# Livello di Rete

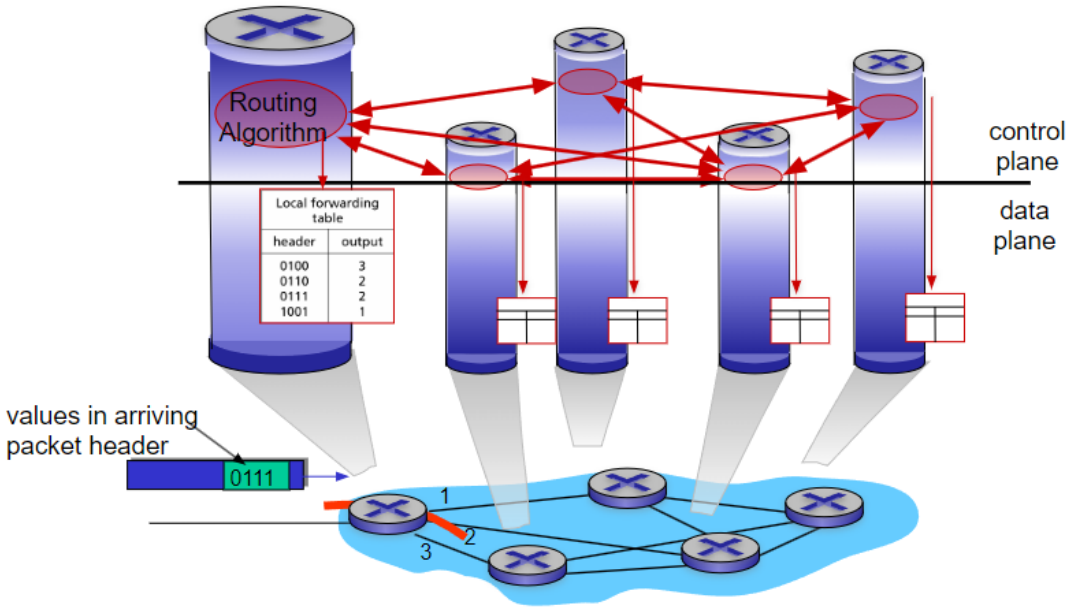
Ci troviamo la livello di rete, ovvero come instradare i pacchetti, abbiamo la **tabella di instradamento** la quale **associa l'ip con una porta di uscita** del router. Ogni router ne ha una.

Abbiamo una trasformazione dei dati che è la seguente:

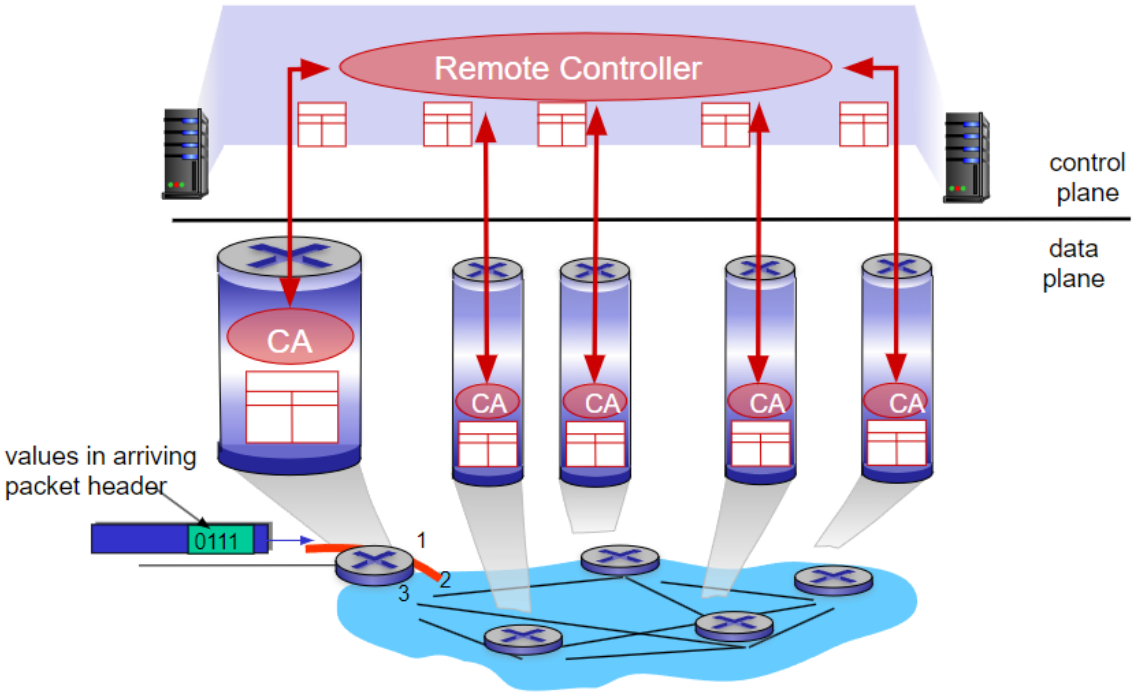
- **Livello applicazione**: i dati sono formattati secondo il protocollo del tipo scelto (Query DNS, SMTP, GET, POST ecc...) e i dati sono chiamati "dati".
- **Livello trasporto**: i dati sono incapsulati grazie a due protocolli ovvero TCP (diventano segmenti) altrimenti UDP (diventano datagrammi)
- **Livello di rete**: a questo punto i nostri dati vengono nuovamente incapsulati e diventano effettivamente dei pacchetti.

Funzione di **forwarding** ovvero cosa fa il router quando arriva un pacchetto quale uscita deve prendere. Il **routing** è il percorso che deve fare il pacchetto vedendo tutto il tragitto, sono due cose diverse compiti del router.

Il **data plane** è **l'esecutore** ovvero il circuito che applica la tabella, il passo prima è scrivere le tabelle, chi lo fa è il **cervello** ovvero il **control plane** ovvero un **algoritmo di routing** il control plane crea le tabelle che molte volte sono sbagliate, il router parlano con altri router vicini di massimo un salto. Un router quando passa l'informazione avviene un passa parola.




Proprio per questo si ha un **remote controller** il quale decide come applicare le regole nelle tabelle di instradamento e avendo la vista sull'intera rete avremo una maggiore efficienza.

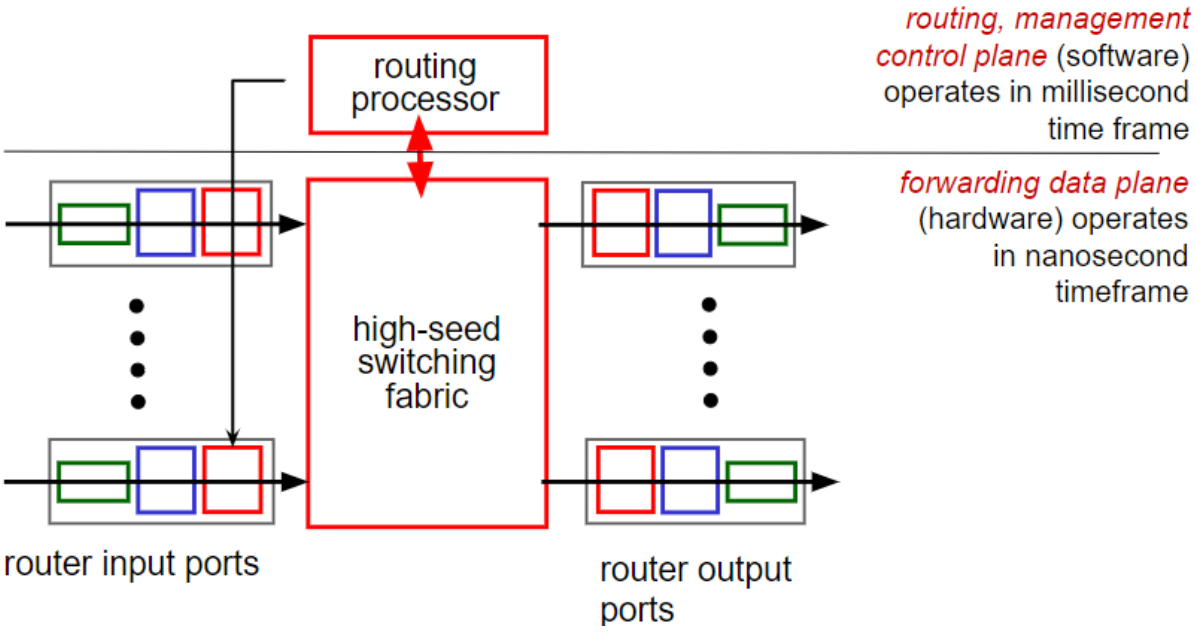


Il **data plane** in questo caso è il **control agent**, il **remote controller** è il **control plane**. Questo remote controller è situato dentro l'ISP oppure dentro una macchina locale.

Come possiamo vedere il concetto di data plane e control plane è in base a chi decide la logica di instradamento, non ci sono componenti fisse in ogni schema.

