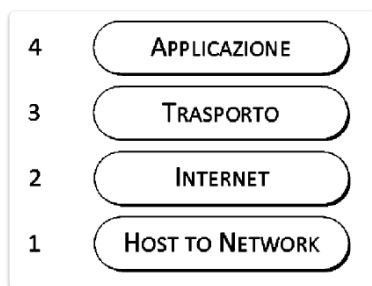


## PROTOCOLLO TCP-IP

- Detto anche modello *internet*
- Architettura a livelli basata su quanto visto nel caso ISO-OSI (seppur semplificata)



### PANORAMICA DEI LIVELLI

#### 1) HOST TO NETWORK

Definito in maniera generica perché adattabile a più contesti

- Permette la possibilità di usufruire del protocollo TCP-IP anche tra reti eterogenee, cioè tra reti che hanno livello collegamento e fisico diversi (così che il protocollo diventa adattabile a più contesti)

#### 2) INTERNET

Livello rete (IP): permette di configurare la rete in modo tale da consentire lo scambio di pacchetti tra nodi della rete

- Si preoccupa dell'*instradamento*, cioè la modalità con cui l'informazione viene gestita nella rete circa i percorsi che deve fare (di default è a datagramma, quindi *connectionless*, sarà poi eventualmente il livello trasporto a implementare tecniche di *connection oriented*)

*Riassumendo: Gestisce lo scambio di pacchetti tra coppie sorgente-destinazione in modalità connectionless non affidabile (no controllo integrità). E' attivato in modalità link-to-link (tutti i nodi della rete hanno libello rete)*

#### 3) TRASPORTO

Detto anche TCP: definisce e attua le modalità di scambio di pacchetti (*connection oriented, connectionless...*) - come nell'ISO-OSI

- I pacchetti scambiati/generati a questo livello sono detti *datagram*
- È attivato su base end-to-end (implementa quindi eventualmente il controllo integrità)
  - Quindi rende possibili entrambe le modalità *connection oriented* e *connectionless*
- Controlla talvolta (e implementa) il controllo dell'integrità della informazione ricevuta

#### 4) APPLICAZIONE

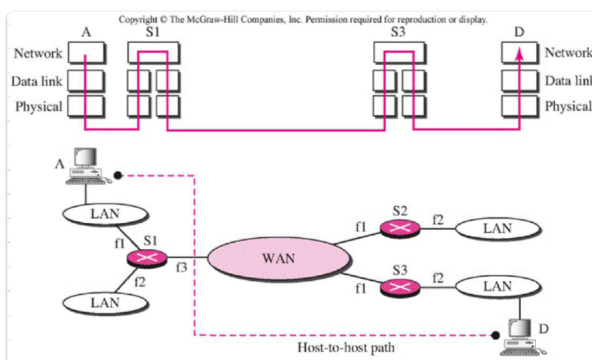
Analogo al caso ISO-OSI

#### Livelli Mancanti rispetto a ISO-OSI (sessione, presentazione). Perché?

Primo motivo: è nata prima dell'ISO-OSI

Era storicamente pensata per un unico fornitore (quindi ad esempio non serviva il livello presentazione perché il formato si supponeva univoco)

### ESEMPIO DI COLLEGAMENTO

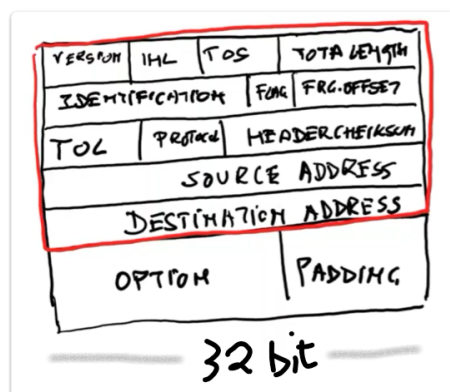


## A vuole connettersi a D

A apre il collegamento (magari a livello applicazione) e inizia il viaggio dei pacchetti:

- Dal livello rete di A si scende fino al livello fisico passando dal livello collegamento, e si invia tramite il mezzo fisico il necessario
- Il materiale passa per la rete locale LAN giunge al router S1 che fa risalire la pila di livelli fino al livello 3 per capire tutte le informazioni che gli sono state inviate (e per capire come dovrà instradare il pacchetto). Finalmente invia (riscendendo la pila) il pacchetto verso l'esterno, in particolare verso una WAN
- Il flusso arriva proprio al router S3 grazie alle informazioni descritte e si effettua il solito passaggio di risalita della pila per capire dove dovrà inviare il dato e poi riscendendo la pila invia effettivamente il dato verso la rete LAN in cui risiede il dispositivo D
- La rete LAN filtra la richiesta e manda tutto a D
- D riceve il pacchetto

## LIVELLO IP



- **Lunghezza minima intestazione:** parte in rosso → 5 righe ciascuna di 4 byte (in tutto quindi 20 byte - sono sempre presenti);
- **Opzioni aggiuntive:** option e padding

## NEL DETTAGLIO

1° riga:

- **Version:** specifica la versione del protocollo IP (esempio: IPv4)
- **IHL:** Intermediate Header Length (lunghezza intestazione intermedia) - specifica l'effettiva lunghezza della intestazione (almeno 20 byte {parte rossa}, ma se ci sono le opzioni aggiuntive è più lunga)
- **TOS:** Type of service - serve a identificare il tipo di servizio associato al datagramma e di conseguenza le politiche di gestione (attraverso il livello rete possono essere eseguite diversi servizi (es. voip / traffico dati). Ognuno avrà le sue criticità. Con il TOS si aiutano i dispositivi di rete a gestire al meglio il datagramma o permettono un inoltro migliore sulla rete). Si cerca quindi ad esempio di dare maggiore priorità a determinati percorsi - è lungo 8 bit
- **Total length:** specifica lunghezza effettiva in byte del datagramma (header + payload) - è lungo 16 bit (quindi lunghezza massima  $2^{16} - 1$  bit)

