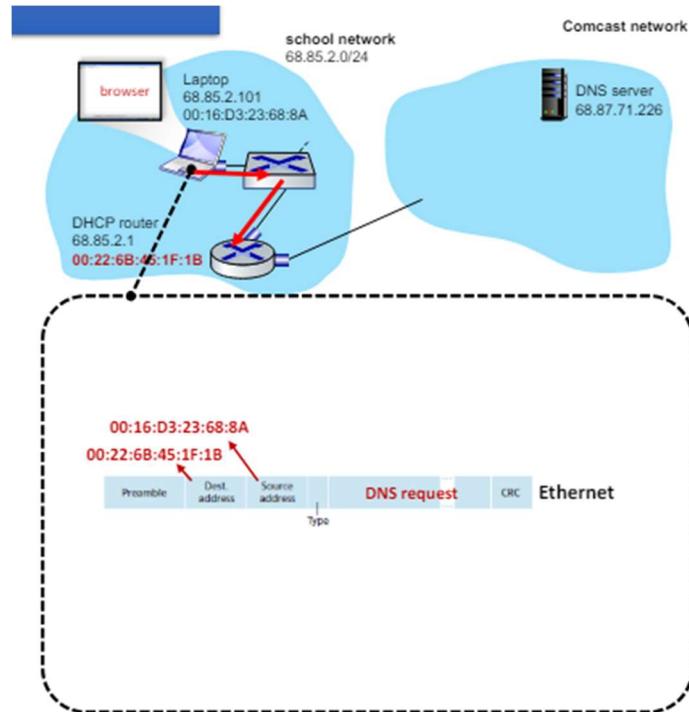
- 10. Il sistema operativo crea un frame query ARP avente un indirizzo target IP 68.85.2.1 (il gateway di default) e broadcast (FF:FF:FF:FF:FF) come indirizzo MAC di destinazione
 - o Il frame viene inviato allo switch, che lo consegna a tutti i dispositivi connessi incluso il gateway del router



- 11. Il router riceve il frame sull'interfaccia dal lato della LAN e scopre che l'indirizzo target IP 68.85.2.1 nel messaggio ARP coincide con l'indirizzo IP dell'interfaccia:
 - o Il gateway del router prepara una risposta ARP, indicando che l'indirizzo MAC 00:22:6B:45:1F:1B corrisponde all'indirizzo IP 68.85.2.1.
 - o Il messaggio di risposta ARP è inserito in un frame di Ethernet avente 00:16:D3:23:68:8A come indirizzo di destinazione (indirizzo del laptop)
 - o Il frame viene inviato allo switch, che lo consegna al laptop.

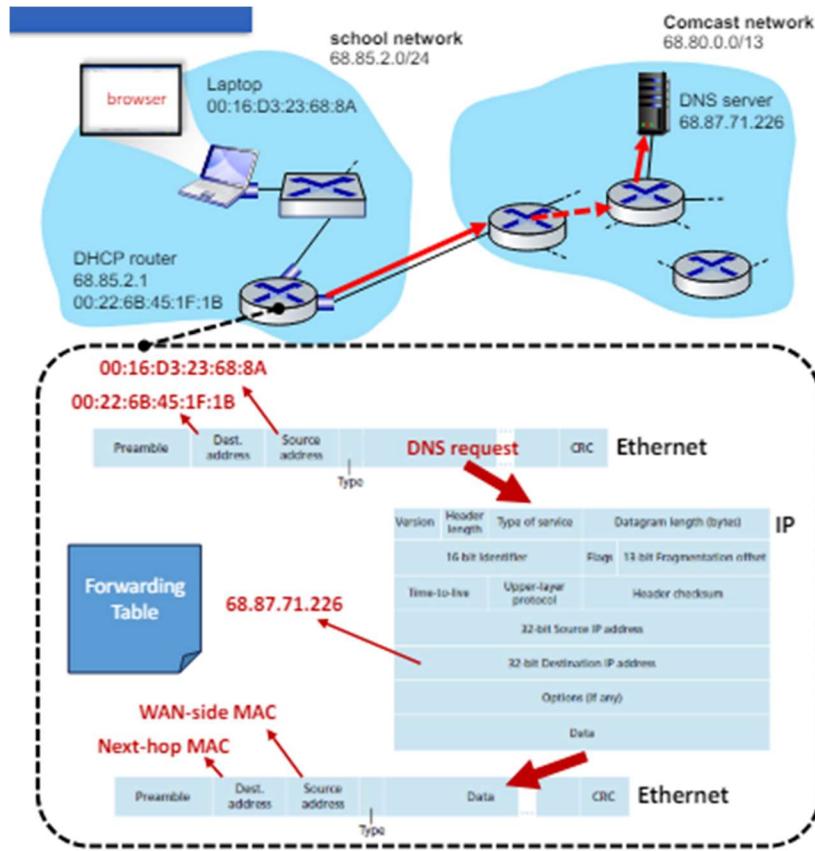


- 12. Il laptop riceve il frame contenente il messaggio di risposta ARP ed estrae l'indirizzo MAC del router gateway (00:22:6B:45:1F:1B)
- 13. Il laptop adesso può inviare il frame di Ethernet contenente la query DNS all'indirizzo MAC del gateway.
 - o Nota che in questo caso il datagramma IP avrà un indirizzo IP di destinazione 68.87.71.226 (il server DNS) e un indirizzo di destinazione MAC 00:22:6B:45:1F:1B (il router gateway).

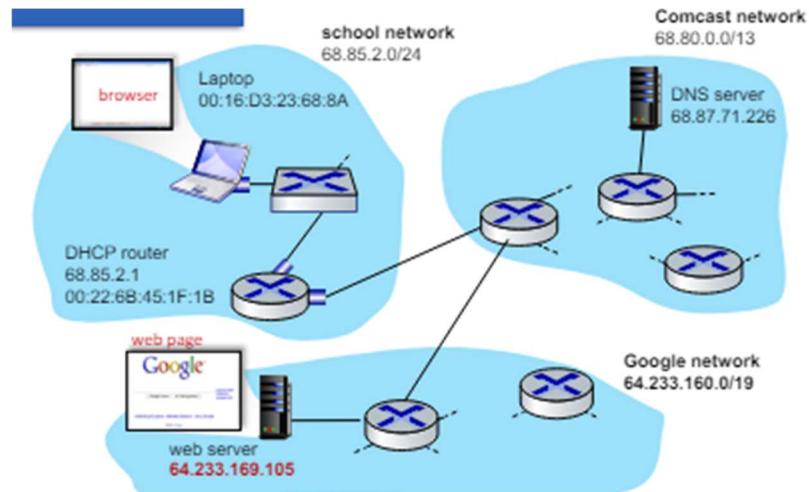


8.2.3 DNS

- 14. Il router gateway riceve il frame di Ethernet e lo decapsula:
 - o Il router estrae il datagramma IP dal frame e recupera l'IP di destinazione del DNS
 - o Il router controlla l'indirizzo IP di destinazione (68.87.71.226) e determina dalla sua forwarding table che il datagramma dovrebbe essere inviato al primo router della network ISP
 - o Il datagramma IP viene posto all'interno di un frame a livello link appropriato per il link che connette il router della scuola al primo router della ISP network e il frame viene inviato su questo link.
- 15. Il primo router dell'ISP riceve il frame ed estrae il datagramma IP:
 - o Il router controlla l'indirizzo di destinazione (68.87.71.226) e recupera dalla forwarding table (creata attraverso un algoritmo link-state o distance-vector) l'interfaccia d'uscita.
 - o Un nuovo frame viene poi creato ed inviato al prossimo router attraverso l'interfaccia selezionata
 - o Questo processo potrebbe essere iterato diverse volte prima che il server DNS venga raggiunto.



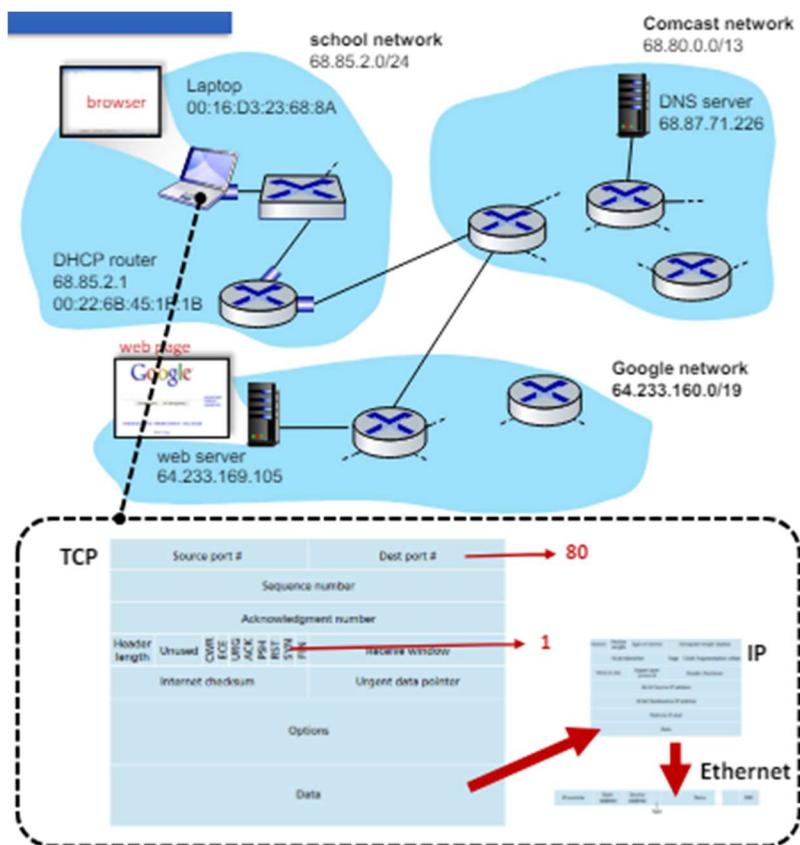
- 16. Eventualmente il datagramma IP contenente la query DNS arriva al server DNS:
 - o Il server DNS estrae il messaggio query DNS, controlla il nome www.google.com nel suo database ed individua l'indirizzo IP 64.233.169.105 (il server di Google)
 - Nota che qui stiamo assumendo un IP proveniente da una cache, altrimenti ulteriori richieste ad authoritative servers dovrebbero essere inviate.
 - o Il server DNS crea un messaggio di risposta DNS contenente l'indirizzo IP recuperato, e lo piazza in un segmento UDP.
 - o Il segmento viene posto in un datagramma IP avente come IP di destinazione 68.85.2.101 (il laptop) e poi nel frame appropriato.
 - o Questo messaggio verrà poi inviato indietro attraverso la network dell'ISP al router della scuola e da lì attraverso lo switch al laptop.
- 17. Il sistema operativo del laptop estrae l'indirizzo IP target dal messaggio DNS. Adesso noi sappiamo come raggiungere il web server di Google.