# Cosa c'è dentro un router?

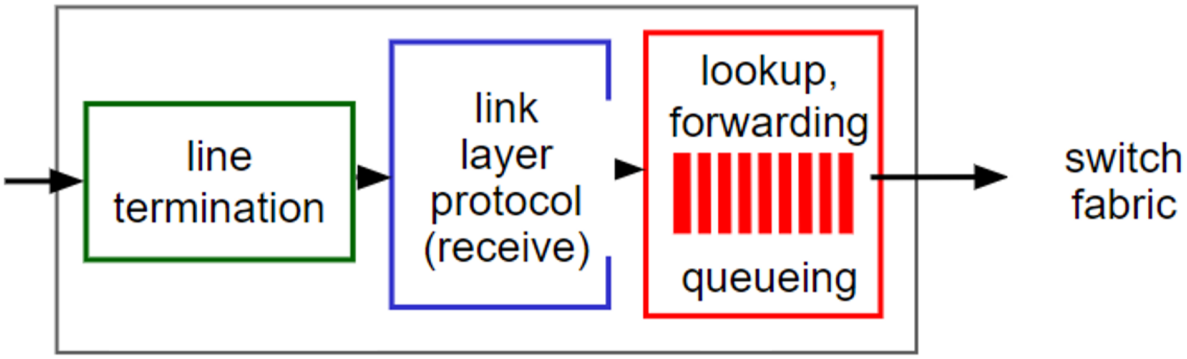
Scopriamo con una vista generica cosa c'è dentro un router, attraverso le sue componenti:




Un router è composto da **porte di entrata** e **porte di uscita**, al suo interno c'è un **processore** che serve per capire il pacchetto in arrivo su che porta deve uscire. Abbiamo quindi due parti la parte di controllo (**control plane**) che ragiona in millisecondi e svolge il **routing** e la parte fisica (**data plane**) che ragiona in nanosecondi che svolge il **forwarding**, quindi ci sarà sempre un minimo di ritardo proprio perchè bisogna computare i dati. La porta di entrata è composta da tre layer principali:

- **Terminazione di linea**: la parte che si occupa di ricevere i bit ovvero il livello fisico

- **Livello collegamento**: la porta Ethernet effettiva dove si rimuove l'header del livello rete per capire attraverso la computazione del processore, dove inviare il pacchetto
- **Buffer di instradamento:** andiamo a vedere l'ip di destinazione e abbiamo una match action ovvero quale sarà l'uscita, qui avviene quindi la scelta di dove inviare il pacchetto. Quì è dove lavora il processore del router per calcolare l'instradamento.
- **Switch frabric**: ovvero il circuito che è stato immesso dentro il router su cui viaggiano i bit




Abbiamo anche una **tabella di instradamento** per capire dove indirizzare il pacchetto in arrivo, come possiamo vedere sono associate il destination address con quale porta di output:

| Destination Address Range | Link Interface |
|---|---|
| da: 11001000 00010111 00010000 00000000<br>a: 11001000 00010111 00010111 11111111 | 0 |
| da: 11001000 00010111 00011000 00000000<br>a: 11001000 00010111 00011000 11111111 | 1 |
| altrimenti | 2 |

Questa tabella quindi crea un instradamento con i range di ip, quando un ip non è in un range c'è sempre una porta che si deve occupare di accogliere questi

indirizzi non in range, in questo caso è la port numero 2. Questo però è molto dispendioso quindi si è creata una tecnica chiamata **longest prefix match** la quale matcha gli indirizzi con il prefisso più lungo:

| Destination Address Range | Link Interface |
|---|---|
| 11001000 00010111 00010*** ******** | 0 |
| 11001000 00010111 00011000 ******** | 1 |
| 11001000 00010111 00011*** ******** | 2 |
| altrimenti | 3 |

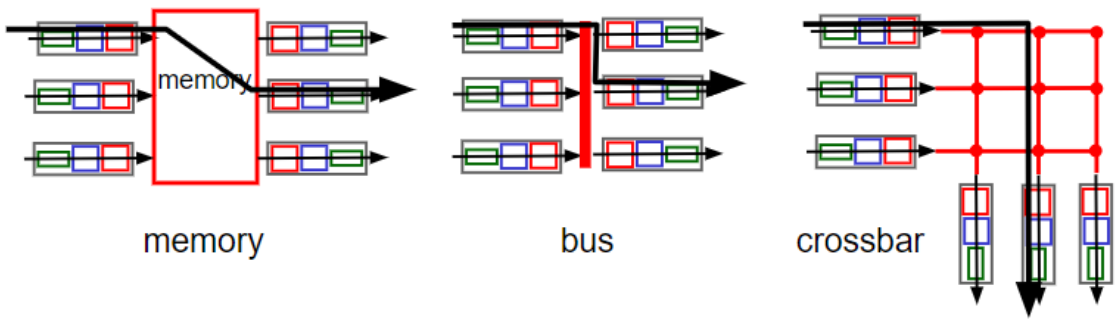
Quindi tutti gli indirizzi che hanno un match iniziali in quella sezione verranno inviati in quella porta senza dover creare dei range.

Ad esempio:

- **11001000 00010111 00010**110 10100001 → la parte evidenziata crea un match con l'interfaccia 0
- **11001000 00010111 00011000** 10101010 → la parte evidenziata indica un match maggiore con la porta 1 invece che con la 2 anche se sono uguali, la 1 è un match più lungo

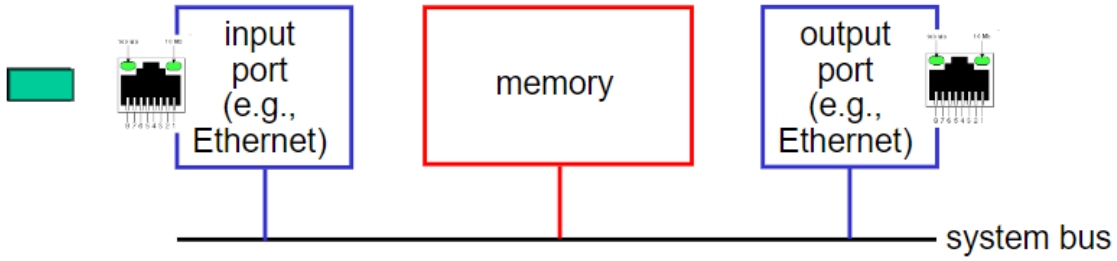
## Tipologie di circuiti del router (switching frabric)

I trasferimenti di pacchetti da un input buffer a un output buffer, con quale rate avviene e ci sono tre tipologie di switching.




## Switching sulla memoria

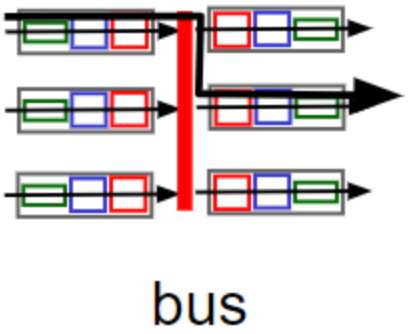
Questo avveniva con la prima generazione di router, nella quale i computer avevano funzione di routing, il pacchetto è copiato dentro la memoria di sistema.




Il rate di invio era molto lento perchè c'era molta lavorazione dietro l'invio del pacchetto. Questo si cita solo per funzioni storiche.

## Switching sul bus

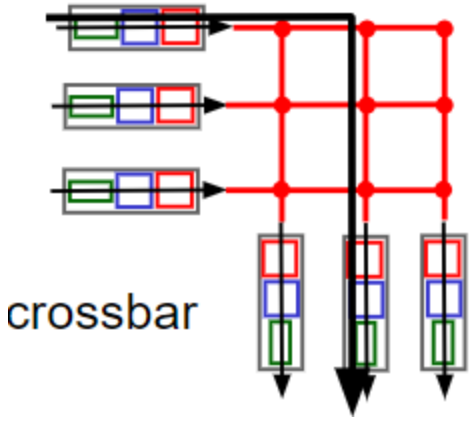
Il pacchetto viene preso da una porta di input e questo pacchetto viene propagato su tutta la linea, solo la porta con la stessa etichetta potrà uscire da quella porta.




Nel caso l'etichetta non corrisponda il pacchetto viene scartato.

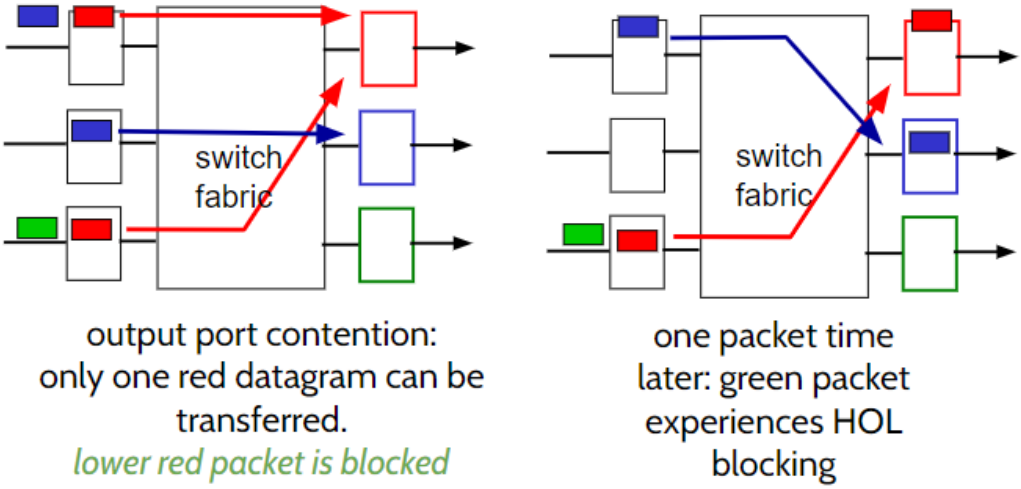
## Switching con una griglia

Ogni porta di ingresso è connessa con la sua porta di uscita, grazie a questa soluzione non abbiamo un intasamento di una linea dato che le linee sono dedicate.