2° riga {informazioni per gestire reti non omogenee, che hanno ad esempio dimensioni diverse per quanto riguarda il campo data link}:

- **Identification:** è una etichetta per i datagrammi che utilizzano questo protocollo - serve in ricezione per poter riconoscere che un certo datagramma (pacchetto) appartiene a un determinato flusso. Utilizzato per la frammentazione (operazione di suddivisione di un datagramma in parti con formati compatibili con il livello *host to network*) - lungo 16 bit
- **Flag:** È lungo 3 bit:
  - il primo è detto bit D: se vale 1, allora il datagramma non può essere frammentato (se arriva a un router che interfaccia una rete di inoltro con formato di trasferimento non compatibile allora il frammento viene scartato - tanto l'IP come livello è non affidabile, quindi un pacchetto può essere talvolta scartato)
  - il secondo è detto bit M: serve per gestire l'invio dei frammenti (se vale 1, vuol dire che successivamente seguirà un altro frammento. Se vale 0, allora tale frammento è l'ultimo della sequenza, quindi dovrebbe essere stato inviato tutto: sarà quindi inviato al livello successivo che eventualmente esegue il controllo d'integrità)
  - il terzo bit non è utilizzato
- **Fragment offset** (piazzamento frammento): viene usato per individuare la posizione del primo byte del frammento rispetto alla intera struttura del datagramma - è lungo 13 bit
- **TTL** (Time To Live): tempo massimo di vita del datagramma nella rete (se vale zero viene scartato - è associato spesso al numero di router/dispositivi che può attraversare: ogni volta che ne incontra uno, viene decrementato di uno questo valore [scandisce il tempo di vita - evita il problema della "inondazione": troppi router di passaggio]) - è lungo 8 bit
- **Protocol:** identifica il tipo di protocollo e quindi il relativo servizio dello strato superiore (trasporto) a cui il datagramma va indirizzato (identifica il servizio TCP oppure UDP) - è lungo 8 bit

- **Header Checksum:** consente di garantire che non ci siano errori (di trasmissione) nel campo di testata (attraverso un confronto tra specifica mittente e specifica destinatario {cfr. dopo}) Svantaggio: tempo e modalità con cui viene utilizzato: ogni volta che le informazioni nella testata subiscono variazioni, è necessario riscrivere il campo (ha un impatto ad esempio quando si utilizza la frammentazione, in cui si riscrive ogni volta l'offset o talvolta anche i campi di flag - problemi anche quando cambia il destination address etc...). Riassumendo: utile per evitare errori ma rallenta la velocità di trasmissione - è lungo 16 bit

3° e 4° riga:

- **Source address:** indirizzo di sorgente - è lungo 32 bit ( $2^{32} - 1$  possibili indirizzi)
- **Destination address:** indirizzo di destinazione - è lungo 32 bit ( $2^{32} - 1$  possibili indirizzi)

6° riga (opzionale):

- **Option:** vari utilizzi:
  - utile per implementare un *servizio di sicurezza* (ad esempio nascondere i dati {payload} che si stanno trasmettendo con metodi crittografici - rendere inaccessibile a chi non è autorizzato)
  - utile per l'*istadamento dalla sorgente* (se vogliamo individuare il percorso migliore da una certa sorgente a una certa destinazione senza conoscere la rete com'è fatta allora si invia un "pacchetto esploratore" in modalità broadcast con TTL settato a 1 si scoprono tutte le strade di "andata (fino alla destinazione) e ritorno (fino alla sorgente)" - di queste si sceglie quella che arriva prima, che è la migliore) - nel campo option si tiene traccia degli indirizzi di ciascun nodo (dispositivo) a cui si giunge, così da avere tutte le informazioni che servono poi per specificare il percorso (servono quindi per gestire il routing)
  - utile per *controllare la congestione della rete* (tenere traccia di quanto tempo ci mette un dispositivo a elaborare il datagramma)
  - altri utilizzi...
- **Padding:** valori di riempimento (fittizi) per arrivare a 32 bit. Sono aggregato sulla stessa riga di options

## FRAMMENTAZIONE E RICOMBINAZIONE

Utile introdurre una misura:

- **MTU:** Maximum Transmission Unit (valore massimo delle dimensioni del datagramma accettate dalla rete di inoltro su cui si appoggia il layer IP)

Facciamo un esempio numerico per capire:

Un host vuole inviare un datagramma su una rete TCP/IP di dimensione 4000 byte totali (payload + almeno 20 byte della testata)

- Si suppone che questo sia riferito a una rete *ethernet* per la quale il formato accettato è di  $L_M = 1500$  byte. Si deve quindi procedere alla frammentazione (essendo  $4000 > 1500$ )

**VINCOLI:**

- ogni frammento deve avere una propria testata IP
- tutti i frammenti eccetto l'ultimo devono avere una lunghezza multiplo intero di 8 byte (se non lo sono, vanno completato con bit "fittizi", e nel caso quindi non si sfrutta al massimo la rete → svantaggio)

## PROGETTO

Abbiamo che:

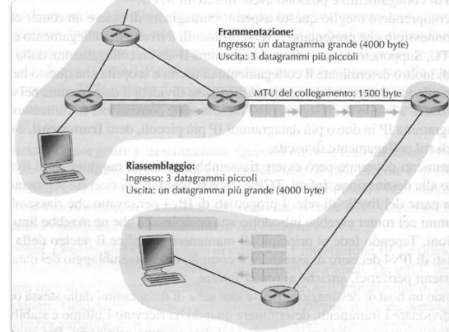
- $L_M = 1500$ 
  - Di cui: 1480 byte payload & 20 byte testata