8.2.4 HTTP

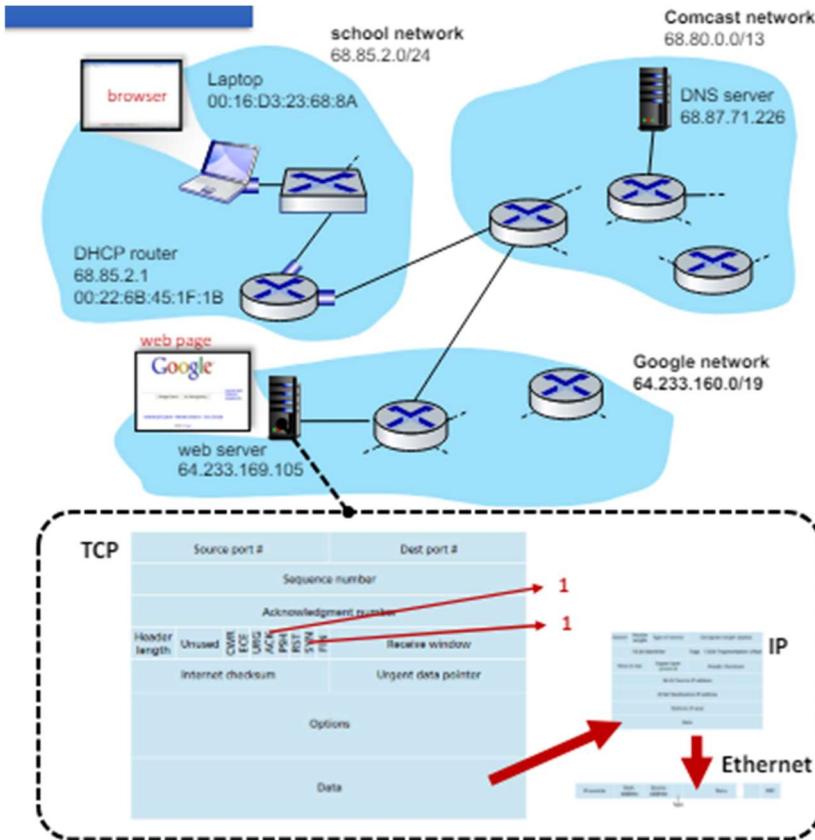
Conoscendo l'IP del server possiamo ora creare una richiesta GET HTTP per l'home page di Google:

- 18. Il browser crea una socket TCP, poi il sistema operativo esegue un handshake a tre vie con il web server di Google:
 - o Il sistema operativo crea un segmento TCP SYN con porta di destinazione 80 (http)
 - o Il segmento viene incapsulato all'interno di un datagramma IP avente come indirizzo di destinazione IP 64.233.169.105 (www.google.com)
 - o Il datagramma viene inserito all'interno di un frame con indirizzo MAC di destinazione 00:22:6B:45:1F:1B (il router gateway) e lo invia allo switch.

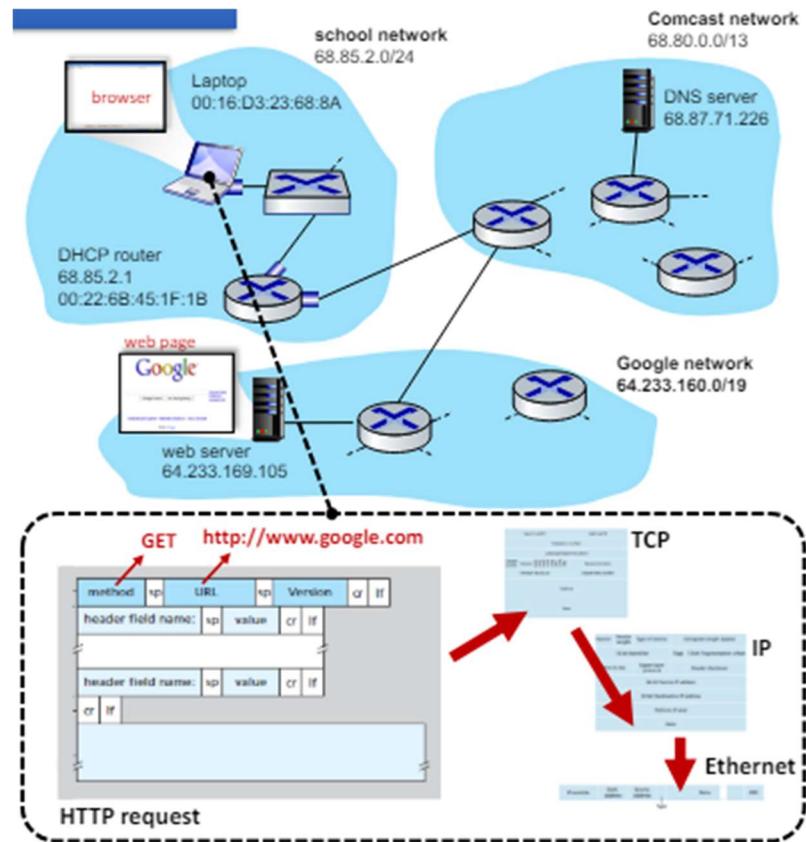


- 19. Tutti i routers (locali, ISP e networks di Google) inoltrano il datagramma al web server usando le loro forwarding tables.
 - o Nota che i frames possono essere modificati lungo il percorso a seconda dei link specifici.
- 20. Il frame contenente il segmento SYN eventualmente arriva al server:
 - o Il datagramma viene estratto dal frame
 - o Il segmento viene estratto dal datagramma, sottoposto a demultiplexing e passato alla socket di benvenuto associata alla porta 80
 - o Una socket connection-specific viene creata tra il web server di Google e il laptop

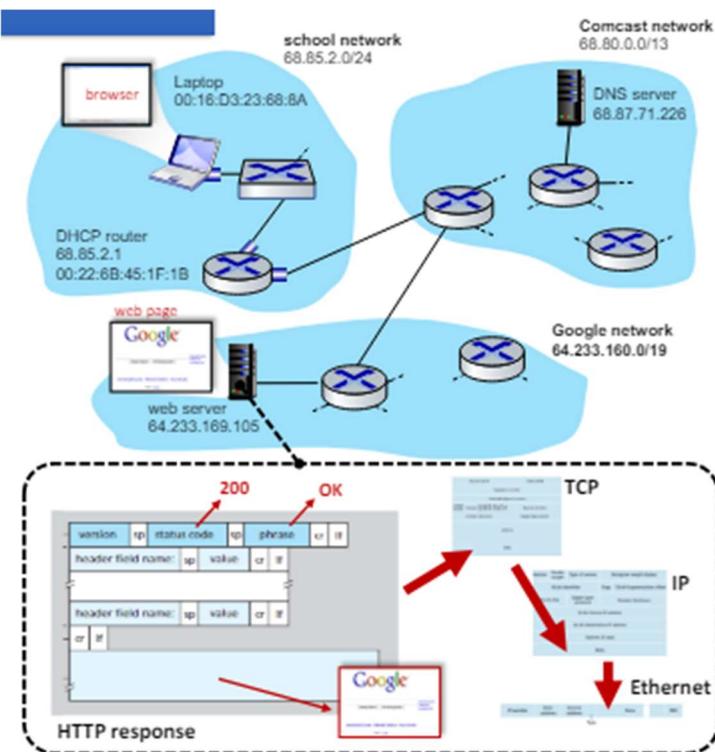
- Un segmento TCP SYNACK viene generato, posto all'interno di un datagramma avente indirizzo di destinazione 68.85.2.101 (laptop)
- Il segmento viene posto in un frame a livello link appropriato per viaggiare attraverso il link che collega www.google.com al suo first-hop router.



- 21. Il TCP SYNACK eventualmente arriva al laptop:
 - Il datagramma è sottoposto a demultiplexing e passato dal sistema operativo alla socket TCP creata nel passo 18
 - La socket entra nello stato “connection established”
- 22. La socket sul laptop adesso è pronta ad inviare bytes al web server:
 - Il browser crea il messaggio GET HTTP contenente l’URL che deve essere recuperato
 - Il messaggio viene scritto nella socket, e la richiesta GET diventa il payload del segmento TCP
 - Il segmento TCP viene piazzato in un datagramma e poi in un frame (incapsulamento)
 - Il messaggio viene consegnato a www.google.com (come negli steps 18-20)



- 23. Il web server di www.google.com riceve il messaggio GET dalla socket TCP:
 - o Il messaggio viene decapsulato e sottoposto a demultiplexing, poi la richiesta GET viene interpretata
 - o Il server crea un messaggio di risposta HTTP avente l'home page di Google nel body
 - o Il messaggio viene scritto nella socket TCP
 - o Il sistema operativo del server incapsula il messaggio in un segmento, in un datagramma e alla fine in un frame



- 24. Il datagramma contenente il messaggio di risposta HTTP viene inoltrato attraverso le 3 networks ed arriva al laptop:
 - o Il sistema operativo del laptop decapsula il messaggio, che viene sottoposto a demultiplexing e passato al browser
 - o Il browser legge la risposta HTTP dalla socket
 - o Il browser estrae il codice html dal body della risposta HTTP
 - o Il browser infine mostra la web page!

9 NETWORK SECURITY

9.1 INTRODUZIONE

La network security è il campo che studia i possibili attacchi alle networks e i possibili modi per prevenirli.

Il networking è parte delle nostre vite (la connessione internet viene considerata quasi come acqua o fornitura di energia) e una grande quantità dei nostri dati sensibili viaggia attraverso le networks.

Ci sono differenti tipi di attacchi che hanno differenti scopi e meccanismi.

- Gli attacchi evolvono assieme alla tecnologia e all'audience delle networks.

9.2 ATTACCHI: MALWARES

Un malware è un software malevolo che può essere trasferito ad un computer attraverso la network (ad esempio file scaricati, allegati alle mail ecc ecc)

Una volta che il malware infetta il nostro dispositivo può danneggiarlo in diversi modi:

- Forzando il sistema a mostrare annunci commerciali (adware)
- Mostrando messaggi di falso allarme per indurre gli utenti a scaricare malwares (scareware)
- Cancellando (wiper) o crittografando (ransomware) i nostri file
- Raccogliendo informazioni private come passwords, numeri di sicurezza ecc (spyware)
- Ottenendo privilegi root del nostro sistema (rootkits)
- Rende un dispositivo uno schiavo o un punto di appiglio per attaccare altri dispositivi (zombie o botnet)

Oggi i malware sono spesso autoreplicanti: una volta che esso infetta un host, da quell'host cerca di entrare in un altro host tramite internet, e dagli host recentemente infettati cerca di infettarne ancora altri e così via.