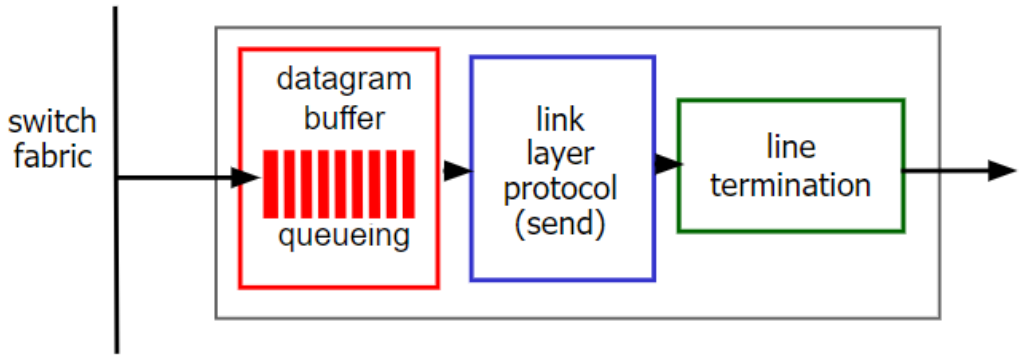

# Code nelle porte di ingresso

Essendo che c'è un minimo di lavorazione dentro il router è normale che avvenga un po' di congestione e quindi di coda all'interno delle porte di input. Code troppo lunghe possono portare a una perdita di pacchetti chiamato **head of the line blocking**, ovvero un pacchetto in cima alla coda che blocca altri pacchetti dal loro instradamento.

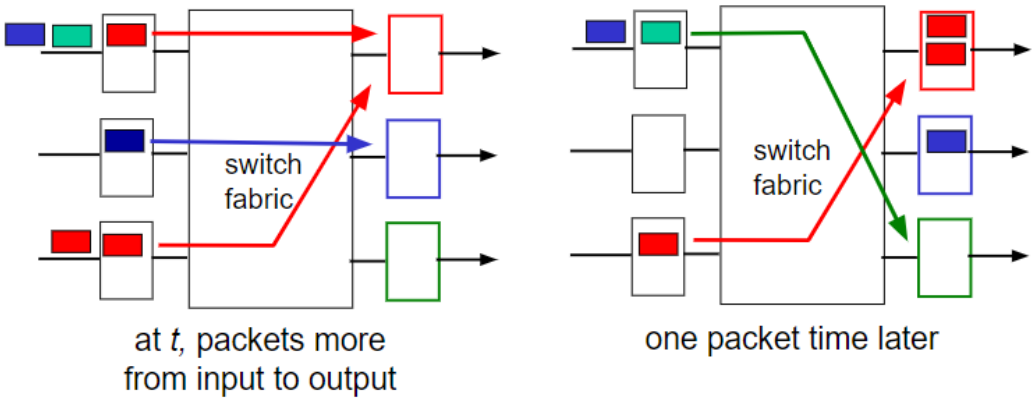



Come in questo esempio il pacchetto rosso sta bloccando il verde, anche se l'uscita del verde è libera il pacchetto non può passare perchè il rosso sta bloccando (**head of the line blocking**).

# Code nelle porte di uscita




Abbiamo un duplice problema sulle porte di uscita: il **buffering** e lo **scheduling**, il primo è un problema di quando abbiamo una congestione dove i pacchetti possono essere persi e poi abbiamo lo scheduling ovvero poter mettere i pacchetti in input nella giusta uscita.




Come in questo caso che arriva un pacchetto rosso, ma noi abbiamo già il buffer rosso pieno quindi lo scheduling deve decidere se è più importante (?)

$$Buffering\ Timer = \frac{RTT * C}{\sqrt{N}}$$

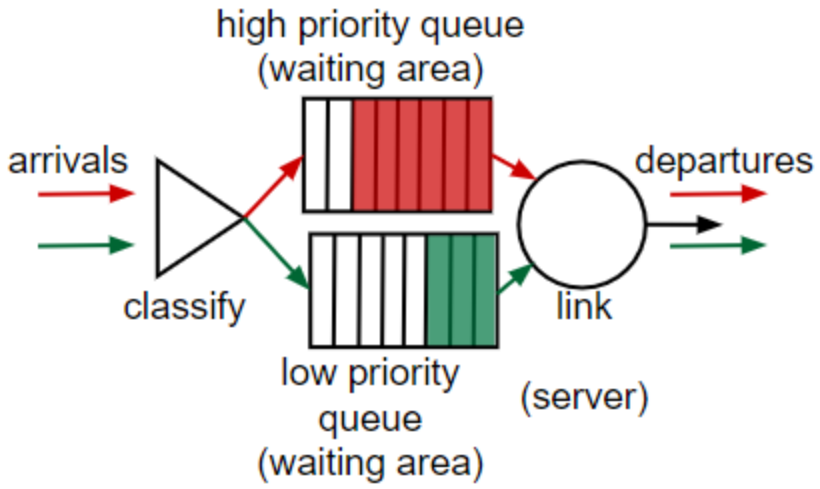
Dobbiamo capire quali pacchetti sono da scartare e quali invece sono più importanti, ci sono diversi algoritmi come ad esempio FIFO dove l'ultimo arrivato viene scartato, ma ci sono molti altri modi:

- **Tail drop:** scarta l'ultimo pacchetto arrivato

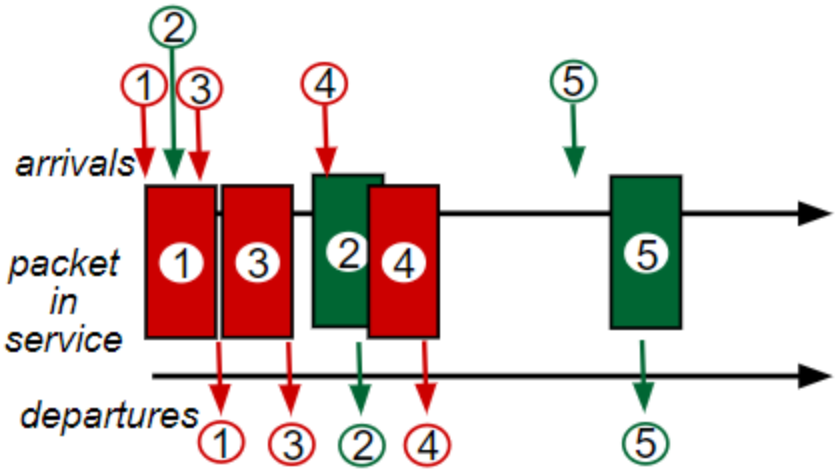
- **Priority:** scarta il pacchetto senza priorità (viene impostato un campo option, non c'entra più il TCP perchè siamo a livello di rete)
- **Random:** scarta un pacchetto in maniera randomica.

## Scheduling con priorità

Questo tipo di scheduling invia il pacchetto con **priorità più elevata ai buffer**, ci sono **diverse classi** di priorità. La **classe** può dipendere dal campo **header**: ip di destinazione o sorgente ad esempio possono influire.



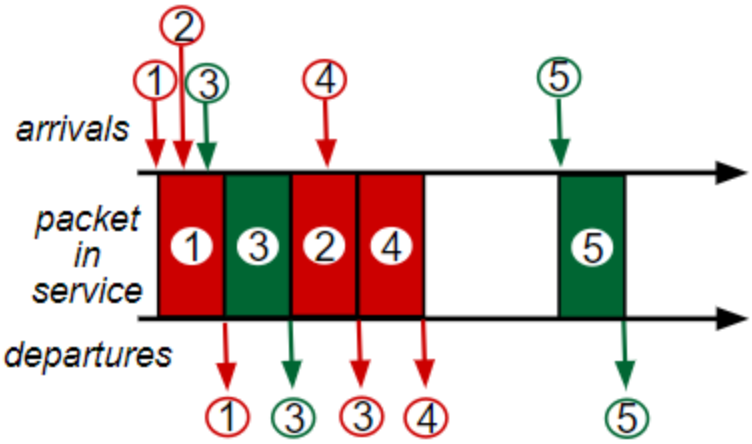

In questo caso vediamo cosa succede con un'arrivo sequenziale ma con priorità sui pacchetti:



Come vediamo il pacchetto 2 arriva prima del 3 ma essendo che ha priorità minore il 3 viene gestito prima. Questo potrebbe creare starvation nella rete ma in internet questa cosa non accade.