Fortunatamente 1480 è un multiplo intero di 8, quindi non c'è bisogno di utilizzare alcun campo (bit) fittizio

## FRAMMENTAZIONE

Per gestire la frammentazione si deve definire il valore in binario dei seguenti campi (2° riga):

- Identificatore (ID)
- Flag
- Fragment offset (spiazzamento)



- si partiziona il contenuto informativo quando il pacchetto deve passare attraverso la rete Ethernet
  - i datagrammi di dimensioni più piccoli (partizionati) hanno ciascuno il medesimo **header**
  - l'**end-user** esegue poi la **ricostruzione**

Frammento	Byte	ID	Spiazzamento	Flag
1° frammento	1480 byte nel campo dati del datagramma	Identificatore = 777	Spiazzamento = 0 (i dati saranno inseriti a partire dal byte 0)	Flag = 1 (segue altro frammento)
2° frammento	1480 byte di dati	Identificatore = 777	Spiazzamento = 185 (i dati saranno inseriti a partire dal byte 1480. (Nota: $1480 = 185 \times 8$ )	Flag = 1 (segue altro frammento)
3° frammento	1020 byte di dati (Nota: $1020 = 3980 - 1480 - 1480$ )	Identificatore = 777	Spiazzamento = 370 (i dati saranno inseriti a partire dal byte 2960. $2960 = 370 \times 8$ )	Flag = 0 (ultimo frammento)

- Nota: il campo ID viene scelto in maniera aleatoria, e identifica come già visto il datagramma (infatti tutti i frammenti hanno lo stesso ID)

🐢: ritardo consegna - l'operazione di frammentazione ha un suo tempo di esecuzione

🔒: sicurezza - un hacker esterno può mandare pacchetti piccoli con numero di spiazzamento casuali o nulli o che magari si sovrappongono con quelli già inviati/ricevuti (in modo che il sistema di ricezione, "confuso", collassa) {nell'IPv6 sarà implementato in maniera più intelligente};

🚗: poca flessibilità/tolleranza: se qualche frammento non arriva a destinazione, viene scartato l'intero pacchetto {solo se il livello trasporto usa TCP si può avere speranze di recupero}

## INDIRIZZAMENTO

- Sintassi per scrivere i campi del datagramma volta a una migliore gestione degli indirizzi IP in generale

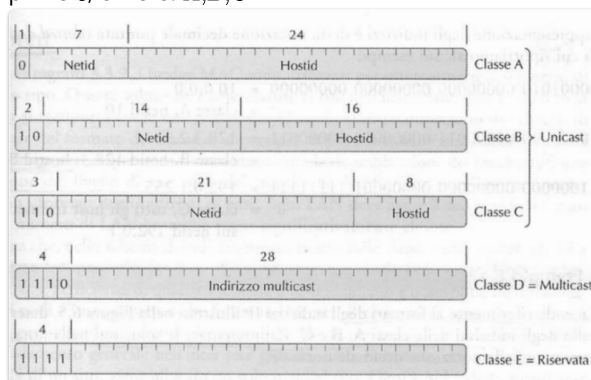
Un indirizzo IP (4 byte - 32 bit) è suddiviso in due parti:

- NET ID: identifica la **rete** di appartenenza del terminale host
- HOST ID: identifica lo specifico terminale (**host**) all'interno di una rete

Ci sono varie modalità di divisione/indirizzamento, vediamo le principali:

### INDIRIZZI BASATI SULLE CLASSI

- 5 classi: A,B,C,D,E
  - Sono utilizzati per scopi comuni solo le prime 3, ovvero: A,B,C



- Ogni classe ha un suo "metodo di divisione dei bit" circa i campi NET ID e HOST ID
- Se il primo bit è 0, allora è di classe A, altrimenti se vale 1 può essere solo B,C,D,E
- Se i primi due bit sono 10, allora è di classe B, altrimenti se sono 11 può essere solo C,D,E
- Se i primi tre bit sono 110, allora è di classe C, altrimenti se sono 111 può essere solo D,E
- Se i primi quattro bit sono 1110, allora è di classe D, altrimenti se sono 1111 può essere solo E

La classe A è destinata ad applicazioni con:

- Poche reti
- Numero alto di Host

La classe B è destinata ad applicazioni con:

- Più reti
- Un po' meno host

La classe C è destinata ad applicazioni con:

- Tante reti
- Pochi host

😊: standardizzato - più facile per la macchina comprendere l'indirizzo

😞: spreco di indirizzi - dato che ogni router dovrebbe assegnare un NET ID con potenziali indirizzi non utilizzati

Con la tecnica del **subnetting** si può dividere il NET ID in più parti, così che bastano meno indirizzi gestire lo stesso numero di dispositivi, evitando sprechi inutili

## SUBNETTING

- Maschera di rete che partiziona l'insieme di HOST ID indirizzati in sottoinsiemi. Essi vengono associati allo stesso NET ID.

