

# Appunti Advanced Networking and Wireless Systems

lezioni aa 2019/20

## Contents

<b>1 IPv6</b>	<b>3</b>
1.1 IPv6 Base Header . . . . .	3
<b>2 IPv6 Header Structure</b>	<b>5</b>
2.1 Hop-by-Hop Options header . . . . .	6
2.2 Routing Header . . . . .	7
2.3 Fragment Header . . . . .	7
2.4 Altri Extension Headers . . . . .	8
<b>3 Categorie di indirizzi IPv6</b>	<b>9</b>
3.1 Link IPv6 . . . . .	9
3.2 Address Scope . . . . .	10
3.3 Notazione di indirizzi . . . . .	10
3.4 Prefissi: notazione e allocazione . . . . .	10
3.4.1 Global Unicast . . . . .	10
3.4.2 Indirizzi Anycast . . . . .	12
3.4.3 Indirizzi Multicast . . . . .	12
<b>4 ICMPv6 e Autoconfigurazione</b>	<b>13</b>
4.1 ICMPv6 . . . . .	13
4.1.1 Formato del messaggio . . . . .	13
4.1.2 Neighbour Discovery protocol . . . . .	14
4.1.3 Router Discovery . . . . .	14
4.2 Configurazione dell'indirizzo . . . . .	15
<b>5 6LoWPAN</b>	<b>17</b>
5.1 Architettura 6LoWPAN . . . . .	17
5.2 Multi-hop forwarding in Low Power and Lossy Networks . . . . .	17
5.2.1 L2 forwarding (Mesh-Under) . . . . .	18
5.2.2 L2 forwarding vs IP model . . . . .	18
5.2.3 L3 forwarding (Route-Over) . . . . .	20
5.2.4 L3 forwarding vs IP model . . . . .	20
5.3 Modello link 6LoWPAN . . . . .	20
5.4 Addressing . . . . .	20
5.5 6LoWPAN adaptation layer . . . . .	21
5.5.1 6LoWPAN header stack . . . . .	21
5.5.2 Mesh addressing layer . . . . .	22
5.5.3 Fragment header . . . . .	22
5.5.4 Header compression . . . . .	23
5.6 6LoWPAN Neighbour Discovery (ND) . . . . .	24
5.6.1 Address Configuration . . . . .	25
5.6.2 Address Registration . . . . .	25

<b>6 RPL</b>	<b>27</b>
6.1 Premessa . . . . .	27
6.2 Concetti generali . . . . .	27
6.3 Istanze RPL . . . . .	27
6.4 Funzione Obiettivo (FO) . . . . .	28
6.5 Formazione del DODAG . . . . .	28
6.5.1 Algoritmo Trickle . . . . .	29
6.6 Comunicazioni Point-to-Point e Point-to-Multipoint . . . . .	31
<b>7 CoAP</b>	<b>32</b>
7.1 Concetti generali . . . . .	32
7.2 Trasmissione dei messaggi . . . . .	33
7.3 Resource Observing . . . . .	36
7.4 Resource Discovery . . . . .	37
<b>8 QoS - Quality of Service</b>	<b>39</b>
8.1 Algoritmi QoS . . . . .	39
8.2 Algoritmi di Scheduling . . . . .	41
8.2.1 GPS - Generalized Processor Sharing . . . . .	41
8.2.2 WFQ - Weighted Fair Queueing . . . . .	43
8.2.3 (WF) <sup>2</sup> Q - Variante di WFQ . . . . .	45
8.2.4 DRR - Deficit Round Robin . . . . .	46
<b>9 DiffServ</b>	<b>47</b>
9.1 Traffic Classification . . . . .	47
9.2 Traffic Conditioning . . . . .	47
9.2.1 Policing . . . . .	47
9.2.2 Shaping . . . . .	48
9.3 DiffServ Code Point . . . . .	48
9.4 Active Queue Management . . . . .	49
9.4.1 RED - Random Early Detection . . . . .	49
<b>10 Core Network Protocols and Architectures</b>	<b>50</b>
10.1 MPLS - Problema . . . . .	50
10.2 MPLS - Funzionamento . . . . .	51
10.2.1 Concetto Label Switching . . . . .	51
10.2.2 Architettura MPLS . . . . .	51
10.2.3 LSP setup control . . . . .	57
10.2.4 Label Distribution Protocol (LDP) . . . . .	60
10.3 RSVP per la distribuzione dei label . . . . .	60
10.4 Supporto MPLS per DiffServ . . . . .	60
<b>11 Traffic Engineering</b>	<b>63</b>
11.1 Link Characterization . . . . .	63
11.2 Protocolli di Routing Estesi . . . . .	63
11.2.1 CSPF . . . . .	63
11.2.2 RSVP-TE . . . . .	63
11.2.3 Fast Re-Routing . . . . .	64
<b>12 BGP - Border Gateway Protocol</b>	<b>65</b>
<b>13 VPN</b>	<b>67</b>
13.1 Concetti generali . . . . .	67
13.2 BGP/MPLS IP VPN . . . . .	68