Il malware può diffondersi sottoforma di virus o worm:

- I virus sono malware che richiedono una qualche forma di interazione con l'utente per infettare il dispositivo dell'utente
 - o Sono tipicamente camuffati come componenti di software legittimi (trojan horses)
- I worms sono malware che possono entrare in un dispositivo senza un'esplicita interazione con l'utente.
 - o Per esempio, un utente potrebbe star eseguendo un'applicazione network vulnerabile entro cui accetta il worm senza alcun intervento. La worm poi scansiona la network per hosts che eseguono una simile applicazione.

9.2.1 ATTACCHI: DoS

Denial-of-Service (DoS) sono attacchi piuttosto comuni e vengono realizzati per rendere una network, un host o un altro pezzo dell'infrastruttura inutilizzabili dagli utenti legittimi.

Vi sono 3 tipi di attacchi DoS:

- 1. Vulnerability attack: inviando messaggi adatti ad applicazioni o sistemi operativi vulnerabili per far sì che essi si interrompano o crashino.
- 2. Bandwidth flooding: inviando una grande quantità di pacchetti a degli host bersaglio, impedendo ai pacchetti legittimi di raggiungere il server.
- 3. Connection flooding: stabilendo un grande numero di connessioni TCP half-open o fully open con l'host target, così che esso smetta di accettare connessioni legittime.



Example of a **DDoS** (Distributed DoS) attack using a **botnet** of zombies (or slaves).

9.2.2 ATTACCHI: PACKETS SNIFFING

Il packets sniffing coinvolge un ricevitore passivo (sniffer) che salva una copia di pacchetti rilevanti da un host target cercando di rubare informazioni sensibili (eavesdropping).

- Gli sniffers possono essere dispiegati in ogni tipo di network broadcast (cablate o wireless) semplicemente copiando pacchetti che invece di essere scartati vengono indirizzati a destinazioni differenti.
- Per le networks non broadcast, uno sniffer può essere posto all'interno di un malware (spyware) e usato per infettare dispositivi della network (ad esempio routers) così che tutto il traffico inoltrato possa essere a sua volta copiato

Poiché gli sniffers sono passivi (nessun traffico addizionale viene iniettato all'interno della network) sono molto difficili da individuare.

Per prevenire lo sniffing possono essere utilizzati approcci di crittografia.

Ci sono differenti sniffers gratuitamente disponibili in internet. Un notevole esempio di pacchetti sniffer è Wireshark (ex Ethereal).

Wireshark è un analizzatore di pacchetti o protocolli utilizzato principalmente per scopi legittimi (soluzione dei problemi, controllo della network, creazione di nuovi protocolli ecc).

Wireshark è disponibile in diversi sistemi operativi (incluso Linux).

9.2.3 ATTACCHI: IP SPOOFING

L'IP spoofing è una tecnica che permette agli host malevoli di inserire nella network pacchetti con falsi indirizzi di sorgente.

- Può essere usato in combinazione con la vulnerabilità delle applicazioni per attaccare specifici host essendo camuffati come un altro utente
- Può essere usato per gli attacchi DoS (in alternativa ai botnets) in quanto messaggi provenienti da IP sorgenti differenti sono più difficili da filtrare

Lo spoofing può anche essere usato per attacchi man-in-the-middle (MitM o MiM), in cui l'attaccante è posto tra i due hosts comunicanti camuffato da entrambi.

- I due hosts credono di star comunicando tra loro, mentre essi stanno in realtà comunicando con l'attaccante

Per prevenire lo spoofing, possiamo usare controlli di integrità del messaggio e autenticazione end-point, permettendoci di determinare se il messaggio non ha subito modifiche o se il messaggio è stato originato dalla giusta sorgente.

9.3 LE BASI DELLA NETWORK SECURITY

Dati i precedenti tipi di attacco possiamo ora definire un set di proprietà che una comunicazione sicura dovrebbe garantire:

- Confidenzialità: solo il mittente e il destinatario previsto dovrebbero essere capaci di capire i contenuti del messaggio trasmesso (evitare lo sniffing)
- Integrità del messaggio: il contenuto della comunicazione non deve essere alterato, né in modo intenzionale né per incidente
- Autenticazione end-point: sia il mittente che il destinatario dovrebbero essere capaci di confermare l'identità dell'altra parte coinvolta nella comunicazione (evitare lo spoofing)
- Sicurezza operazionale: affidarsi ad un'infrastruttura network che previene gli hosts malevoli dall'intrufolarsi nella comunicazione

Le prime tre proprietà sono basate sul software mentre l'ultima (sicurezza operazionale) di solito si basa su specifici hardware (firewalls, sistemi di controllo di intrusione).

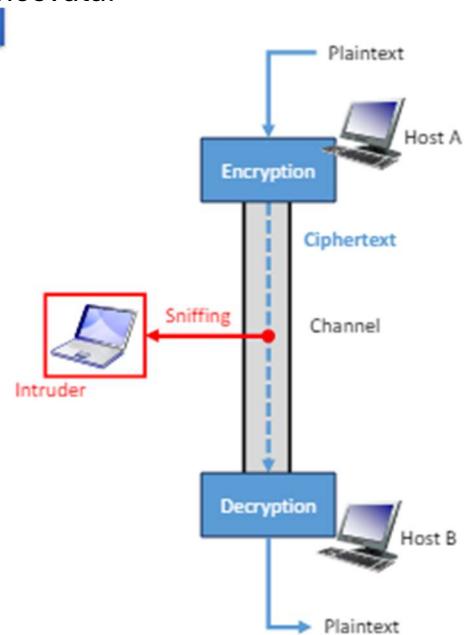
9.3.1 CRITTOGRAFIA

Una tecnica di crittografia permette ad un mittente di mascherare i dati così che diventino incomprensibili per un intruso.

- Gli intrusi non possono ottenere informazioni dai dati intercettati, ma il ricevente dovrebbe essere in grado di recuperare i dati originali dai dati mascherati.

Nella sua forma iniziale il messaggio viene chiamato plaintext (o cleartext) ed è leggibile da tutti.

Prima di iniettare il messaggio nel canale un host usa un algoritmo di crittografia per trasformare il messaggio in una forma non leggibile chiamata ciphertext che deve essere decrittografata quando viene ricevuta.



9.3.1.1 CRITTOGRAFIA: CHIAVI

In molti sistemi crittografici moderni, inclusi quelli usati da internet, la tecnica di crittografia è conosciuta e standard per tutti (incluso l'intruso). La parte sconosciuta dell'algoritmo sono le chiavi di crittografia e decrittografia.

Una chiave è una stringa alfanumerica che deve essere fornita all'algoritmo di crittografia/decrittografia per crittografare/decrittografare i messaggi.

Le chiavi di crittografia e decrittografia possono essere identiche (simmetriche) o differenti (asimmetriche).

CRITTOGRAFIA: SIMMETRICA

Nella crittografia simmetrica vi è solo una chiave che viene usata sia per la crittografia che per la decrittografia.

Crittografia: il messaggio plaintext insieme alla chiave viene passato all'algoritmo di crittografia per generare un ciphertext che può essere inviato in modo sicuro attraverso la network.

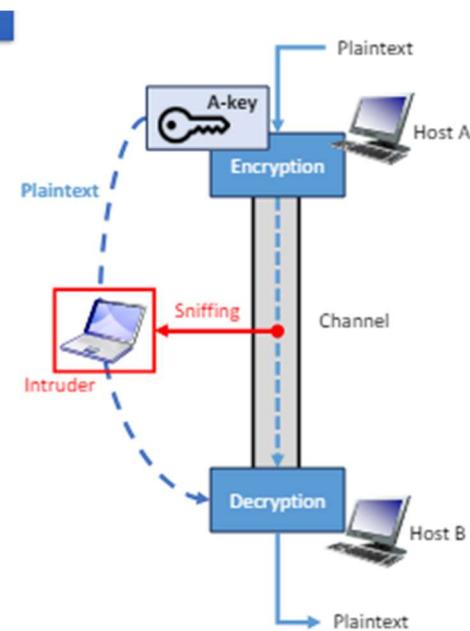
Decrittografia: il messaggio cyphertext assieme alla chiave viene passato all'algoritmo di decrittografia per ricreare il messaggio iniziale plaintext (leggibile).

CRITTOGRAFIA: SCAMBIO DELLA CHIAVE

Il problema con la crittografia simmetrica, o con la crittografia con chiave unica, è che essa richiede che la chiave segreta venga comunicata (problema dello scambio della chiave).

- Gli hosts possono usare un canale sicuro per scambiare la chiave
- Gli hosts possono usare alcuni protocolli per permettere loro di convergere su una chiave condivisa.

Se le due parti non possono stabilire uno scambio di chiave sicuro iniziale, essi non saranno capaci di comunicare in modo sicuro senza il rischio che i messaggi vengano intercettati e decrittografati da una terza parte che acquisisce la chiave durante lo scambio di chiave iniziale.



CRITTOGRAFIA: ASIMMETRICA

Nella crittografia asimmetrica vi è un sistema a due chiavi (chiave pubblica e privata).

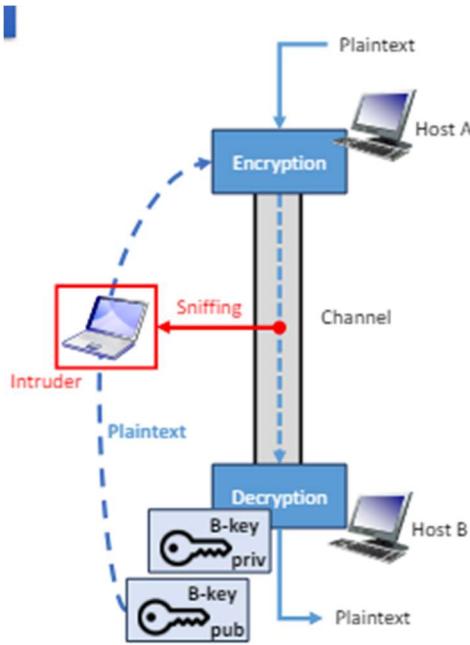
- Un messaggio che è crittografato con una chiave deve essere decrittografato con l'altra e viceversa.

L'idea è che la chiave pubblica può essere inviata su canali non sicuri o condivisa in pubblico, mentre la chiave privata è solo disponibile al suo proprietario.