## INDIRIZZO SENZA CLASSI

- I campi NET ID e HOST ID hanno dimensione variabili, non esistono quindi vincoli di dimensioni dei campi

😊: più flessibile

😞: più complicato per la macchina

## NAT

- Network Address Translation (Traduttore Indirizzi di Rete)

Ad ogni **rete** viene assegnato un solo indirizzo IP

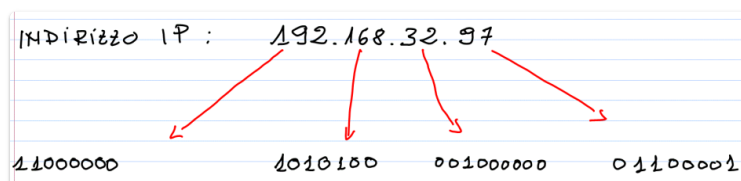
- Per le comunicazioni interne si utilizzano indirizzi privati, che hanno valenza cioè solo all'interno della rete
- Quando un host della rete decide di aprire una comunicazione con un dispositivo esterno alla rete sorgente, si mostrerà con l'indirizzo pubblico attribuito alla rete

- 😞: diventa più complicato in fase di ricezione capire l'esatto dispositivo che ha fatto una richiesta, perché appare solo l'indirizzo della rete
- 😊: meno spreco di indirizzi - una rete con  $n$  dispositivi ha bisogno solo di un unico indirizzo per interfacciarsi
- 

## NOTAZIONE DEGLI INDIRIZZI: DECIMALE PUNTATA

- 4 numeri (veicolati con il loro valore in binario) separati da un punto

Esempio:



Fissata la classe dell'indirizzo (esempio classe B), si assegna la possibilità di inserire gli indicatori di HOST ID e SUBNET ID nel campo originariamente previsto solo per le informazioni di HOST ID

- Prendendo la classe B come esempio abbiamo 2 byte da gestire per l'HOST ID. Con questa nuova modalità i medesimi 2 byte possono essere gestiti per HOST ID e SUBNET ID
- Nota: i router leggono solo **i primi due campi** che rappresentano la rete (NET ID)
  - I 2 byte suddetti della classe B hanno una **importanza locale** (della sottorete in considerazione, perché verso l'esterno come detto si interfacciano con un unico indirizzo)

## MASCHERA DI RETE (SUBNET MASK)

- Serve per definire i "confini" dei sottoindirizzi di una rete (NET ID)
- Vale 1 nelle posizioni di SUBNET ID
- Vale 0 nelle posizioni di HOST ID

L'esempio più comune è il seguente (classe B):

255.255.255.0  $\longleftrightarrow$  11111111.11111111.11111111.00000000  
NET ID SUBNET ID HOST ID

In questo caso:

- I primi 2 byte sono dedicati a NET ID (per com'è definita la classe B)
- Il terzo byte è relativo al SUBNET ID
- Si lascia libero (solo) l'ultimo byte per indirizzare gli host locali (HOST ID)

Per poter *estrarre* le informazioni necessarie per gestire l'operazione d'inoltro ai router, per ogni rete raggiungibile abbiamo in memoria le maschere di rete associate

La **procedura** è la seguente:

- AND logico per individuare le informazioni circa la maschera di rete (se trovo 0 invece vuol dire che ho trovato le informazioni dell'host id)

Esempio:

Ad un campus universitario è stato assegnato un indirizzo IP di classe B seguente:

150.10.0.0

Avrà maschera di sottorete, essendo di classe B:

255.255.255.0  $\longleftrightarrow$   $\underbrace{11111111.11111111}_{\text{NET ID}}.\underbrace{11111111}_{\text{SUBNET ID}}.\underbrace{00000000}_{\text{HOST ID}}$

## INDIRIZZI SENZA CLASSI: COME VIENE GESTITO IL SUBNETTING

Vediamo con un esempio: si vogliono gestire 1000 indirizzi ( $10^3$ )

- Si devono assegnare rispettando la regola di avere un numero finale espresso come potenza di intera di due: avendo 1000 indirizzi, avremo come riferimento 1024, ovvero  $2^{10}$  (quindi 10 bit per l'HOST ID e i restanti 22 per la maschera)

In notazione senza classi:

w.y.z.k/n , nel nostro caso: w.y.z.k/22

## PROCEDURA DI INDIVIDUAZIONE DELLA SOTTORETE

Esempio:

- A una sottorete è stato allocato un blocco di 1024 indirizzi, così localizzato:

da 200.30.0.0 a 200.30.3.255

*Trovare la maschera di sottorete*

- Traduciamo in binario

200.30.0.0  $\longleftrightarrow$  11001000.00011110.00000000.00000000  
 200.30.3.255  $\longleftrightarrow$  11001000.00011110.00000011.11111111

Facendo l'AND logico, si ottiene la maschera di sottorete:

$\underbrace{255.255.}_{\text{NET ID}}\underbrace{252.0}_{\text{sottorete} \rightarrow} \longleftrightarrow \underbrace{11111111.11111111}_{\text{NET ID}}.\underbrace{111111}_{\text{SUBNET ID}}\underbrace{00.00000000}_{\text{HOST ID}}$

## PROCEDURA DI ROUTING = Interrogazione a un database locale interno al Router

Ogni router contiene in memoria (database):

- NET ID di ciascuna rete
- Una copia della maschera degli indirizzi delle reti

Ciascun router quindi che riceve un pacchetto:

- Legge l'IP di destinazione
- Esegue AND logico tra l'indirizzo e le tutte le maschere nel database del router corrispondenti (cfr. esempio successivo)
- In caso di "match" ✓ si identifica la porta del router a cui è associato e il relativo indirizzo destinazione associato al router successivo posizionato meglio secondo l'algoritmo di routing utilizzato (da sovrascrivere nel campo *header* del datagramma)
- Si reitera fino alla destinazione
- (se non c'è corrispondenze: si manda tutto a un router di default finché TTL  $\neq$  0)

### Nota:

Il matching può portare ad ambiguità: accade cioè che il match con un host dia luogo a corrispondenze multiple