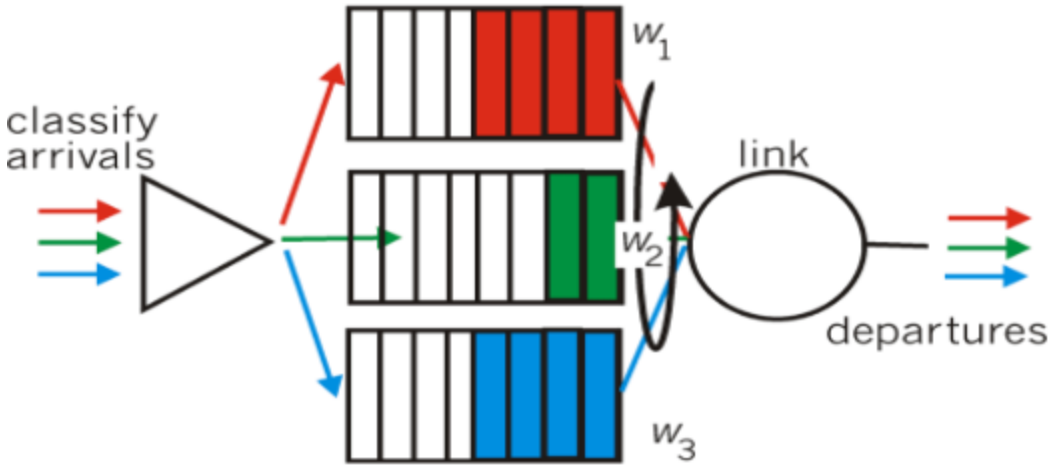
## Scheduling Round Robin

Funziona a classi multiple come quelli con priorità, ciclicamente legge le code di diverse classi inviando un pacchetto completo da quest'ultima. Non esiste quindi una priorità di invio, ma serve per creare una classificazione.

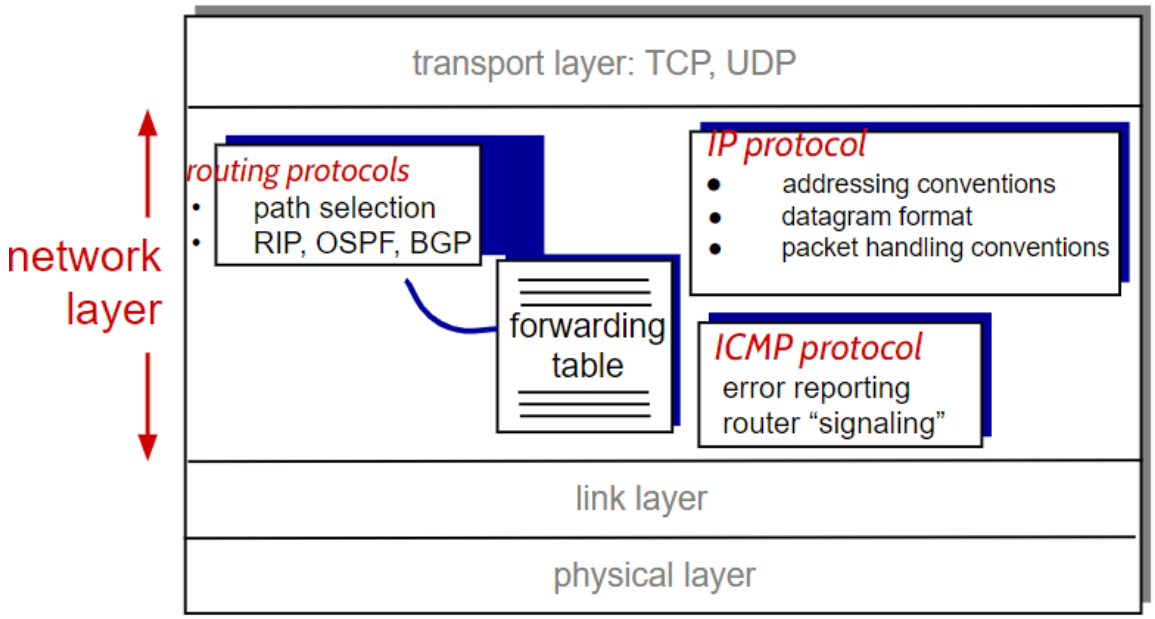



## Scheduling con pesi (Weight Fair Queue)

La WFQ unisce il Round Robin con lo scheduling con priorità, a ogni coda di pacchetti abbiamo un peso diverso e viene calcolato un numero che a seconda del peso prende un pacchetto da quello coda, si basa su probabilità.

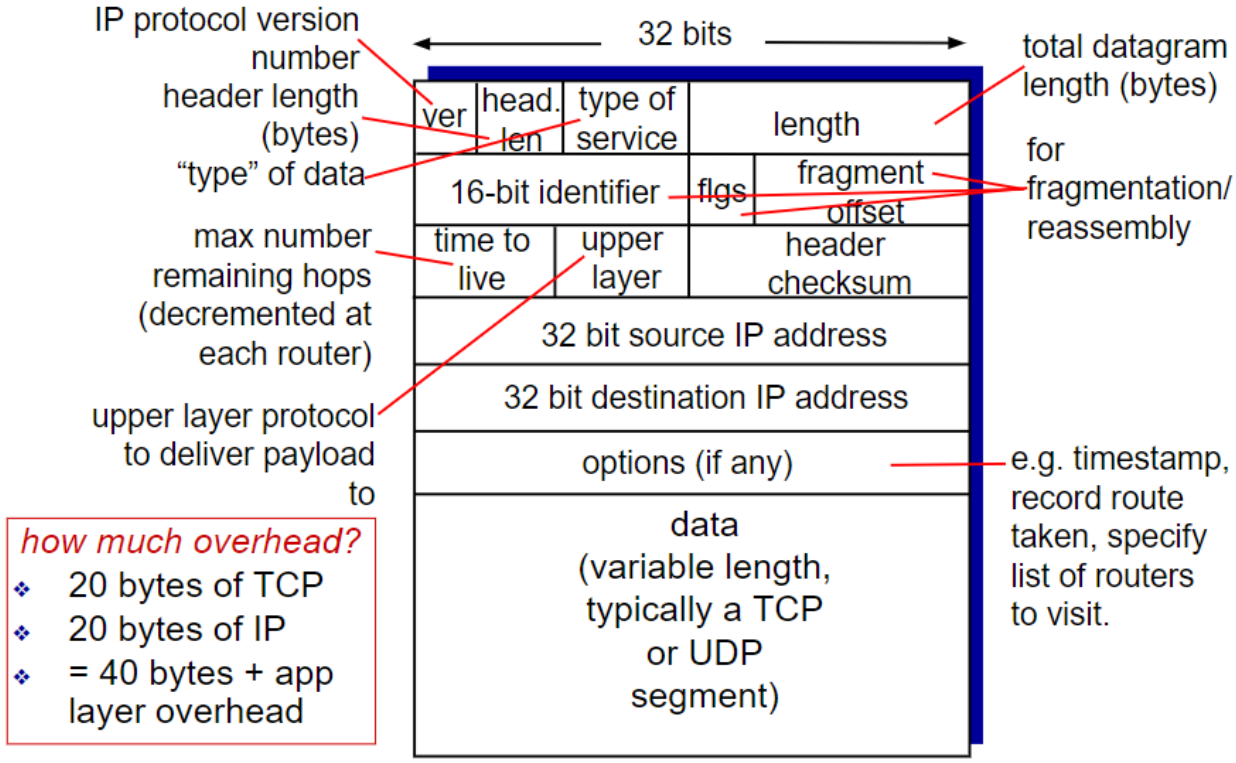



# Il livello di rete




Attualmente ci troviamo nel livello di rete, non ci sono più i pacchetti TCP o UDP come abbiamo detto in precedenza ma abbiamo **protocolli di routing** che sono mondiali, abbiamo il protocollo **IP** e **ICMP** (comando di ping ad esempio è un pacchetto ICMP).

# Come è formato il pacchetto IP




- **IP version**: la versione attuale è ancora IPv4 ma tra qualche anno ci aspetta di avere IPv6
- **Head Len**: indica quante opzioni ci sono
- **Type of service**: tipo di servizio richiesto
- **Length**: quanto è lungo tutto il pacchetto
- **Frammentazione**: indica quanto deve essere lungo il pacchetto da inviare, che può essere meno della dimensione attuale con un identificativo, con dei flag per dire se è l'ultimo pacchetto o meno.
- **TTL:** ovvero quanti router può passare prima di raggiungere la destinazione, quando raggiunge lo 0 il pacchetto viene scartato.
- **Upper Layer**: quale protocollo è stato usato
- **Checksum:** per vedere se il pacchetto è ancora integro, ma oggigiorno è praticamente inutile perchè avviene il controllo sul livello TCP

- **IP sorgente:** da dove arriva il pacchetto
- **IP destinazione:** ip di destinazione del pacchetto
- **Opzioni:** opzioni come la priorità
- **Dati:** un pacchetto con header e dati TCP o UDP

Se ci viene suggerito di ignorare gli header il pacchetto è di 1460 bytes. I dati che dobbiamo spedire sono massimo 1460 quindi.