Un approccio tipico è di usare la chiave pubblica per la crittografia e la chiave privata per la decrittografia:

- Se l'intruso sniffs la chiave pubblica, è comunque impossibile per lui/lei decrittografare i messaggi

- Un host A userà la chiave pubblica di B per crittografare i messaggi che possono essere visti soltanto attraverso la chiave privata di B, che è solo in possesso di B.



CERTIFICATION AUTHORITY

Nella crittografia con chiave pubblica sarebbe utile poter verificare se una chiave pubblica appartiene veramente all'entità con la quale si vuole comunicare:

- Altrimenti potremmo utilizzare la chiave di qualcun altro (attaccante) e potremmo crittografare messaggi che sono leggibili da entità illecite.

Collegare una chiave pubblica ad una particolare entità viene tipicamente svolto dal Certification Authority (CA), il cui lavoro è quello di validare le identità e rilasciare certificati. Un CA svolge i seguenti ruoli:

- 1. Un CA verifica che un'entità è chi dice di essere. Non vi è alcun protocollo per questo, bisogna fidarsi che il CA ha eseguito una rigorosa ed adatta verifica dell'identità.
 - o Funziona come un processo di selezione naturale: se un CA è inaffidabile nessuno si fiderà di lui
 - o Vi sono diversi CA federali o statali che forniscono un'affidabilità ragionevole, ma dobbiamo sempre fidarci di loro.
- 2. Una volta che il CA verifica l'identità dell'entità, il CA crea un certificato che lega la chiave pubblica dell'entità alla sua identità. Il certificato contiene la chiave pubblica e un identificatore globalmente univoco del suo proprietario (per esempio un nome o un indirizzo IP).

9.3.2 MESSAGE INTEGRITY

Il message integrity (anche conosciuto come message authentication) è il problema di controllare se:

- 1. Il messaggio non è stato manomesso
- 2. Il messaggio è stato invece originato dall'host previsto

Possiamo creare un check-item simile al checksum o CRC. Tipicamente, una funzione hash è utilizzata per creare tali oggetti.

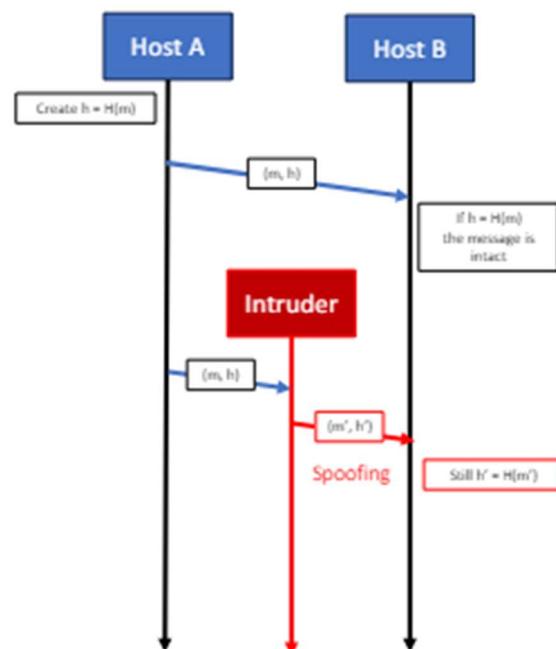
- Ricorda: una funzione hash è una qualsiasi funzione che può essere utilizzata per mappare dati di dimensione arbitraria in valori di dimensione fissa.

Una funzione hash di crittografia è una funzione H che converte un messaggio x in una stringa a dimensione fissata $H(x)$ così che risulta modo difficile (computazionalmente irrealizzabile) trovare un altro messaggio y tale che $H(y) = H(x)$.

Come per il checksum o CRC, possiamo allegare questo hash nel messaggio:

- 1. L'host A crea il messaggio m e calcola l'hash $h = H(m)$.
- 2. L'host A aggiunge h al messaggio m , creando un messaggio esteso (m, h) ed invia il messaggio esteso a B.
- 3. L'host B riceve il messaggio (m, h) e calcola $H(m)$. Se $H(m) = h$ il messaggio è intatto.

Questo approccio è ovviamente imperfetto. Un intruso potrebbe falsificare l'intero messaggio (m, h) creandone uno nuovo ad hoc (m', h') che è ancora consistente con la funzione di hashing H .



Per evitare questo, A e B necessitano di un segreto condiviso s (una chiave condivisa o una password) che è una stringa nota soltanto a loro.

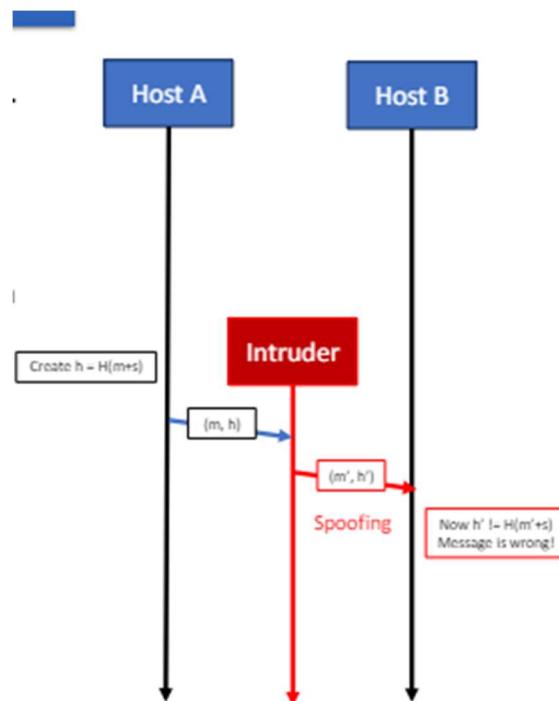
- Questo funziona praticamente come una crittografia simmetrica dove s è l'unica chiave privata.

Assumendo che tale s esista allora:

- 1. L'host A crea un messaggio $m + s$ (una concatenazione del messaggio e del segreto) e calcola l'hash $h = H(m + s)$ noto anche come message authentication code (MAC).
- 2. L'host A aggiunge il MAC al messaggio m creando un messaggio esteso (m, h) ed invia il messaggio esteso a B.
- 3. L'host B riceve il messaggio esteso (m, h) e conoscendo s calcola il MAC $H(m + s)$. Se $H(m + s) = h$, il messaggio è intatto.

Come in tutti gli approcci simmetrici, anche qui è necessario scambiare tale segreto.

- Questo segreto può essere scambiato combinando crittografia asimmetrica e certificati.



9.3.3 END-POINT AUTHENTICATION

L'autenticazione end-point è il processo di un'entità che prova la sua identità ad un'altra entità nella computer network.

Tipicamente, un authentication protocol (AP) verrebbe eseguito prima che le due parti in comunicazione eseguano qualche altro protocollo.

- È usato per stabilire le identità di entrambe le parti coinvolte prima che esse comincino a lavorare

- Esempi di tali protocolli potrebbero essere un protocollo reliable data transfer (basato su TCP), un protocollo di routing information exchange (basato su DV o LS), un protocollo e-mail (ad esempio SMTP).

Per eseguire l'authentication end-point possiamo fare affidamento sul certification authority (CA) precedentemente introdotto, ma questo non è abbastanza.

Assumiamo che l'host A ha bisogno di autenticarsi a B prima che essi possano iniziare a lavorare assieme.

Se A e B hanno già comunicato e l'indirizzo IP di A è ancora lo stesso, B potrebbe autenticare A controllando l'indirizzo IP all'interno del datagramma.

- Se l'indirizzo è uno ben conosciuto, possiamo assumere che A sta inviando il messaggio.

Questa semplice tecnica di autenticazione funziona per gli intrusi "ingenui" che cercano di mascherarsi come A, ma è chiaramente non sufficiente: un intruso "meno ingenuo" potrebbe contraffare l'indirizzo nel pacchetto.

Come abbiamo visto, è ancora possibile contraffare un datagramma IP in modo da inserire uno specifico indirizzo IP sorgente (spoofing).

Un modo per evitare lo spoofing di IP è attraverso i routers: un router potrebbe essere configurato per inoltrare solo i datagrammi legittimi, ad esempio contenenti indirizzi di sorgente IP che davvero appartengono agli hosts.

- Il router dovrebbe essere consapevole degli indirizzi IP degli hosts (altrimenti gli hosts potrebbero essere irraggiungibili).
- Il router può individuare pacchetti il cui indirizzo IP non è coerente con quello della fonte.
- I datagrammi contraffatti potrebbero essere eliminati dal router così che essi non possano danneggiare nessuno.

Sfortunatamente, questa capacità non è universalmente sviluppata né applicata, pertanto dobbiamo assumere lo spoofing come possibile.

Un altro approccio è quello di usare una password segreta (più comune): la password lavora come un segreto condiviso tra l'authenticator e la persona che viene autenticata.

- Gmail, Facebook, telnet, FTP e molti altri servizi usano l'autenticazione della password.
- Questo segreto condiviso può anche essere usato per il controllo dell'integrità.

Il primo problema qui è che l'intruso potrebbe intercettare la password dai messaggi usando sia la password che l'IP contraffatto.

L'intercettazione può essere evitata crittografando la password, così che diventi illeggibile per l'intruso.

- Nota che se usiamo una crittografia simmetrica qui, non abbiamo bisogno di una password addizionale perché la chiave privata stessa (che deve essere un segreto condiviso tra i due hosts) potrebbe funzionare anche come password.

La crittografia è utile, ma ancora non siamo al sicuro, un intruso “molto sveglio” potrebbe ancora essere capace di intrufolarsi nella comunicazione mascherato da A attraverso un playback attack.

In un playback attack (o reply attack) l’intruso cerca di imitare un host:

- L’intruso fiuta i pacchetti da A a B cercando di ottenere la versione crittografata della password di A.
- Se riesce, l’intruso potrebbe riprodurre la versione crittografata della password a B in una nuova sessione, usando la password persino senza comprenderla.
- A differenza degli attacchi standard MiM (che sono real-time), sniffing and playback sono eseguiti separatamente.

Qui il problema è che l’host B non è capace di dire se A era attivo, ad esempio se vi è una sessione attiva in esecuzione tra il reale A e B.

- Questo problema è qualcosa di simile allo stabilimento della connessione TCP in cui i due hosts usano un handshake a tre vie per assicurare che entrambi siano attivi sulle due estremità.

Nell’handshake TCP gli hosts usano numeri di sequenza iniziali casuali (oppure numero di sequenza molto vecchio) per impedire a possibili ritrasmissioni di essere interpretate come segmenti SYN/ACK. Una simile idea può essere adottata per scopi di autenticazione.