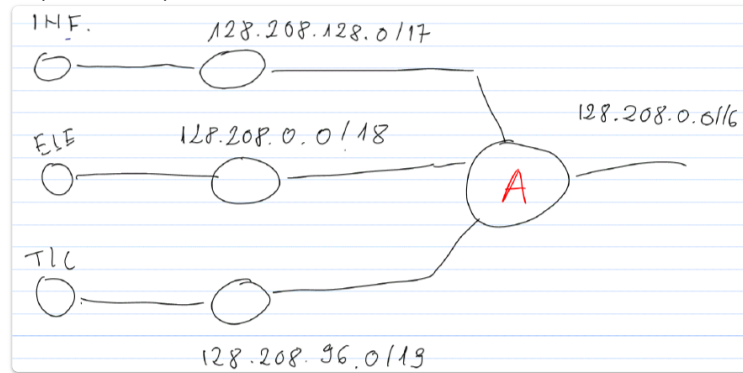
- In tali casi si sceglie la corrispondenza con la maschera di sottorete più lunga (con più 1): statisticamente infatti è più probabile che sia quello giusto

## ALTRO ESEMPIO

Si assegnato i seguenti blocchi:

INFORMATICA	10000000	11010000	1xxxxxx.xxxxxxx
ELETTRONICA	10000000	11010000	00xxxxxx.xxxxxxx
TELECOMUNICAZIONI	10000000	11010000	011xxxxxx.xxxxxxx

La struttura della rete sarà la seguente (in decimale):



Quando arriva un pacchetto sul router A, esso deve capire qual è la destinazione.

Esempio:

128.208.2.151

- Si effettua l'AND logico tra l'indirizzo suddetto e tutte le maschere delle sottoreti connesse/disponibili sul router

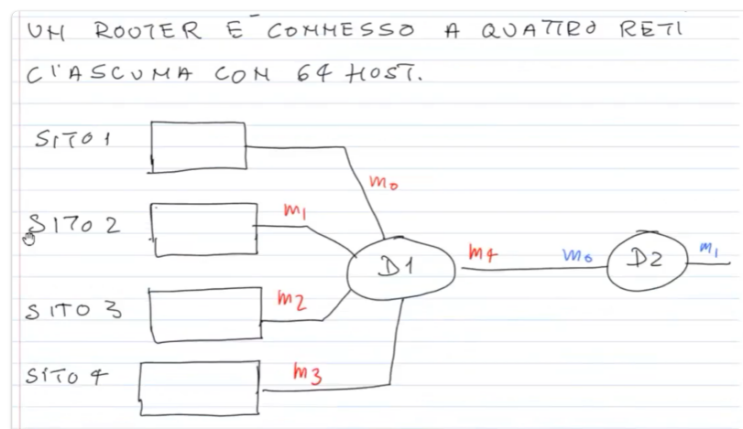
CASO 1:

VERIFICA SE È DESTINATO AD INFORMATICA.  
 MASCHERA RETE : 255.255.128.0  
 ESEGUO AND LOGICO.  
 IND. IP : 10000000 11010000 00000010 10011111  
 MASK : 11111111 11111111 10000000 00000000  
 RISULTATO:  
 10000000 11010000 00000000 00000000  
 128.208.0.0  
 NON È INDIRIZZATO A INFORMATICA.

CASO 2:

CONSIDERIAMO ELETTRONICA.  
 10000000 11010000 00000000 00000000  
 RISULTATO  
 10000000 11010000 00000000 00000000  
 128.208.0.0  
**CORRISPONDENZA!**  
 DESTINAZIONE ELETTRONICA!

ESEMPIO: DETTAGLIO SU TABELLE DI ROUTING



QUESTE TABELLE DEVONO INCLUDERE 4 INFORMAZIONI:

- LA MASCHERA DI RETE
- ID DELLA RETE
- IL NUMERO DELL'INTERFACCIA DI USCITA
- ID DEL ROUTER SUCCESSIVO.

Le relative tabelle sono:

IND. RETE / MASK	ROUTER NEXT	INTERFACCIA
140.24.7.0/26	—	m <sub>6</sub>
140.24.7.64/26	—	m <sub>1</sub>
140.24.7.128/26	—	m <sub>2</sub>
140.24.7.192/26	—	m <sub>3</sub>
00600	ID D1 D2	m <sub>4</sub>

Da leggere come:

- Se arriva [INDIRIZZO/MASK]
- Allora inoltra a [PROSSIMO ROUTER]
- Sulla interfaccia [m<sub>i</sub>]

In caso di match: fine (primi quattro casi)

Si noti come solo m<sub>4</sub> si interfaccia a un nuovo router. Cioè se arriva una richiesta da un indirizzo diverso da quelli descritti precedentemente, allora manda tutto sul router di default D2 attraverso l'interfaccia m<sub>4</sub>

La tabella di D2 invece:

LA TABELLA DI INOLTRO DI D2 PUÒ ESSERE SEMPLIFICATA ASSOCIANDO ALL'INTERFACCIA m<sub>4</sub> DI D2 UN QUASI DATAGRAMMA CON ID COMPRESO DA 140.24.7.0 a 140.24.7.255

- In genere si associa l'indirizzo con la sotto maschera più lunga

## DHCP

*Dynamic Host Configuration Protocol*

L'host attraverso questo protocollo acquisisce in maniera automatica (plug-in-play):

- L'indirizzo IP
- La Maschera di Rete (sottorete)
- Gateway (indirizzo router)

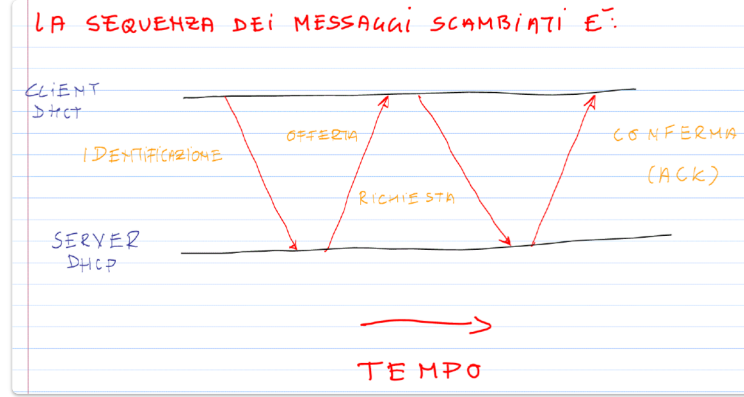
L'indirizzo è dinamico perché può cambiare col tempo (e viene concesso solo quando c'è la necessità)