## Frammentazione dei pacchetti IP

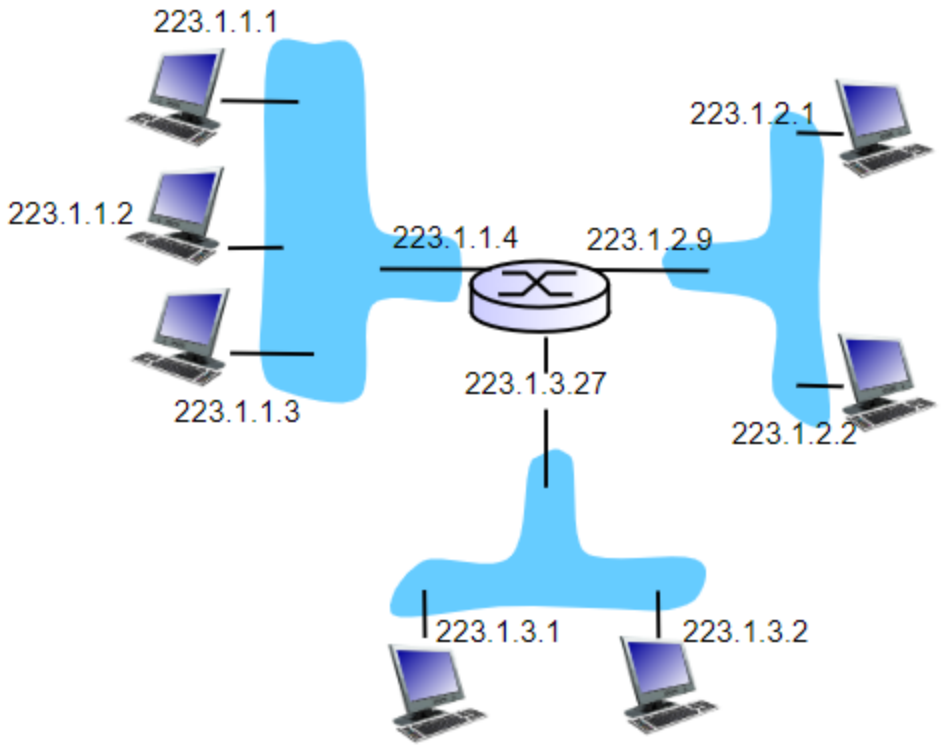
I router in rete hanno una MTU, ovvero il massimo che un pacchetto può esser grande per esser gestito da quel link, se il pacchetto è più grosso deve esser **frammentato**. Ci sono quindi i dati che sono divisi in pacchetti e vengono inviati, vediamo come avviene la frammentazione di un pacchetto grande 3980 bytes:

| Frammento | Byte | Id | Offset | Flag |
|---|---|---|---|---|
| 1° frammento | 1480 bytes | 777 | 0 | 1 |
| 2° frammento | 1480 bytes | 777 | 185 (1480) | 1 |
| 3° frammento | 1020 bytes | 777 | 370 (2960) | 0 |

L'offset indica dopo quanti byte bisogna inserire i dati in arrivo, il calcolo è `offset * 8.` Nota per la dimensione dell'ultimo pacchetto: totale del pacchetto di 3980 - 1480 - 1480 = 1020 bytes.

# Indirizzamento degli IP

L'indirizzo IP è di 32 bit e l'interfaccia del router anche essa a 32 bit, abbiamo una divisione tra parte di host e parte di rete. Abbiamo l'interfaccia che è il collegamento tra router e il dispositivo. ogni IP è associato a una interfaccia.

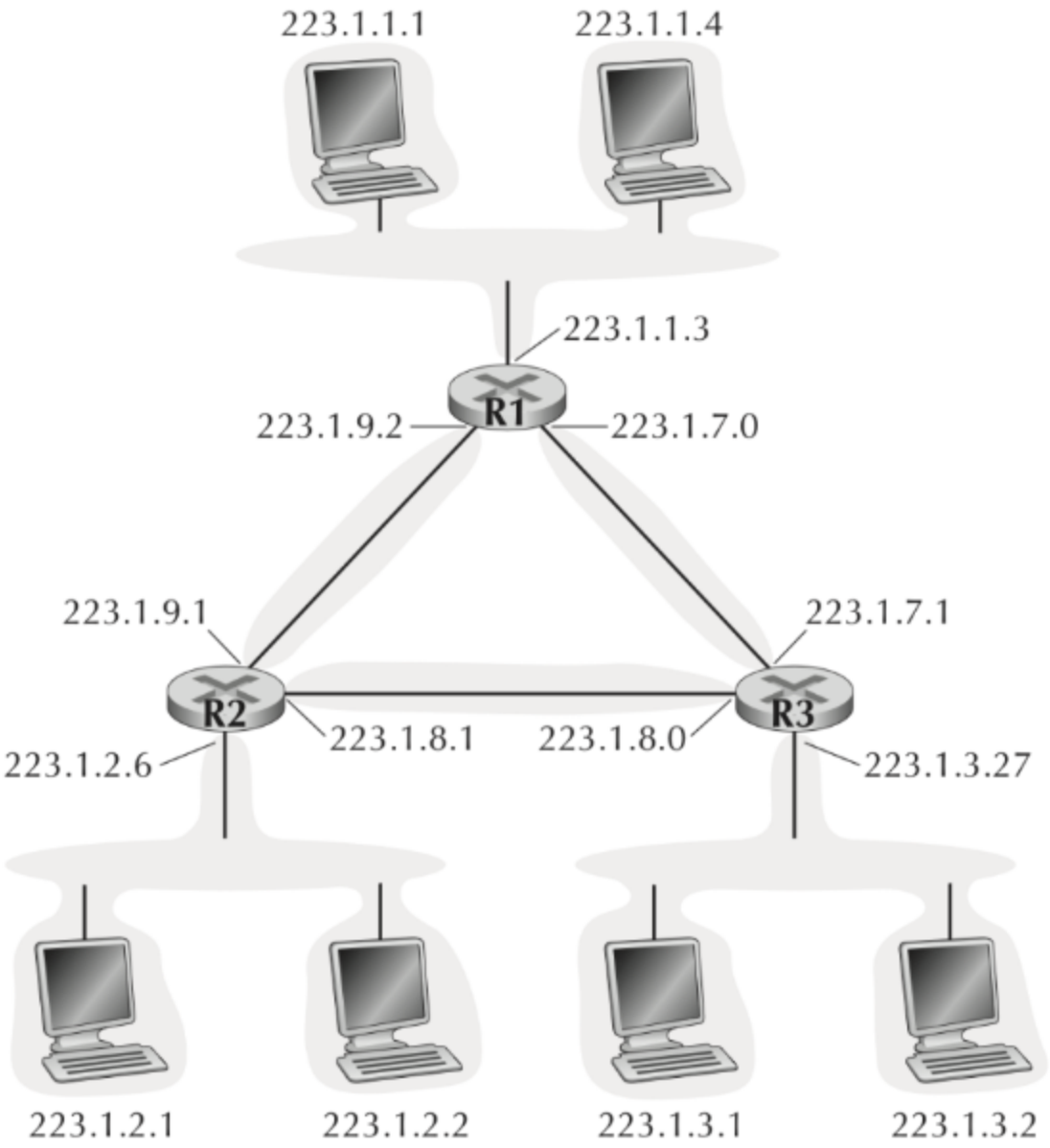



223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

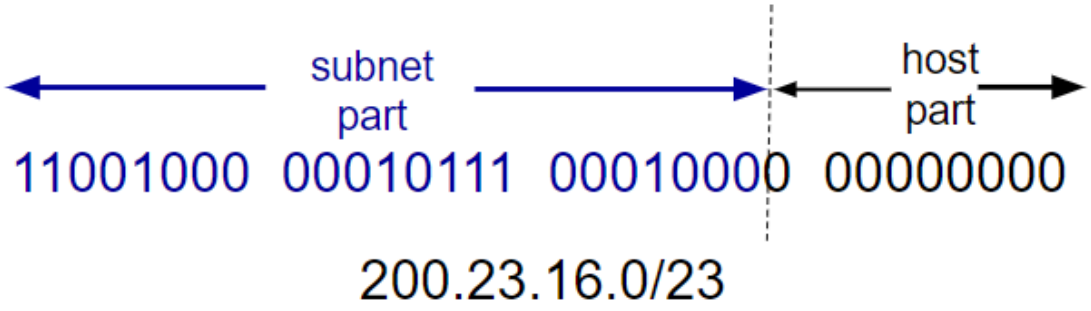
Il router andrà ad **associare** in quale **sottorete di trova la destinazione** lo capisce perchè è la prima da sinistra disponibile per la sottorete (in questo caso il terzo valore è la sottorete). E il restate byte invece per gli host. Da molti anni a questa parte le sottoreti non si dividono più in byte ma per non sprecare spazi in una rete abbiamo un indirizzamento più **dinamico.**

I primi 24 bit sono di rete mentre i restanti 8 degli host, questo significa che per identificare una rete ci vogliono 24 bit e per un host all'interno della rete ce ne vogliono 8. Andremo a scrivere quindi /24 per indicare quali bit sono fissi. Il router va a controllare solo i primi 3 byte per l'indirizzamento.




In questo caso abbiamo 6 sottoreti, vi bastano 3 bit del campo subnet per poterle identificare quindi avremo uno /19 e i restanti sono per la parte di host.

**CIDR**: Classless InterDomain Routing ovvero poter dare una porzione arbitraria alla lunghezza dei campi. Avremo sempre una divisione del tipo **a.b.c.d/x** dove x è il numero in bit della subnet.




### Esercizio

Per una Intranet si ha a disposizione la rete in classe B 129.174.0.0. Nella Intranet occorre installare almeno 15 reti locali collegate mediante dei router;

Descrivere come possono essere ricavati gli indirizzi per le sotto-reti e dire quanti host al massimo possono contenere le sotto-reti.

- **Sottorete 1**: 129.174.0.0/20, **gateway**: 129.174.0.0, **broadcast**: 129.174.0.255
- **Sottorete 2**: 129.174.1.0/20, **gateway**: 129.174.1.0, **broadcast**: 129.174.1.255
- **Sottorete 3**: 129.174.2.0/20, **gateway**: 129.174.2.0, **broadcast**:129.174.2.255
- **Sottorete 4**: 129.174.3.0/20, **gateway**: 129.174.3.0, **broadcast**: 129.174.3.255
- **Sottorete 5**: 129.174.4.0/20, **gateway**: 129.174.4.0, **broadcast**: 129.174.4.255

...