9.3.3.1 END-POINT AUTHENTICATION: NONCE

In questo caso usiamo un nonce: un numero random o pseudo random che un protocollo usa soltanto una volta nella vita.

Esempio: assumiamo che A e B condividano una chiave simmetrica, il nonce può essere usato come segue:

- 1. L’host A invia un messaggio “I am A” a B.
- 2. L’host B sceglie un nonce R e lo invia ad A.
- 3. L’host A crittografa il nonce insieme alla password usando la chiave segreta simmetrica A-B ed invia il nonce crittografato indietro a B.
- L’host B decrittografa il messaggio ricevuto. Se il nonce decrittografato è uguale al nonce inviato ad A, allora A è autenticato.

Questa password diventa una sorta di password da usare una sola volta:

- Tutte le passwords crittografate hanno questo nonce number con esso, così che essi non possano riutilizzarlo in un’altra sessione (da qualcun altro)

- Se la password + nonce è correttamente ricevuta da B, possiamo ragionevolmente assumere che vi è un A attivo dietro di esso.

9.4 SSL

Finora abbiamo visto che diverse tecniche di crittografia possono essere utilizzate ed integrate per fornire confidenzialità, integrità dei dati e autenticazione end-point alle connessioni TCP.

Tali tecniche vengono implementate attraverso la Secure Socket Layer (SSL) che è una versione migliorata del TCP che fornisce servizi di sicurezza.

- Vi è anche una versione leggermente modificata della versione 3 di SSL chiamata Transport Layer Security (TLS)

SSL è supportata da tutti i più popolari browsers Web and Web servers.

- È usata da Gmail così come da anche tutti i siti di commercio in internet, come Amazon o eBay.

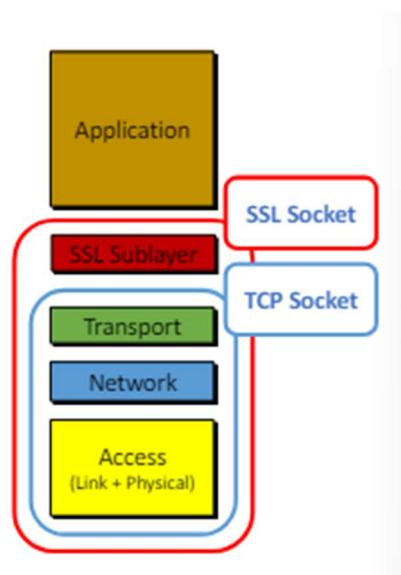
È possibile notare che spesso la connessione tra il nostro browser e un sito web utilizza HTTPS (http+SSL/TLS) piuttosto che HTTP.

Siccome SSL mette in sicurezza TCP, può essere utilizzato da qualsiasi applicazione che viene eseguita su TCP.

SSL fornisce una semplice Application Programmer Interface (API) con le sockets, che è simile e analogo all'API di TCP. Quando un'applicazione vuole utilizzare SSL, l'applicazione include librerie e classi di SSL.

Sebbene SSL tecnicamente risiede nel livello applicativo, dalla prospettiva dello sviluppatore sembra simile ad un protocollo di trasporto che fornisce i servizi di TCP migliorati mediante servizi di sicurezza.

SSL ha tre fasi principali: handshake, key derivation, data transfer.



9.4.1 SSL: HANDSHAKE

Assumiamo che un client host (B) voglia usare SSL per comunicare con un server (A). Durante la fase SSL handshake, il client B necessita di:

- Stabilire una connessione TCP con A
- Verificare che A sia veramente A (non un server fake)
- Mandare ad A una master secret key (segreto condiviso)

Il processo funziona come segue:

- Una volta che la connessione TCP è stabilita, B invia ad A un messaggio hello
- Il server A risponde con un certificato, che contiene la sua chiave pubblica (asimmetrica)
- Siccome il certificato è stato firmato da un CA, il client B crede che la chiave pubblica nel certificato appartenga ad A.
- B poi genera un Master Secret (MS) che verrà utilizzato solo per questa sessione SSL (nonce), crittografa l'MS con la chiave pubblica di A e la invia ad A.
- Il server A decrittografa il messaggio con una chiave privata per ottenere MS. Dopo questa fase, solo B ed A conoscono il master secret per questa sessione SSL.

9.4.2 SSL: KEY DERIVATION

Siccome MS è un segreto condiviso tra B ed A, può essere utilizzato come chiave simmetrica per ogni controllo di crittografia ed integrità dei dati durante la sessione.

D'altro canto, è più sicuro per A e B usare differenti chiavi per la crittografia e il controllo di integrità nei due flussi di dati (da A a B e da B ad A).

Le quattro chiavi possono essere generate a partire da MS:

- $E_{B \rightarrow A}$ = chiave di crittografia della sessione per i dati inviati da B ad A
- $M_{B \rightarrow A}$ = chiave MAC della sessione per i dati inviati da B ad A
- $E_{A \rightarrow B}$ = chiave di crittografia della sessione per i dati inviati da A a B
- $M_{A \rightarrow B}$ = chiave MAC della sessione per i dati inviati da A a B

9.4.3 SSL: DATA TRANSFER

SSL mette in sicurezza TCP scomponendo il flusso dei dati (dall'application) in records. Per ciascun record:

- 1. SSL aggiunge un MAC ad ogni record per il controllo integrità (record_i+MAC)
- 2. Il record modificato viene crittografato
- 3. Il record crittografato viene passato a TCP per la trasmissione

Questo processo assicura l'integrità dei dati per i singoli record, ma cosa succede per l'intero flusso?

Siccome soltanto il payload è crittografato, un MiM tra A e B è ancora in grado di danneggiare il flusso invertendo o rimuovendo i segmenti:

- Per esempio, il MiM può catturare due segmenti consecutivi, invertirne l'ordine assieme ai loro numeri di sequenza TCP, ed inviare i due segmenti invertiti al destinatario.
- Il destinatario potrebbe non notare l'inversione fino a quando i dati non arrivano al livello applicativo.

Per evitare questo problema SSL integra anche un numero di sequenza all'interno del messaggio prima della crittografia:

- Il reale messaggio crittografato sarà record_i+MAC+sequenceNumber_i
- Conoscendo la procedura e il MAC, il destinatario sarà capace di certificare l'integrità del messaggio

9.5 FIREWALL

Un firewall è una combinazione di hardware e software che isola (protegge) una network da Internet permettendo/negando ai pacchetti di passare.

Tutto il traffico in ingresso/in uscita dovrebbe passare attraverso il firewall e solo al traffico autorizzato (definito tale dalle politiche della sicurezza locale) è permesso passare.

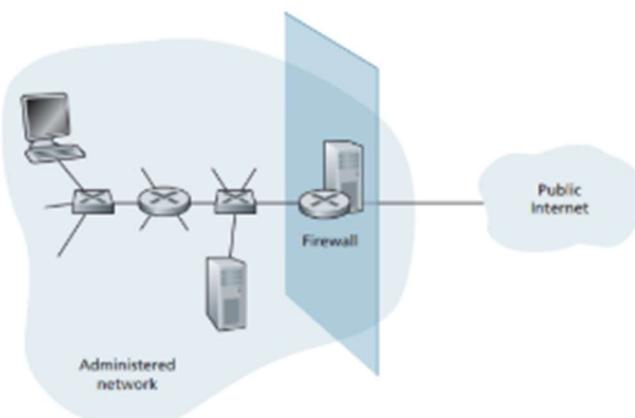
- Il firewall lavora come un singolo access point alla network pubblica, ma grandi organizzazioni possono avere livelli multipli o distribuiti di firewall.

Il firewall stesso deve essere immune alla penetrazione.

- Se compromesso, può fornire un falso senso di sicurezza.

Adesso vedremo 3 approcci:

- 1. Traditional packet filtering
- 2. Stateful filtering
- 3. Application gateway



9.5.1 FIREWALL: TRADITIONAL PACKET FILTER

Un packet filter è tipicamente implementato su un gateway (il router che connette la network locale all'ISP).

Esso esamina ogni singolo datagramma, determinando se il datagramma dovrebbe essere accettato o scartato a seconda delle regole stabilite dallo specifico amministratore.

- Si può avere uno specifico set di regole per far uscire o entrare datagrammi o per le singole interfacce.

Le regole sono tipicamente basate su:

- Indirizzi IP di sorgente o destinazione
- Il tipo di protocollo nell'apposito campo del datagramma IP (TCP, UDP, ICMP, ecc)
- Porta di sorgente o di destinazione
- I bits TCP flag (SYN, ACK, ecc)
-

L'amministratore di una network configura il firewall basandolo su una politica dell'organizzazione, qui ci sono alcuni esempi:

- Concedere solo traffico web settando una regola che blocca tutti i segmenti TCP SYN aventi porta di destinazione differente da 80
- Negare (alcuni) servizi di streaming settando una regola che blocca traffico non critico UDP (spesso usato per lo streaming)
- Negare il ping in ingresso settando una regola che blocca le risposte sul ping ICMP in uscita.

Una politica di filtraggio può anche basarsi su una combinazione di indirizzi e numero di porte (può essere IP-specific).

- Per esempio, un filtering router potrebbe inoltrare tutti i datagrammi Telnet (numero di porta 23) eccetto coloro che vanno o provengono da una lista di specifici indirizzi IP
- Nota: tali politiche non forniscono protezione contro lo spoofing.

Le regole del firewall vengono implementate nei router con access control lists.

Consideriamo l'esempio di una semplice access control list per un'interfaccia che connette un router con l'ISP.

- Vogliamo che soltanto il traffico web passi.

Possiamo settare 4 regole come segue:

- La prima regola è concedere a qualsiasi pacchetto TCP con porta di destinazione 80 di lasciare la network dell'organizzazione.

- La seconda regola concede ad ogni pacchetto TCP con porta sorgente 80 e flag ACK=1 di entrare nella network dell'organizzazione. (ricorda che tutti i messaggi web trasportano numeri ACK)
- La terza e quarta regola concedono insieme ai pacchetti DNS di entrare e uscire dalla network dell'organizzazione.

Vi è un problema con il traditional filtering: esso è stateless.

Anche se restrittiva, questa tabella concede ad ogni pacchetto in arrivo dall'esterno con ACK=1 e porta sorgente 80 di superare il filtro.

Sfortunatamente tali pacchetti sono noti per essere utilizzati nei DoS attacks.

La soluzione più ingenua consisterebbe nel bloccare anche i pacchetti TCP ACK, ma tale approccio impedirebbe agli utenti interni di navigare sul web.