- Lo scopo è quello di aumentare l'utilizzo degli indirizzi IP: solo chi è attivo sulla rete possiede un indirizzo (non conviene vincolare un indirizzo IP esclusivamente per un dispositivo considerando che questo non sarà sempre collegato sulla rete)
  - In questo modo in generale (non contemporaneamente): # utenti gestiti > # indirizzi

Serve un apposito **server DHCP** associato alla rete

- Il DHCP quindi è un protocollo == *client-server* == in cui il *client* è il dispositivo che richiede di connettersi





- Il client si fa identificare
- Il server (o anche più di uno) risponde (rispondono) con una o più offerte (di indirizzo, banda, durata etc...)
- Il client quindi sceglie una delle offerte e lo comunica al server
- Il server conferma (se l'indirizzo è ancora disponibile) con un acknowledgement

Vediamo come sono fatti i singoli messaggi scambiati:

#### MESSAGGIO DI IDENTIFICAZIONE (DHCP DISCOVERY)

Un dispositivo nuovo nella rete:

- Non ha indirizzo IP
- Non sa dove si trova il server DHCP

Per tali ragioni, il messaggio viene inviato in **broadcast** 255.255.255.255, con indirizzo sorgente 0.0.0.0

- Il messaggio ha in allegato un *identificatore* che identifica in maniera univoca la richiesta (e quindi il client che l'ha fatta)
  - Il campo identificatore della richiesta è detto *transaction-id* ed è lungo 4byte
    - La prima volta viene scelto in modo casuale
- Il datagramma viene inviato utilizzando UDP con porte sorgente-destinazione: 68-67
  - La porta è una entità logica (etichetta) che ci permette di separare i datagrammi in relazione al servizio a cui si riferiscono (utile soprattutto ai livelli superiori)

#### MESSAGGIO DI OFFERTA (DHCP OFFER)

Viene generato in risposta a una richiesta di accesso.

Contiene le seguenti informazioni:

- *transaction-id* sorgente
- L'indirizzo proposto
- La maschera di rete
- Tempo di validità dell'offerta

Il messaggio viene mandato in broadcast nuovamente con UDP (porte 67-68)

#### MESSAGGIO DI RICHIESTA (DHCP REQUEST)

Il client sceglie l'offerta più conveniente e comunica:

- Al server scelto che ha accettato l'offerta
- Agli eventuali altri server che ha accettato una offerta diversa

Questo lo effettua inviando il messaggio specificando gli attributi univoci dell'offerente scelto. Viene mandato in broadcast. Chi riconosce nel messaggio l'offerta fatta comprende di essere stato selezionato (gli altri sanno di essere esclusi)

- Nota: per fini commerciali e di sicurezza, non si utilizza l'IP assegnato come sorgente

#### MESSAGGIO DI CONFERMA (DHCP ACK)

Il server DHCP informa il client che l'indirizzo scelto è ancora disponibile e può cominciare a utilizzarlo

**Bonus:** se l'offerta per qualche motivo è saltata (ad esempio se l'indirizzo IP proposto non è più disponibile), il server lo comunica inviando un messaggio detto DHCP-HACK in broadcast

## NAT

Network Address Translation (Traduzione Indirizzi Di Rete)

Un ISP in generale deve gestire un numero di utenti ben superiore al numero di indirizzi che ha a disposizione. Per tale ragione si utilizza il NAT che permette di assegnare un unico indirizzo IP di "interfacciamento verso l'esterno" per una intera rete  
In questo modo:

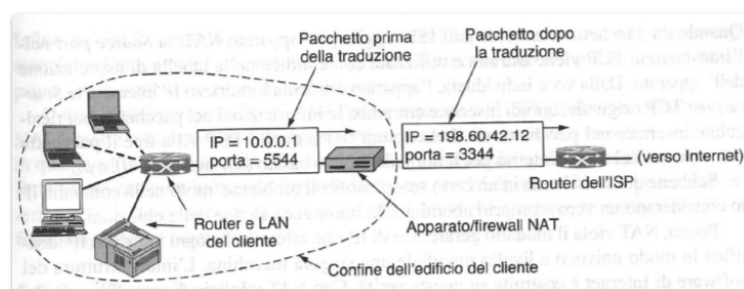
- All'interno della rete si continuano ad utilizzare gli indirizzi privati
- Quando si vuole interfacciarsi con l'esterno lo si fa attraverso un unico indirizzo (gateway) assegnato dall'ISP

Gli **indirizzi privati** sono divisi come segue:

10.0.0.0	10.255.255.253/8
172.16.0.0	172.31.255.253/12
192.168.0.0	192.168.255.253/16

Il NAT permette la traduzione grazie a una apposita **tabella di matching**, in cui in essa si associa ciascun indirizzo privato della rete con il relativo indirizzo pubblico

## FUNZIONAMENTO



- Un dispositivo manda il suo datagramma verso il router NAT
- Il server NAT associa l'indirizzo (in questo caso appartenente alla prima famiglia) con l'indirizzo pubblico e traduce anche il numero di porta, mettendone uno associato in maniera univoca al client che ha richiesto l'uscita (questo risolve quelle ambiguità in cui due host della stessa rete richiedono lo stesso servizio: in quel caso se il NAT non assegnasse una porta "fittizia" e non la salvasse nella sua tabella, poi dopo non saprebbe in fase di ricezione a chi indirizzare i successivi datagrammi in risposta)