- **Sottorete 15**: 129.174.15.0/20, **gateway**: 129.174.15.0, **broadcast** 129.174.15.255

Ogni sottorete può avere al massimo 4094 host dato che i due estremi sono bloccati per gateway e host (12 bit per la parte host).

# Assegnazione degli IP

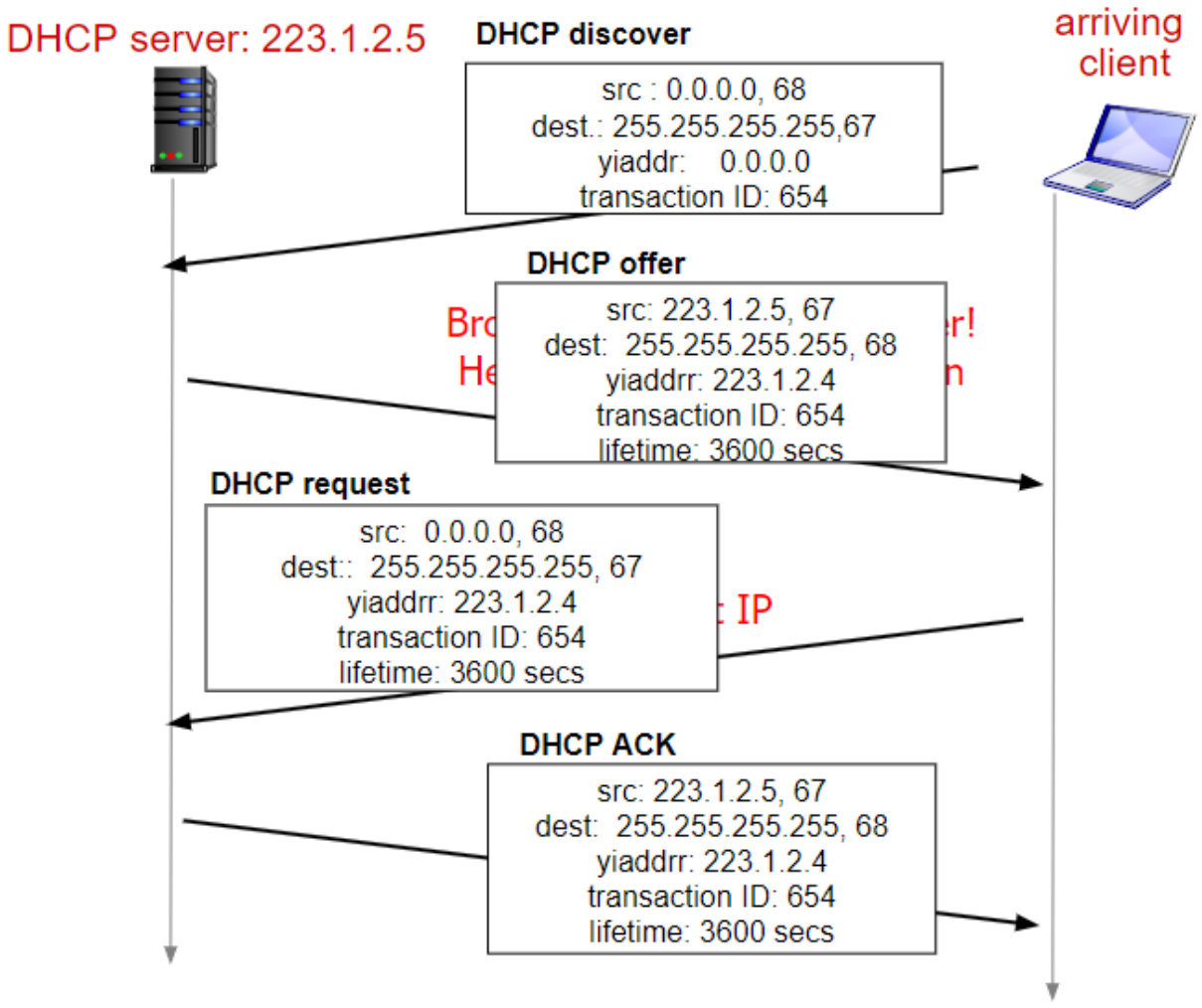
Come avviene l'assegnazione degli indirizzi IP in una rete? Ci sono due metodi:

- **Hard coded**: ovvero si assegna da pannello terminale l'ip del dispositivo
- **DHCP** (Dynamic Host Configuration Protocol): un server con un range di indirizzi assegna in automatico l'indirizzo

# DHCP

Un host per richiedere un indirizzo ha una sequenza di messaggi con il server sulla **porta 67**:

- DHCP **discover**: msg → opzionale
- DHCP **offer**: msg → opzionale
- DHCP **request**: msg
- DHCP **ack**: msg




Ecco il funzionamento per l'assegnazione di un indirizzo:

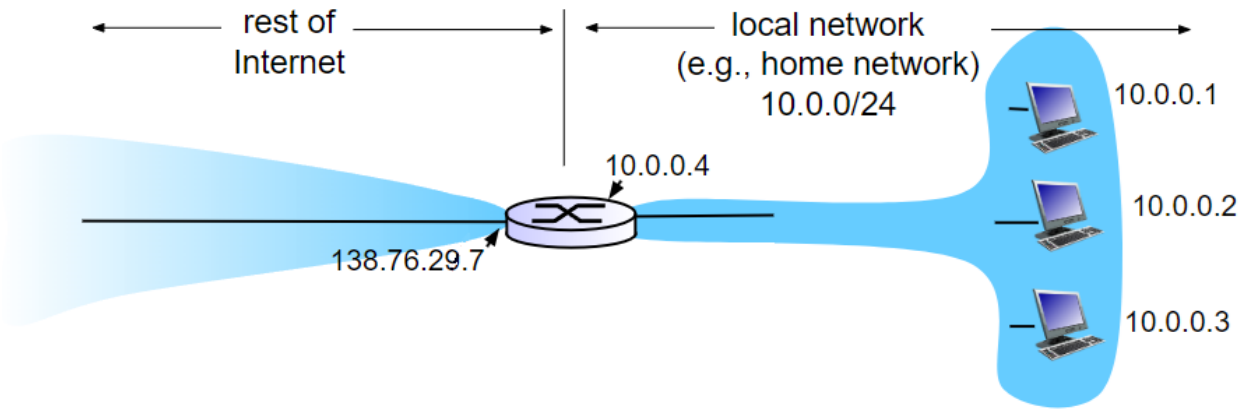
1. Il client arriva alla rete ed effettua una “**discover**” ovvero invia un messaggio broadcast a tutta la rete, sarà poi il server DHCP a rispondere.
2. Il server risponde con una “**offer**” con una broadcast dato che il client non ha ancora un indirizzo IP, il messaggio contiene l’indirizzo e il client può trovare questo messaggio tramite il transaction ID.
3. Il client risponde in broadcast con la “**request**”, questo perchè il client può ricevere più offerte da parte di più server DHCP, dato che la discover era in broadcast possono arrivare più risposte, basandoci sull’IP offerto, il server DHCP che l’ha fornito sa che è stato scelto il suo e anche sul transaction ID.
4. Il server risponde con un “**ack**” ovvero che il server DHCP fornisce l’autorizzazione finale al client per poter usare quell’indirizzo.

Il protocollo DHCP fornisce in risposta molto più che un indirizzo IP, ci fornisce anche il **gateway**, il nome del server **DNS** e anche la **netmask** per capire a quale sottoreti apparteniamo.

Un ISP riceve un blocco di IP **dall’ICANN** ovvero un’organizzazione che gestisce i DNS root, e risolve dispute sull’assegnazione dei nomi di dominio.

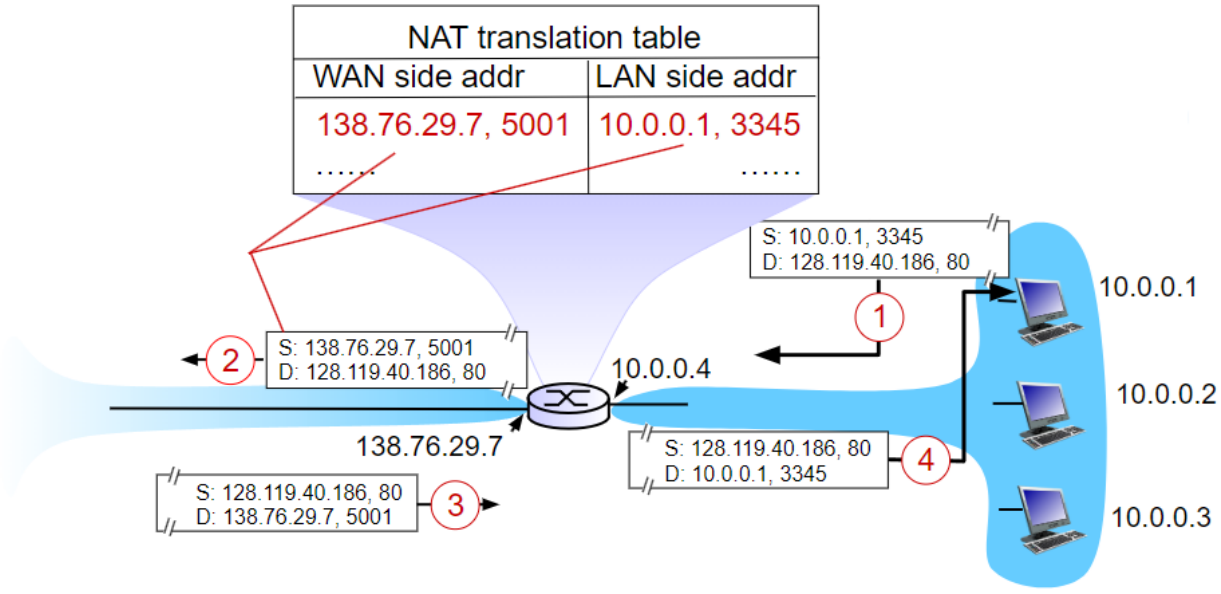
# Network Address Translation (NAT)

In una casa, ufficio o altre strutture, non possiamo assegnare a ogni host un indirizzo univoco altrimenti gli indirizzi IP sarebbero già finiti da tempo! Proprio per questo si è semplificato in questo modo il processo: assegnare a un gruppo di host della stessa rete un **unico indirizzo IP di uscita**.