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

9.5.2 FIREWALL: STATEFUL PACKET FILTER

Gli stateful filters risolvono questo problema tracciando tutte le connessioni TCP in corso in una connection table, per comprendere se il traffico è dovuto ad una legittima connessione:

- Il firewall riconosce l'inizio di una nuova connessione (la sequenza SYN-SYNACK-ACK dell'handshake a tre vie) e la fine di una connessione (il pacchetto FIN)
- Il firewall può anche (conservativamente) assumere che la connessione finisce quando non è stata rilevata alcuna attività sulla connessione per un certo periodo di tempo (60 secondi)

Una stateful table tipicamente include una colonna “check connection” che specifica se i pacchetti si trovano entro la connessione stabilita.

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

In this case only ACK packets within established connections are allowed.

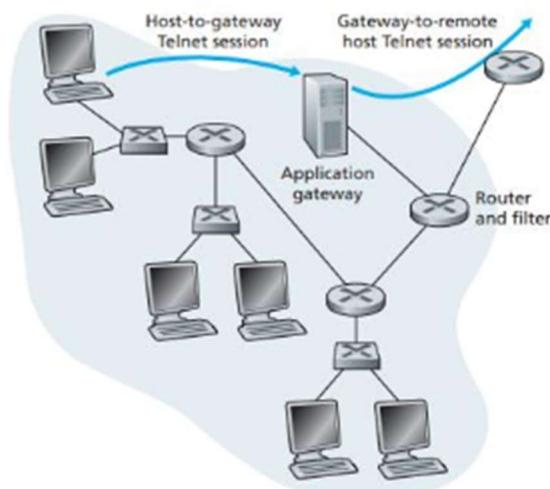
9.5.3 FIREWALL: APPLICATION GATEWAY

I suddetti metodi di filtraggio principalmente controllano IP e porte, ma potrebbe essere utile concedere o negare funzionalità a seconda degli utenti o delle applicazioni, per esempio:

- Solo ai tecnici potrebbe essere concesso l'utilizzo di determinati protocolli (ad esempio Telnet con il mondo esterno) che sono negati ai normali utenti.
 - Solo specifici tipi di messaggi o comandi sono concessi all'interno di un protocollo.

Tale compito è al di fuori delle capacità dei filtri traditional e stateful in quanto l'informazione riguardante l'identità degli utenti interni è un dato a livello applicativo e non è pertanto incluso negli headers IP/TCP/UDP.

Un application gateway o un application-level gateway (AGL) ci permette di filtrare i pacchetti a seconda dei dati del livello applicativo. Questi sono tipicamente implementati come un server separato che lavora in combinazione con il firewall.



L'idea generale è quella di negare tutte le connessioni ad uno specifico protocollo (Telnet) eccetto quelle che sono da/per l'application gateway.

Se un utente vuole usare l'application gateway (e poi il protocollo riservato) lui/lei deve effettuare il log in (mediante user ID e password).

- Qui il server controlla se l'utente ha il permesso per quel protocollo.
 - In questo caso tutte le richieste/risposte sono inoltrate attraverso l'application gateway (proxy).

Le networks interne spesso dispongono di molteplici application gateways per differenti protocolli (Telnet, HTTP, FTP, e-mail ecc).

Vi sono alcuni svantaggi:

- Un different application gateway è necessario per ciascuna applicazione
 - Penalità alle prestazioni dovuta all'utilizzo del proxy (specialmente con molti utenti)
 - Software su dispositivi host devono sapere come usare l'application gateway

9.6 SISTEMI DI RILEVAMENTO INTRUSIONE

Per individuare alcuni tipi di attacchi potrebbe essere necessario effettuare un'ispezione dettagliata del pacchetto controllando traffico sospetto oppure le caratteristiche degli attacchi noti.

Il Intrusion Detection System (IDS) è un gruppo di dispositivi specializzati che monitorano la network ricercando pacchetti sospetti.

- Il ruolo primario dell'IDS è di riconoscere pacchetti potenzialmente malevoli e di allertare l'amministratore della network (quest'ultimo poi è incaricato di fare qualcosa)
- In più è anche possibile bloccare potenziali pacchetti malevoli. In questo caso ci possiamo riferire ad esso con il nome Intrusion Prevention System (IPS)

Le organizzazioni si basano su IDS per individuare un largo range di attacchi (o di potenziali attacchi):

- Scansione della network o delle porte
- DoS flooding
- Worms e virus
-

Il sistema è spesso composto da:

- Uno o più sensori IDS
- Un singolo centrale processore IDS che colleziona ed integra l'informazione e invia gli allarmi

IDS può essere:

- Signature-based: mantiene un database estensivo delle firme degli attacchi, ad esempio un set di regole relative all'attività di intrusione (il più comune)
- Anomaly-based: crea un profilo di traffico e controlla i flussi che sono statisticamente inusuali.

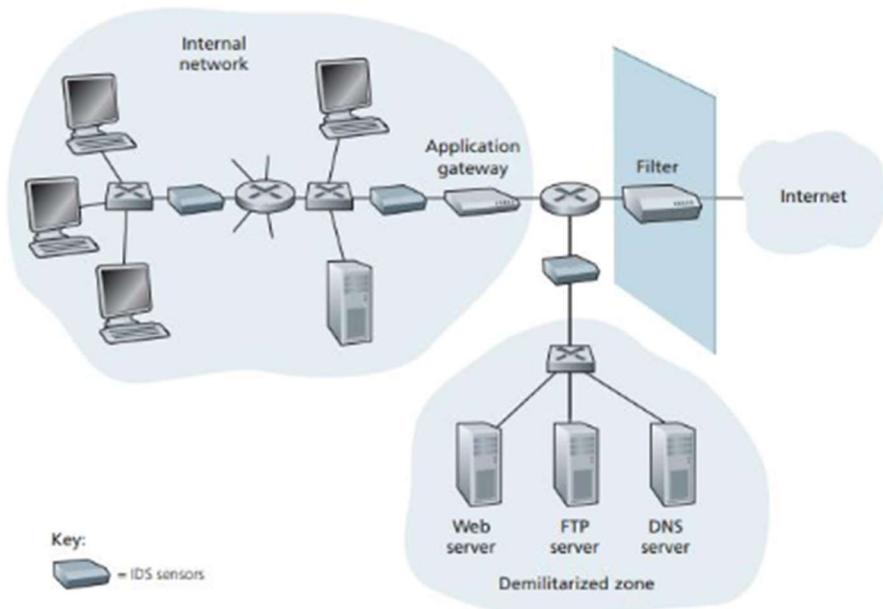
DEMILITARIZED ZONE

Nelle ampie network vi è spesso il problema di avere 2 differenti livelli di sicurezza per dispositivi differenti.

- Per esempio, ai servers che necessitano di comunicare con il mondo esterno farebbe comodo un filtering più leggero.

La demilitarized zone (DMZ) è una regione a bassa sicurezza in cui le restrizioni sono limitate e i servers possono essere ospitati.

Un tipico approccio è quello di porre la DMZ in mezzo a due firewalls: uno esterno (meno restrittivo) e uno interno (più restrittivo).



Esempio di DMZ all'interno di un IDS:

- La regione di alta sicurezza è protetta da un packet filter e da un application gateway ed è monitorata dai sensori IDS.
- La regione di bassa sicurezza (la DMZ) è protetta soltanto dal filtro e monitorata dai sensori.

10 NETWORK PROGRAMMING

10.1 INTRODUZIONE

Il network programming è l'atto di creare applicazioni network che vengono eseguite su molteplici, forse differenti, dispositivi.

- Le network applications possono basarsi su differenti linguaggi di programmazione, code libraries, oppure sistemi operativi.
- I dispositivi coinvolti nelle applicazioni di una network possono essere piuttosto eterogenei (possono essere differenti in termini di hardware, architettura ecc)

Tali applicazioni si affidano alle infrastrutture della network per comunicare.

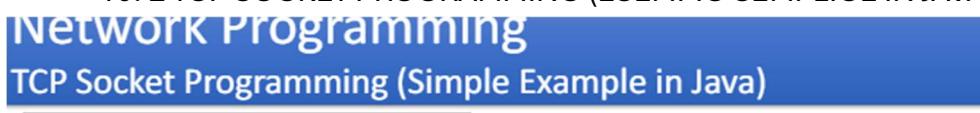
Fortunatamente le infrastrutture della network sono standardizzate, non vi è bisogno di preoccuparsi del networking siccome la maggior parte dei processi sono nascosti e gestiti da sockets (black-box).

Le sockets sono largamente utilizzate in diversi linguaggi di programmazione e sistemi operativi.

La programmazione network spesso richiede abilità in molteplici aree e strumenti di programmazione:

- Differenti linguaggi di programmazione (Java, Python, C/C++, ecc)
- Sistemi operativi (Linux, Windows, ecc)
- Stack ISO/OSI e protocolli (HTTP, TCP, UDP, ecc)
- Formati di scambio di dati (JSON, YAML, XML, ecc)
- REST APIs
-

10.2 TCP SOCKET PROGRAMMING (ESEMPIO SEMPLICE IN JAVA)



```
import java.io.*;
import java.net.*;

public class TCP_server {
    public static void main(String[] args) {
        if (args.length < 1) return;
        int port = Integer.parseInt(args[0]);
        try (ServerSocket welcome_socket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);
            while (true) {
                Socket new_socket = welcome_socket.accept();
                InputStream input = new_socket.getInputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(input));
                OutputStream output = new_socket.getOutputStream();
                PrintWriter writer = new PrintWriter(output, true);
                System.out.println("New client connected");
                String msg = reader.readLine();
                System.out.println("Received: " + msg);
                writer.println("Hello from Server");
                new_socket.close();
                System.out.println("Connection closed");
            }
        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

```
import java.net.*;
import java.io.*;

public class TCP_client {
    public static void main(String[] args) {
        if (args.length < 2) return;
        String client_name = args[0];
        String hostname = "127.0.0.1";
        int port = Integer.parseInt(args[1]);
        try (Socket socket = new Socket(hostname, port)) {
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);
            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            writer.println("Hello from " + client_name);
            String msg = reader.readLine();
            System.out.println(msg);
            socket.close();
            System.out.println("Connection closed");
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Network Programming