- ☹️: il processo di traduzione richiede un certo tempo  $t$  di esecuzione
- ☹️: il matching in tabella avviene in fase d'invio verso l'esterno, ma se prima arriva una richiesta dall'esterno, il NAT poi non sa a chi mandare il datagramma di preciso perché non ha salvato nulla sul database locale
- 😊: risolto con il NAT trasversale (server specifico più complesso)
- 😊: specificando le porte relative al servizio che si vuole attuare è più facile individuare l'host di destinazione della rete privata
- ☹️: il server NAT rende la rete **centralizzata**: a causa di questo, se si dovesse rompere per qualche motivo, l'intera rete non potrebbe più interfacciarsi verso l'esterno
- ☹️: problemi quindi di affidabilità e sicurezza (pensiamo a un hacker che butta giù il server)

## IPv6

Introdotta a causa della carenza d'indirizzi a tutte le nuove necessità di poter gestire i servizi in modo più efficiente. Permette anche di implementare alcuni servizi specifici in maniera più concreta rispetto a IPv4 (ad esempio la sicurezza, che viene inserita nel campo dati di IPv6, invece che nel campo Options di IPv4)

- 😊: Meno campi da interpretare (e quindi meno controlli, meglio... più veloce la lettura nei routers), ma stesse funzionalità

Version (4 bit)	Traffic Class (8 bit)	Flow Label (20 bit)	
Payload Length (16 bit)		Next Header (8 bit)	Hop Limit (8 bit)
Source Address (128 bit)			
Destination Address (128 bit)			

## DESCRIZIONE DEI CAMPI

- **Version** (4 bit): specifica la versione del protocollo

- **Traffic Class** (8 bit): simile al campo TOS di IPv4 → specifica il tipo di traffico/servizi che si devono gestire
- **Flow Label** (20 bit): [etichetta di flusso] - serve per distinguere i flussi di datagrammi quando questi sfruttano lo stesso servizio. Consente la piena gestione di questi pacchetti in fase di inoltramento, con un criterio definito a seconda del caso. Questa filosofia riguardo la modalità di inoltramento ricorda molto il *connection oriented*
  - 🗉: questa cosa va un po' contro l'idea di IP protocollo che è nato connection less
- **Payload Length** (16 bit): lunghezza della parte dati del datagramma (dimensione totale meno header) -  $2^{16} - 1 = 65535$  possibili combinazioni. *Nota*: l'header ha lunghezza fissa di 40 byte
- **Next Header** (8 bit): specifica l'incapsulamento della parte dati successiva e può riferirsi a protocolli di livello superiore (trasporto), come il TCP {fa tutto parte del payload}
- **Hop Limit** (8 bit): sostituisce il TTL dell'IPv4

Nota: Manca il campo *checksum* che proteggeva da errori le informazioni della testata

- 😊: Elaborazione più veloce (non devo ricalcolarlo tutte le volte che si cambia la testata)
- 😞: Si suppone però che in ogni trasferimento non si abbiano errori o mis-match
  - 📶: rischio apparentemente alto, ma si ipotizza al contempo che si utilizzino mezzi di trasmissione di qualità superiore

### 🔍 Come si comporta IPv6 con reti non omogenee e quindi con la frammentazione?

Notiamo che non ci sono campi dedicati nel datagramma all'operazione di frammentazione, quindi ci chiediamo ad esempio *come si realizza la compatibilità di formato su tutto il percorso end-to-end che coinvolge reti non omogenee (cioè con formato di trasporto diverso)?*

*In IPv4 la frammentazione avviene all'occorrenza, cioè con modalità link-to-link: quando i datagrammi sono troppo grandi il router li frammenta (il mittente non è a conoscenza di ciò)*

*In IPv6 la frammentazione avviene in modalità end-to-end: prima dello scambio di dati, si scopre la dimensione massima di trasferimento in tutto il percorso attraverso un protocollo detto MTU discovery. In questo modo la sorgente individua il valore più piccolo e implementa già lei (in partenza) la frammentazione che poi sarà gestita in fase di ricezione*

### 📅 Novità IPv6

Riassumendo, le novità di IPv6 rispetto a IPv4 sono:

- Assenza del campo checksum
- Frammentazione con filosofia end-to-end invece di link-to-link
- Indirizzamento più ampio (vedi dopo)

## INDIRIZZAMENTO

Abbiamo a disposizione un campo indirizzo esteso: 16 byte (contro i 4 byte di IPv4)

Il tipo di indirizzamento prevede 4 modalità:

- Multicast (da uno verso molti: un indirizzo verso tanti destinatari - se tutti i destinatari allora broadcast)
- Unicast (un indirizzo verso un destinatario)
- Anycast (indirizzo IP condiviso, cioè con più corrispondenze all'interno della rete)
  - Quindi, se una interfaccia può avere più indirizzi IPv6, il NAT non ha più utilità (se necessario basta attribuire un nuovo indirizzo, non ci sono problemi di carenza)

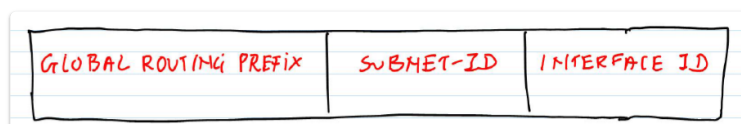
Gli indirizzi sono espressi in esadecimale, ad esempio:

2001:0db8:85a3:0000:0000:8a2e:0370:7334

### SINTASSI UNICAST IPV6

Si identifica in modalità univoca una sola interfaccia di rete

Ha una sintassi **gerarchica**



- Global Routing Prefix: identifica una area molto estesa - 48 bit
- Subnet-Id: identifica una rete di più piccole dimensioni - 16 bit
- Interface-Id: detto anche Host-ID, identifica un singolo dispositivo appartenente a una subnet - 64 bit