Quando i pacchetti di host diversi lasciano la rete avranno tutti lo stesso IP di sorgente in questo caso 128.76.29.7 mentre la rete locale ha un range privato ovvero 10.0.0/24.

Com'è possibile? Attraverso le porte e una tabella di corrispondenze tra l'host che ha fatto la richiesta e la **traduzione** in indirizzo IP pubblico. Quando riceviamo la risposta sarà il NAT che si è segnato in quale porta era stata fatta la richiesta e manda la risposta al giusto host.




1. L'host con indirizzo 10.0.0.1 alla porta 3345 vuole fare una richiesta al sito 128.119.40.186 sulla porta 80
2. Il NAT a questo punto crea una tupla che associa l'host con l'indirizzo ip pubblico 138.76.29.7 sulla porta 5001
3. La risposta arriva all'indirizzo del NAT sulla porta 5001, questo router speciale risolve l'associazione all'host
4. L'host riceve il messaggio dal NAT e quindi abbiamo effettuato il percorso

Il campo port number ha 16 bit, quindi possiamo gestire 60000 host in una rete.

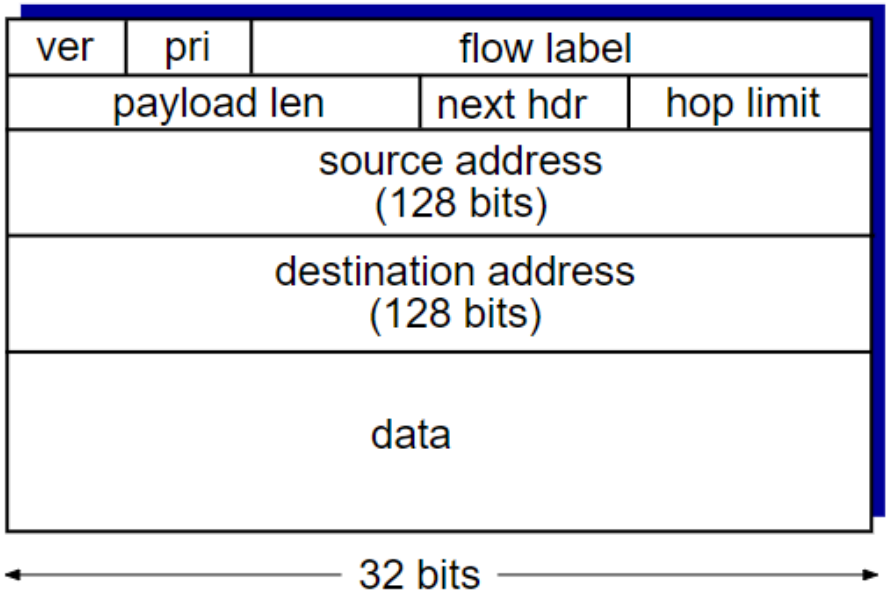
## Il NAT è controverso

I NAT essendo dei router dovrebbero processore solo dati a livello 3, invece vanno a usare le porte che sono di livello 4, con l'indirizzamento IPv6 si dovrebbe risolvere il problema degli IP e quindi i NAT non dovrebbero più esistere. Se un host fosse un server che deve fornire un servizio, il NAT dovrebbe lasciare passare dei dati su quelle porte del server.

# Protocollo IPv6

Nasce per sopperire alla ormai poca disponibilità di IPv4. Tra qualche anno ci sarà il cambio totale, nel 2011 c'è stato l'ultimo blocco di indirizzi assegnato e quindi ufficialmente non ci sono più blocchi disponibili.

L'header adesso misura 40 byte e non esiste più frammentazione del pacchetto.




2^128 about 3e+38

- **Version**: quale versione di IP si sta usando
- **Priority**: se il pacchetto ha priorità rispetto ad altri
- **Flow Label**: in quale flow si sta usando il pacchetto (es. videochiamate o streaming)
- **Payload Length**: quanto è lungo il messaggio
- **Next Header**: se i dati del pacchetto sono TCP o UDP
- **Hop Limit**: quanti hop prima di bloccare il messaggio (ex. TTL)

- **Source address**: chi spedisce il messaggio
- **Destination address**: chi lo deve ricevere

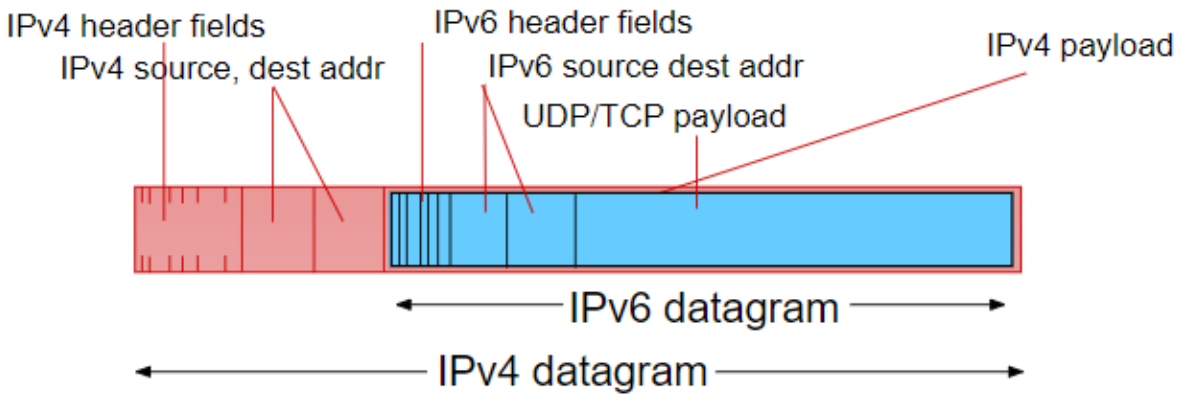
Potremmo dare un indirizzo a ogni granello di sabbia al mondo con IPv6. Altri cambiamenti sono:

- Rimosso il checksum perchè lento
- Mosso il campo opzioni, non è più nell'header
- Aggiunto ICMPv6 ovvero per indicare se il pacchetto è troppo grande (Packet Too Big all'host mittente) e supporta messaggi unicast ovvero gruppi IP non contigui.

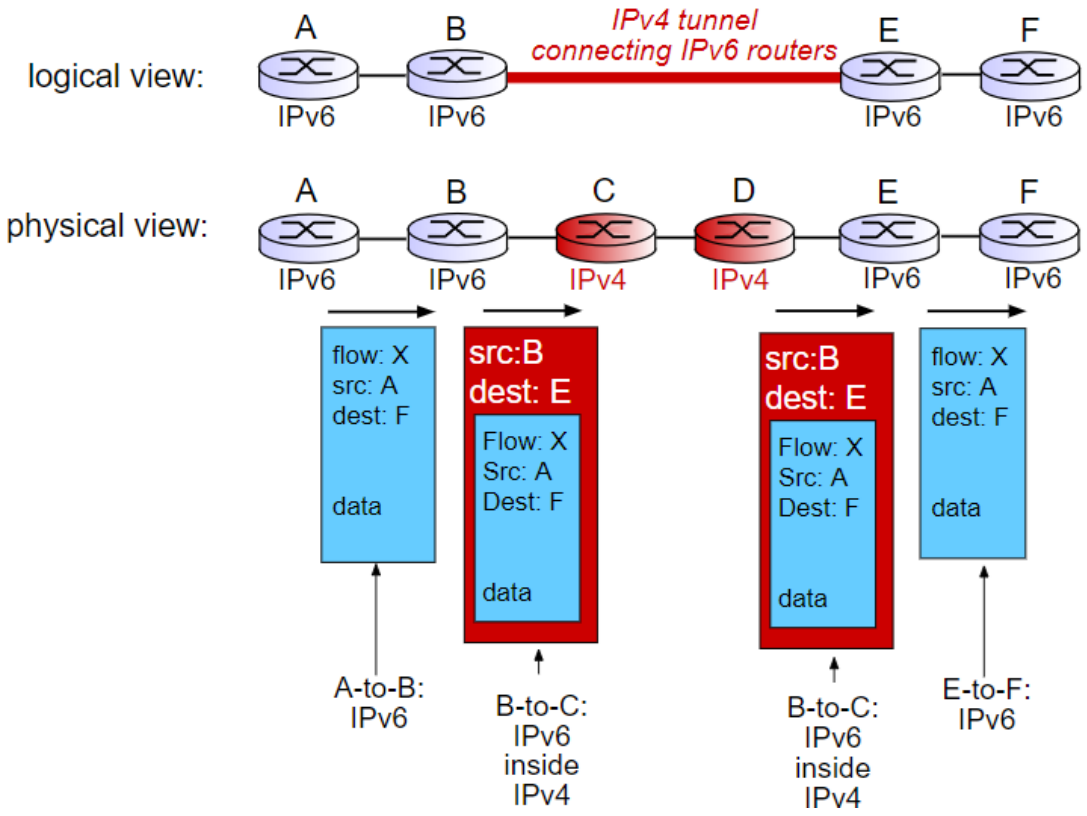
## Migrazione da IPv4 a IPv6

Attualmente ci sono routers che supportano IPv4 e altri IPv6 quindi come si può fare la transizione completa?