# 1 IPv6

Gli indirizzi IPv4 stanno finendo a causa della rapida crescita degli utenti Internet (limite teorico di 4.3 miliardi di indirizzi), quindi perchè IPv6 non è ancora lo standard?

- Implica un grande cambiamento nell'infrastruttura di rete.
- Infrangerebbe un grande principio del protocollo IP: l'*univocità* degli indirizzi.
- Il NAT funziona talmente bene nel risolvere problemi di spazio di indirizzamento IPv4 che fino ad oggi non si è sentito il bisogno di passare a IPv6.

Tuttavia le cose ora stanno cambiando: la crescita demografica accoppiata ad una crescente richiesta di accesso ad internet (pensiamo a paesi come Cina ed India) si traducono in una grande richiesta di indirizzi IP nel futuro prossimo. Inoltre, in futuro, con l'avvento dell'IoT ci saranno numerosi oggetti sempre connessi ad internet (es. sensori domestici) per i quali servirà un indirizzo IP permanente (*always-on-access*).

## 1.1 IPv6 Base Header



Figure 1: IPv6 Base Header

- **Version (4 bits)**: indica che stiamo utilizzando IPv6.
- **Traffic class (1 byte)**: sostituisce il campo Type of Service dell'IPv4. Può essere utilizzato per dare priorità a certi a datagram di certe applicazioni (per esempio, pacchetti ICMP) rispetto a datagram di altre applicazioni.
- **Flow label (20 bits)**: generato randomicamente. Distingue pacchetti che richiedono stessi trattamenti in modo da facilitare la gestione del traffico real time. È considerata una feature sperimentale. Per alcune applicazioni è importante andare a rilevare una sequenza di pacchetti trasmessi da un sender a un receiver. Questo perché i requisiti di un'applicazione possono dipendere da alcune metriche di performance che considerano più pacchetti contemporaneamente. Senza la flow label, per identificare pacchetti appartenenti allo stesso flusso si usavano implicitamente gli indirizzi di destinazione e sorgente. Con questo nuovo parametro si può specificare in modo esplicito. Il sender mette lo stesso valore di flow label per i pacchetti che appartengono allo stesso flow, quindi il flusso viene riconosciuto dai router guardando IP Address del sender e flow label. Da aggiungere, inoltre, che con il NAT non funzionava, perché il source address cambia (ci possono essere più sender dietro lo stesso NAT con lo stesso indirizzo).
- **Payload length (2 bytes)**: non include la lunghezza dell'header, com'era in IPv4. Gli header di estensione sono considerati come parte del payload.
- **Hop limit (1 byte)**: Analogico al campo TTL nell'IPv4, ma in questo caso non è espresso in secondi. Quindi, ogni device intermedio prima della destinazione, riduce di uno l'hop limit, se questo arriva a 0 il pacchetto viene scartato.

- **Next header** (*1 byte*): Assomiglia al tipo di protocollo in IPv4 ma è molto di più, riflette la nuova organizzazione dei pacchetti IPv6.