TCP Socket Programming (Example in C/C++ and Java)

```
// client-side [includes omitted]
int main() {
    int sockfd, status;
    char buffer[1024];
    const char *hello = "Hello from client";
    struct sockaddr_in servaddr;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n;
    if ((status = connect(sockfd, (struct sockaddr *)&servaddr,
        sizeof(servaddr))) < 0) {
        printf("Connection Failed \n");
        return -1;
    }

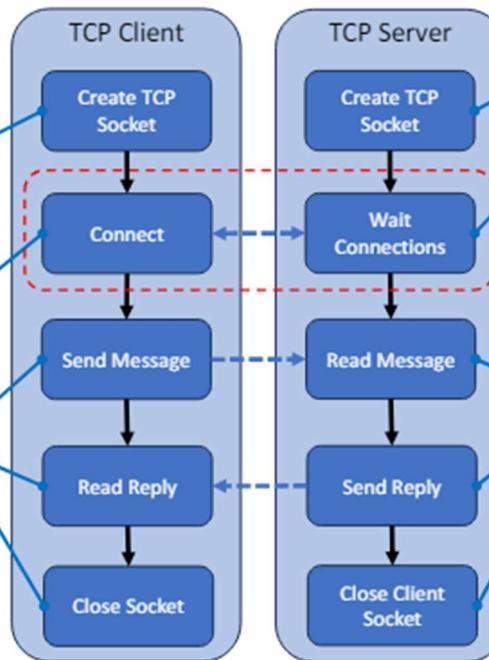
    send(sockfd, hello, strlen(hello), 0);
    std::cout << "Hello message sent." << std::endl;

    n = read(sockfd, buffer, 1024);
    std::cout << "Received " << n << "buffer< " << std::endl;

    close(sockfd);

    return 0;
}
```

TCP client in C/C++
from previous lesson!



```
import java.io.*;
import java.net.*;

public class TCP_server {
    public static void main(String[] args) {
        if (args.length < 1) return;

        int port = Integer.parseInt(args[0]);

        try (ServerSocket welcome_socket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket new_socket = welcome_socket.accept();

                InputStream input = new_socket.getInputStream();
                BufferedReader reader = new BufferedReader(
                    new InputStreamReader(input));

                OutputStream output = new_socket.getOutputStream();
                PrintWriter writer = new PrintWriter(output, true);

                System.out.println("New client connected");

                String msg = reader.readLine();
                System.out.println("Received: " + msg);

                writer.println("Hello from Server");

                new_socket.close();
                System.out.println("Connection closed");
            }
        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

10.3 DATA EXCHANGE

Negli esempi precedenti abbiamo visto semplici programmi scambiarsi stringhe (plain text), ma ciò è raramente un caso realistico.

Le reali applications network tipicamente si scambiano strutture dati (strutture, liste ecc), pertanto una più complessa rappresentazione dell'informazione potrebbe essere utile.

Diversi linguaggi (o formati) di scambio di dati (o interchange) sono stati usati nel corso degli anni per fornire un modo standard di rappresentare e scambiare strutture dati per applications network.

A causa della loro leggibilità da parte dell'uomo, i formati più comuni (e aperti) usati per lo scambio di dati sono:

- XML
- YAML
- JSON

10.3.1 DATA EXCHANGE: XML

XML (eXtensible Markup Language) è un linguaggio designato per essere facilmente comprensibile sia dagli umani che dai computer (simile a HTML).

- Tipicamente ha estensione file: .xml

Il concetto chiave di XML sono i tags. Un tag è un costrutto markup avvolto tra parentesi angolari (<..>), ogni contenuto in un file XML è circondato dallo start-tag (<TAGNAME>) e dal end-tag (</TAGNAME>).

- I contenuti all'interno dei tags possono essere valori semplici (stringhe, numeri, bool, ecc) o altri tags.

A differenza di HTML, in XML pochi tags sono predefiniti, dovremmo creare tags a nostra discrezione.

- I documenti XML sono convenzionalmente avvolti in <xml>...</xml> tags, ma questi sono raramente implementati per il data exchange.

Questo è un semplice esempio di come rappresentare i dati riguardanti una persona. Qui abbiamo:

- Un root tag <person> che colleziona i dati di una persona. Esso contiene 3 tags interni:
 - o Il tag <personID> specifica l'ID della persona (ad esempio di un database) e contiene un numero
 - o Il tag <firstName> contiene una stringa con il nome della persona
 - o Il tag <lastName> contiene una stringa con il cognome della persona

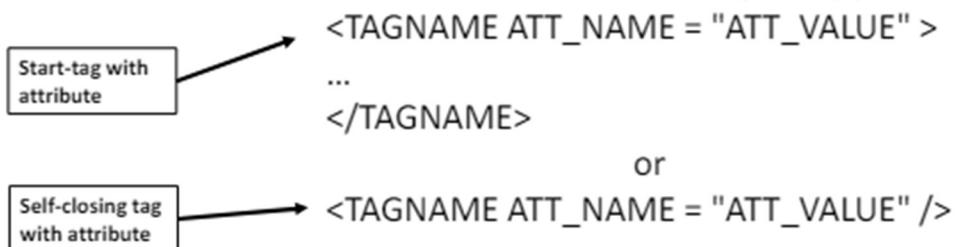
È importante notare che tutti i contenuti (non importa quanto banale) sono avvolti da una start-tag ed una end-tag.

```
<person>
  <personID> 77 </personID>
  <firstName> John </firstName>
  <lastName> Doe </lastName>
</person>
```

Potrebbe essere scomodo specificare tutti i contenuti all'interno dei tags. XML offre la possibilità di specificare attributi e tags self-closing per semplificare la definizione del contenuto.

Un self-closing tag è un tag standalone che si chiude da solo: <TAGNAME />

Gli attributi sono coppie nome-valore che esistono entro uno start-tag o un self-closing tag. Il valore degli attributi deve essere posto all'interno di doppie virgolette (""):



Seguendo il precedente esempio, possiamo usare attributi e tags self-closing per rappresentare una persona in modi diversi:

- Inserire alcuni attributi nella start-tag
- Inserire tutti gli attributi in una tag self-closing

```
<person personID = "77" >
  <firstName> John </firstName>
  <lastName> Doe </lastName>
</person>
```

```
<person personID = "77" firstName = "John" lastName = "Doe" />
```

Possiamo specificare un gruppo di persone annidando multiple persone in un tag addizionale chiamato `<people>`.

```
<people>
  <person personID = "77" >
    <firstName> John </firstName>
    <lastName> Doe </lastName>
  </person>
  <person personID = "78" >
    <firstName> Alice </firstName>
    <lastName> Doe </lastName>
  </person>
</people>
```

10.3.2 DATA EXCHANGE: YAML

Yaml (Yaml Ain't Markup Language) è un linguaggio designato per essere minimale e facilmente leggibile dall'uomo.

- Estensione tipica del file: `.yaml` o `.yml`

A differenza di XML, in YAML i contenuti vengono definiti attraverso spazio ed indentazioni (come in python).

- Nota: le indentazioni tab-based non sono permesse, devono essere utilizzati solo gli spazi bianchi.

In YAML i nomi e i valori sono separati dal simbolo dei due punti (NAME: VALUE) e vi è una specifica sintassi per definire gli elementi come strutture, liste o dizionari.

Questo è un semplice esempio di come rappresentare i dati riguardo una persona. Abbiamo:

- Una struttura “person” che contiene 3 campi interni:
 - o Il campo “personID” specifica l’ID della persona (ad esempio di un database) come un numero
 - o Il campo “firstName” contiene una stringa con il nome della persona
 - o Il campo “lastName” contiene una stringa con il cognome della persona

È possibile notare che le virgolette non sono necessarie per determinare i valori.

```
person:  
  personID: 77  
  firstName: John  
  lastName: Doe
```

Con YAML possiamo anche rappresentare liste e dizionari.

Una lista può essere rappresentata in due modi:

- 1. Come una struttura i cui elementi sono preceduti da un dash (-)
- 2. Come una sequenza di elementi separati da una virgola all’interno di parentesi quadre ([E1, E2, ...])

Un dizionario viene rappresentato come una sequenza di coppie nome-valore separate da virgolette all’intero di parentesi graffe ({ N1: V1, N2: V2, }).

Estendendo l’esempio precedente, possiamo definire una lista di persone contenente 2 strutture person.

Possiamo definire una lista di nomi.

Possiamo definire i campi di una struttura (la person in questo caso) come un dizionario.

- Questo metodo è largamente usato per il data exchange.

```
people:  
  - person:  
      personID: 77  
      firstName: John  
      lastName: Doe  
  - person:  
      personID: 78  
      firstName: Alice  
      lastName: Doe
```

```
names: [John, Alice, Bob]
```

```
person: {personID: 77, firstName: John, lastName: Doe}
```

10.3.3 DATA EXCHANGE: JSON

Il JSON (JavaScript Object Notation) è un formato semplice per il data exchange che è leggibile dagli umani, facilmente analizzato dalle macchine e basato su convenzioni simili al C per la rappresentazione dei dati.

- Estensione tipica del file: .json.

JSON è in qualche modo un compromesso tra semplicità ed efficienza, ed è considerato come un linguaggio ideale per il data interchange (uno dei più usati).

- La convenzione simile al C è anche ben conosciuta dai programmati.

Come in YAML, JSON data è specificato come una coppia nome-valore divisa dal simbolo dei due punti (:) dove il nome è una stringa (all'interno delle doppie virgolette ""):

“DATANAME” : DATAVALUE

Gli oggetti JSON sono gruppi di dati all'interno di parentesi graffe dove i data items sono separati da virgole ({ “N1”: V1, “N2”: V2, }).

Le JSON arrays sono rappresentate come dati o oggetti separati da virgole all'interno di parentesi quadre ([“N1: V1, “N2: V2, ...]).

A differenza di YAML, i valori per i dati di JSON hanno una sintassi simile a c per differenti tipi. I possibili valori sono:

- String (all'interno delle doppie virgolette):
 - o “name”: “Bob”
- Number (integer, float, double):
 - o “age” : 27
 - o “weight” : 60.5
- Array (contenente i dati generici di JSON):
 - o “pets” : [“cat”, “dog”]
 - o “siblings” : []
- Boolean (true/false):
 - o “isAlive” : true
- Null (ad esempio not available):
 - o “phoneNumber” : null
- JSON object.

Questo è un esempio di come rappresentare i dati di una persona. Abbiamo:

- Un root object che contiene i dati di “person”, il cui valore è un JSON object addizionale avente 3 campi:
 - o Il “personID” specifica l'ID della persona (ad esempio di un database) e contiene un numero
 - o Il “firstName” contiene una stringa con il nome della persona

- Il “lastName” contiene una stringa con il cognome della persona.

In questo caso l’indentazione non è necessaria (come in C) ma è solo utilizzata per una migliore visualizzazione.

```
{  
  "person": {  
    "personID": 77,  
    "firstName": "John",  
    "lastName": "Doe"  
  }  
}
```

Anche in questo caso, possiamo definire una lista di persone inserendo multipli JSON objects all’interno di un JSON array.