Non possiamo cambiare da un giorno all'altro tutti i router del mondo di usare IPv6. Questo perchè ci sarebbero bug e problemi di coordinazione tra le varie reti. Attraverso il **tunneling** possiamo sfruttare la presenza in contemporanea dei due, se un mittente e un destinatario sono entrambi in IPv6 possiamo comunque passare per un router IPv4, ovvero inglobando il pacchetto IPv6 dentro il payload di IPv4.

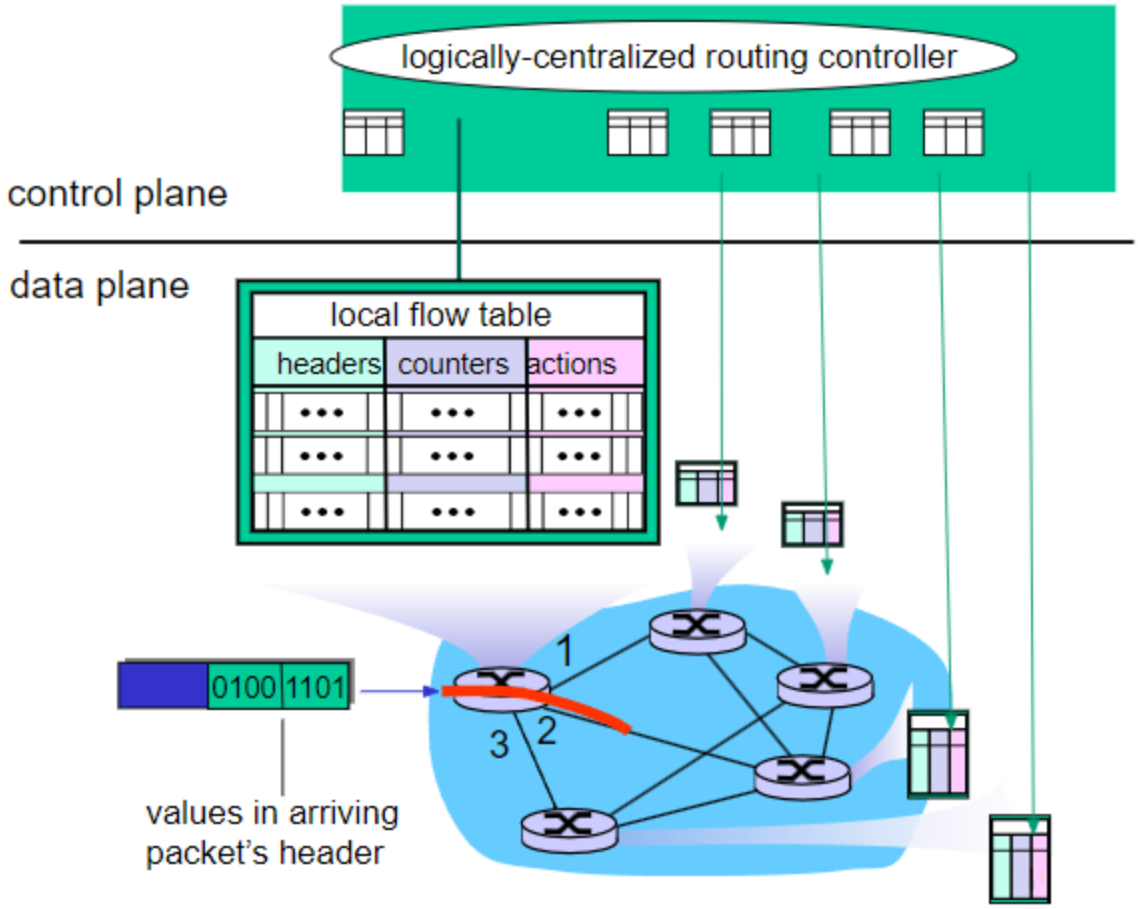



In azione quindi si crea un tunnel logico tra l'ultimo nodo IPv6 e il primo nodo prima del destinatario IPv6.




# Forwarding Generalizzato e SDN

Ogni router ha una tabella di flusso ovvero tabelle che per l'inoltro si possono basare su: **header**, **conteggio** o **azioni**.

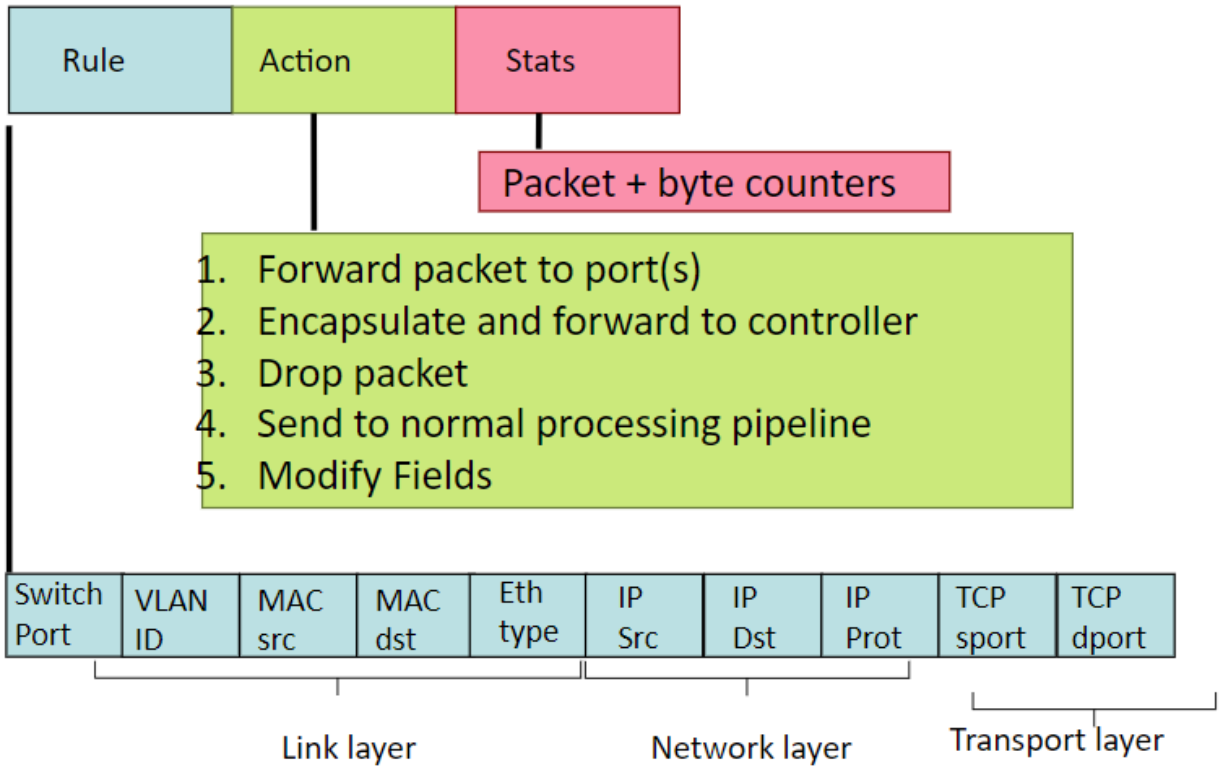



Attraverso alcuni pattern nell'header possiamo definire azioni di inoltro verso una porta. Si possiamo modificare il pacchetto, eliminarlo o inoltrarlo.

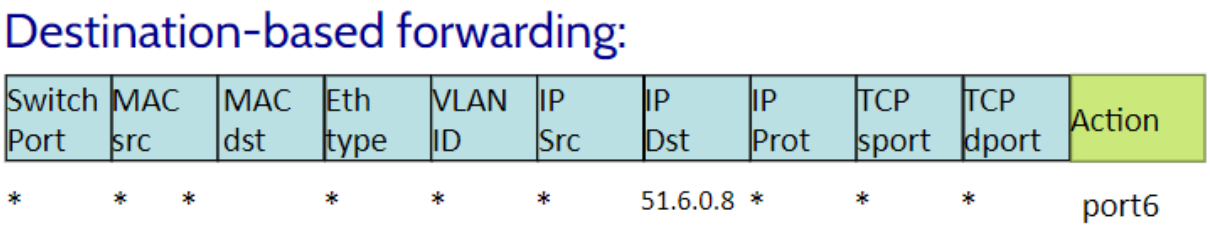
Esempio di regole:

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

Come possiamo vedere in questa **flow table** ci sono **regole**, **azioni** e **statistiche**.




Abbiamo un esempio pratico su come si usa questa flow table:




In questo caso se l'indirizzo di destinazione è 51.6.0.8 l'azione è quella di inoltrarlo alla porta 6. A seconda dell'unione **match + action** possiamo identificare Router, NAT, Switch o Firewall grazie a OpenFlow (SDN = Software Defined Networking).