Esistono anche indirizzi unicast specifici:

## INDIRIZZI LINK LOCAL

Utilizzati solo su base link, cioè tra dispositivi *direttamente connessi*

## INDIRIZZI SITE LOCAL

Utilizzato come indirizzi privati, ovvero per connessioni tra dispositivi di una stessa rete/sito (non sono accessibili dall'esterno)

## CONFRONTO HEADER IPv4 vs HEADER IPv6

- Campo IHL non presente in IPv6: le testate hanno tutte lunghezza uguali, cioè 20 byte (specificava la lunghezza dell'header). Eventuali estensioni di dimensione sono eseguite nella parte dati (payload)
- Campo Protocol rimosso in IPv6 perché il protocollo di livello TCP è specificato nel campo *next header*, situato nella parte dati (payload)
- Campi legati alla frammentazione assenti in IPv6 perché tale operazione è effettuata in modalità end-to-end (non serve quindi trasferire le informazioni lungo il tragitto - viene già fissato tutto in precedenza con MTU discovery)
- Checksum non presente (si auspica di star utilizzando mezzi di trasmissione di qualità con bassa probabilità di errore)

😊: IPv6 più veloce e flessibile (e quindi più adatto alle nuove esigenze) - durante il tragitto i datagrammi passano molto più velocemente attraverso ai routers perché vengono effettuati meno controlli

😊: ha risolto definitivamente (?) la carenza di indirizzi

## PASSAGGIO IPv4 → IPv6

Cambiare standard non è una cosa semplice.

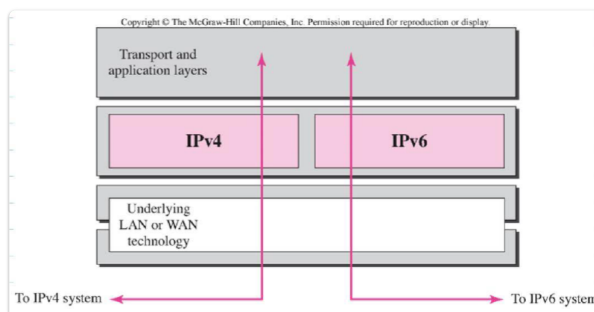
Bisogna perciò adattarsi al caso.

Risulta di fondamentale importanza rendere *retrocompatibili* i sistemi IPv6 con i precedenti IPv4

Sono state sviluppate due tecniche a riguardo:

- DUAL STACK (doppia pila)
- TUNNELING

## DUAL STACK



Struttura che implementa entrambe le versioni a livello rete: si associa a un indirizzo IPv4 un indirizzo IPv6 (eseguendo una conversione grazie all'utilizzo di un router/gateway), risalendo la pila fino allo strato TCP con un protocollo e poi riscendendo con un altro

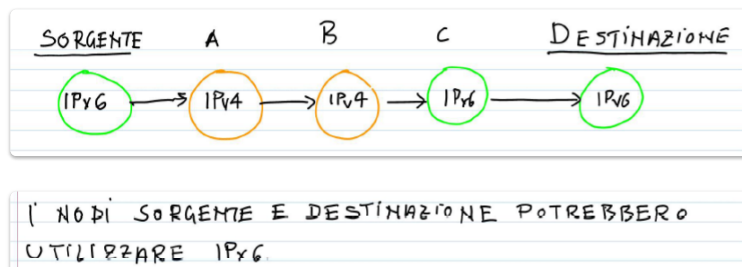
😊: facile da implementare

😞: non risolve il problema della carenza di indirizzi IPv4

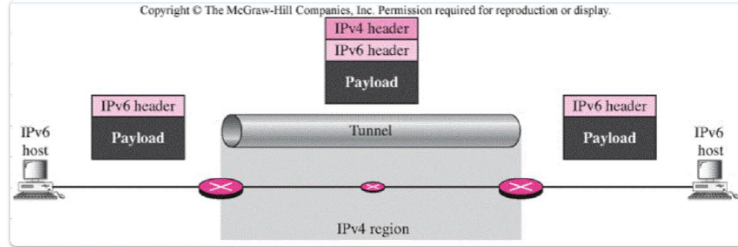
😞: conversione al contrario (IPv6 → IPv4) si perdono funzionalità: abbiamo meno informazioni in testata (invece ciò che si perde in IPv4 → IPv6 viene integrato con servizi più evoluti). Questo porta ad esempio l'impossibilità di gestire alcune parti/funzionalità (come il *flow label*, ovvero la gestione dei flussi: no possibilità di classificare i flussi a seconda delle esigenze di servizio)

😞: si deve utilizzare un router/gateway (costi + prestazioni elevate)

### ESEMPIO:

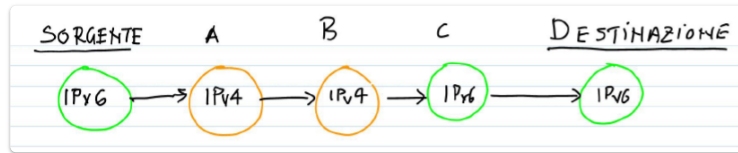


## TUNNELING



Si incapsula IPv6 dentro la parte dati di IPv4 per viaggiare su reti non compatibili con IPv6. Una volta che si giunge nella "zona compatibile", si estrae la parte IPv6 e si sfrutta quella come protocollo di default

Nel caso precedente:



Nelle parti gialle di usa IPv4 e quindi si incapsula con il tunnelling, mentre in quelle verdi si può usare direttamente IPv6 (dopo aver estratto l'informazione - "un-tunnelling")