```
{  
  "people": [  
    {"person": {  
      "personID": 77,  
      "firstName": "John",  
      "lastName": "Doe"  
    }},  
    {"person": {  
      "personID": 78,  
      "firstName": "Alice",  
      "lastName": "Doe"  
    }},  
  ]  
}
```

Possiamo anche semplificare la sintassi rimuovendo il dato “person”.

- Nota che il JSON array può contenere sia JSON data sia JSON objects.

```
{  
  "people": [  
    {  
      "personID": 77,  
      "firstName": "John",  
      "lastName": "Doe"  
    },  
    {  
      "personID": 78,  
      "firstName": "Alice",  
      "lastName": "Doe"  
    }  
  ]  
}
```

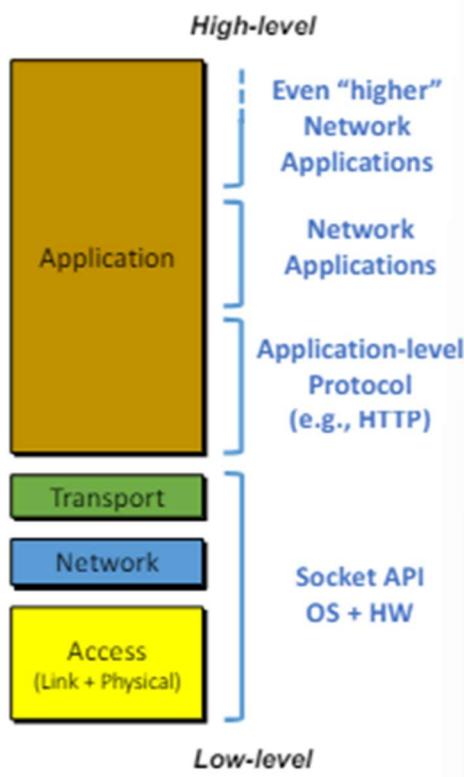
10.4 PROGETTAZIONE DI APPLICAZIONI DI RETE

Possiamo progettare applicazioni che usano i protocolli di trasporto UDP o TCP ma per i protocolli a livello applicativo possiamo scegliere tra 2 alternative:

- Standard protocol
- Proprietary (custom) protocol

Un possibile approccio (abbastanza comune) è di progettare le applicazioni “sulla cima” del protocollo standard http (come le applicazioni basate sul web):

- http è ben conosciuto: diversi elementi della network già lavorano con http (servers, browsers, ecc) e diversi linguaggi di programmazione hanno già APIs relative ad http.
- http è versatile: diverse funzioni possono essere implementate attraverso una combinazione di metodi standard http e header lines.

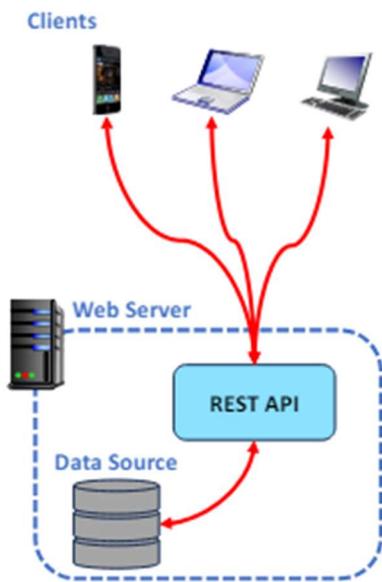


10.4.1 REST

Il REST (Representational State Transfer) è un set di principi architetturali che guida il programmatore nel definire applicazioni basate sul web modulari, scalabili e flessibili.

Le applicazioni web che seguono tali principi sono tipicamente chiamate RESTful APIs.

In teoria, i principi REST non sono legati a specifici standards, linguaggi di programmazione o tecnologie, ma sono fortemente correlati ad http.

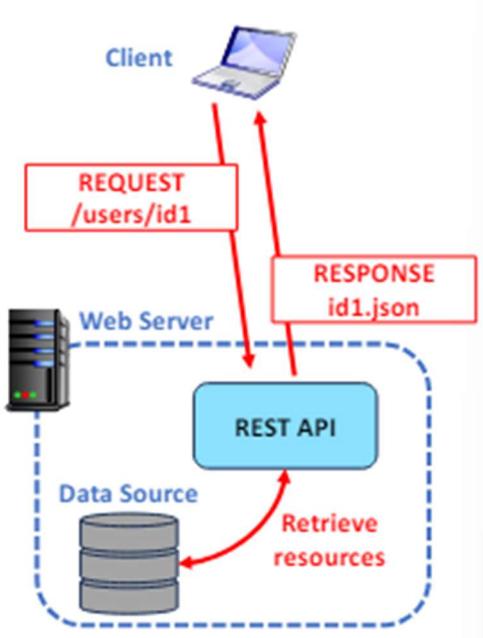


10.4.1.1 ELEMENTI ARCHITETTURALI DI REST

Un API di REST lavora come un’interfaccia tra multipli (e differenti) clients e le risorse condivise (ad esempio un database).

Gli elementi principali di REST da dover identificare sono:

- Risorse: i pezzi di informazione che gli hosts sono disposti a manipolare. Le risorse sono identificate dagli URIs (Uniform Resource Identifiers).
 - o Possiamo vedere l’URI come una forma generalizzata di URL che include anche i nomi della risorsa.
- Rappresentazioni: come le risorse sono strutturate e rappresentate in uno specifico format (ad esempio plain text, XML, YAML, JSON, ecc)



10.4.1.2 I SEI PRINCIPI DI REST

- 1. Client-server: REST si basa sulle applicazioni client-server dove i due hosts sono indipendenti:
 - o L'applicazione client conosce solo l'URI delle risorse da richiedere.
 - o L'applicazione server passa solo risorse tramite http.
- 2. Statelessness: lo stato della conversazione non viene mantenuto, ogni richiesta deve essere autonoma.
- 3. Cacheability: le risorse dovrebbero essere cacheable sul lato del client e del server.
 - o Le risposte dovrebbero anche contenere informazioni riguardo la possibilità di permettere o meno il caching per una specifica risorsa.
- 4. Interfaccia uniforme: tutte le richieste API per la medesima risorsa possono sembrare uguali, a prescindere da dove provenga la richiesta.
- 5. Architettura di sistema a più livelli: i RESTful APIs dovrebbero considerare che i messaggi possono essere inoltrati attraverso differenti intermediari (non vi è un collegamento diretto tra il client ed il server) ma questo non dovrebbe essere percepito né dal client né dal server.
- 6. Codice su richiesta (opzionale): le risposte possono anche contenere codice eseguibile (come gli applets di Java). In questi casi, il codice dovrebbe essere eseguito solo su richiesta.

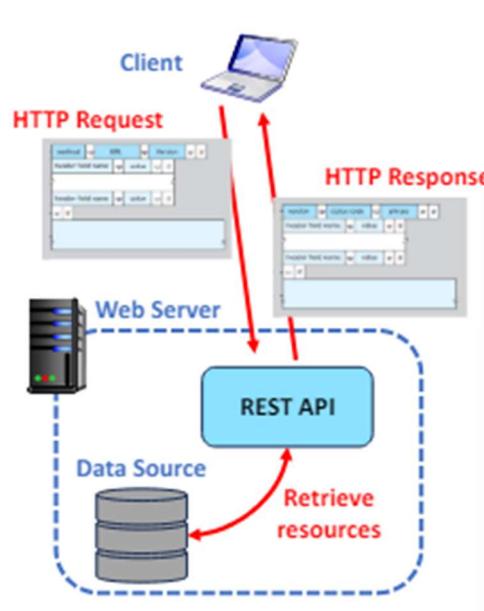
10.4.1.3 GLI API DI REST IN PRATICA

È possibile notare che i principi di REST assomigliano molto ad alcune caratteristiche di http. In un certo modo, REST definisce come usare “coraggiosamente” http in un'applicazione network.

L'idea generale è di manipolare le risorse rappresentate in modo standard attraverso messaggi rappresentati in modo standard.

Possiamo affidarci ad http per implementare la comunicazione client/server:

- Il client accede alle risorse attraverso le richieste di http
- Il server fornisce le risposte attraverso le risposte di http.



Le richieste di http in REST:

- Le risorse possono essere identificate attraverso il campo [URL] della richiesta.
- Le operazioni sulle risorse possono essere specificate all'interno del campo [Method] di una richiesta.
 - o Le operazioni CRUD (Create, Read, Update, Delete) sono tipicamente prese in considerazione, e possono essere implementate attraverso i 4 metodi di http:
 - Get: recuperare la risorsa (read)
 - Post: modificare la risorsa (update)
 - Put: aggiungere una nuova risorsa (create)
 - Delete: rimuovere una risorsa (delete)
- Possibili risorse e opzioni possono essere specificate all'interno del [Body] e [Header] rispettivamente.

La risposta http in REST:

- Le informazioni circa la richiesta sono fornite attraverso i campi [Status code] e [Phrase]. Comunemente i messaggi usati sono:
 - o 200 per le risorse gestite con successo
 - o 201 per la creazione riuscita di una nuova risorsa
 - o 404 per risorsa non trovata
 - o 405 per metodo non supportato
 - o Ecc..

Le possibili risorse e opzioni possono essere specificate in [Body] e [Header] rispettivamente.

10.4.1.4 RAPPRESENTAZIONE DELLA RISORSA

L'URL di http viene usato per identificare le risorse. I formats comuni sono:

- [http(s)]://[Domain name of REST API][:Port]/[API version]/[Path to resource]
 - Example: <http://myapi.example.com/v1/users/1>
- [http(s)]://[Domain name][:Port]/[REST API]/[API version]/[Path to resource]
 - Example: <http://example.com/myapi/v1/users/1>

I percorsi per le risorse e le sotto risorse sono comunemente specificati usando il seguente pattern:

- /[resource name]/[resource id]/[sub-resource name]/[sub-resource id] ..
 - Example: /users/1/nickname

Le risorse possono essere rappresentate in differenti formati di data exchange. Il più comune è JSON, ma diversi linguaggi possono essere usati contemporaneamente.

- Per esempio, il formato selezionato può essere specificato dalla header line "Content-type".

10.4.2 GUIDELINEA

Linee guida per nominare ed utilizzare le risorse:

- Usare i nomi al plurale non i verbi:
 - o Good: /users
 - o Bad: /doSomethingOnUsers
- Usare i metodi http invece dei termini CRUD (o simili) nei nomi:
 - o Good: /users (con il metodo GET)
 - o Bad: /getAllUsers
- Identificare risorse uniche con gli IDs:
 - o Good: /users/1
 - o Bad: /users?id=1
- È possibile utilizzare queries per selezionare o ordinare risorse:
 - o Good: users?nickname=Bob&sort=age
 - o Bad: /usersNamedBobSortedByAge