Nell'header dell'IPv4 c'era IHL (Internet Leader Length) che serviva per comunicare al ricevitore e ai nodi intermedi la lunghezza dell'header, che era variabile per via del campo options di dimensioni variabili. IPv6, invece ha un header di dimensione fissa, il controllo viene fatto via Hardware. 40 Byte. IPv6 non è provvisto di frammentazione. In IPv4, se il pacchetto è più grande di MTU, si frammenta; in IPv6 si droppe. MTU varia a seconda del canale. 1280 è il minimo MTU che si può assumere su un qualunque link di internet. Se si vuole spedire di più, bisogna stimare MTU lungo il path.

## 2 IPv6 Header Structure

Un Base Header (40 bytes), più zero o più extension headers di dimensione variabile.

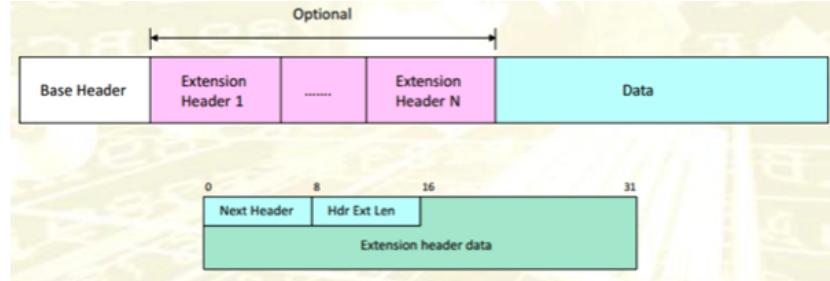


Figure 2: IPv6 Header Structure

Ogni extension contiene una sola feature e tutti i dati necessari a questa feature. Next header contiene le informazioni sul tipo di dati contenuti nel successivo header. Header extensions length contiene la lunghezza dell'header, in pratica contiene un puntatore alla prossima estensione.

Quindi, ogni extension header può essere considerato come un sotto-protocollo che svolge una nuova funzione. E' possibile che in un extension header sia contenuto un altro pacchetto IPv6: questa tecnica è chiamata "tunneling".

Valori nel campo "Next Header" (base e extension):

Values	Meaning
0	Hop-by-Hop Options header
4	IPv4
6	Transmission Control Protocol
17	User Datagram Protocol
41	IPv6
43	Routing header
44	Fragment header
46	Resource Reservation Protocol
50	Encapsulating Security Payload header
51	Authentication header
58	Internet Control Message Protocol v6
59	No Next header
60	Destination Options header
135	Mobility Header (Mobile IPv6)
...	...

Figure 3: IPv6 Next Header Values

Gli extension headers vengono inseriti solo quando è necessario. Sono processati nell'ordine in cui appaiono dal nodo identificato dal destination address.

Valori che può assumere il campo Next Header:

Value	Extension Header
0	Hop-by-Hop Options Header
43	Routing Header
44	Fragment Header
50	ESP Header
51	Authentication Header
59	No Next Header
60	Destination Options Header

Figure 4: IPv6 Next Header Values used

I valori “ESP header” e “Authentication Header” servono per sicurezza, autenticazione ed encryption. Valore “No next Header” serve a specificare che non c’è nient’altro nel pacchetto dopo.

La regola generale è che il base header venga processato da ogni dispositivo intermedio da cui passa il pacchetto (ogni router del percorso più la destinazione). Mentre gli extension header vengono lette solo a destinazione. L’unica eccezione riguarda l’ “hop by hop options header”.

Il “processare un header” si riferisce al fatto che il payload viene mandato al livello superiore. Solitamente si rispetta un ordine, ovvero il “routing header” deve essere prima del “fragment header”.

## 2.1 Hop-by-Hop Options header

Porta informazioni opzionali che devono essere esaminate da ogni nodo sul cammino (hop-by-hop).

Se il pacchetto contiene più extension headers, questo deve essere il primo.  
Formato:

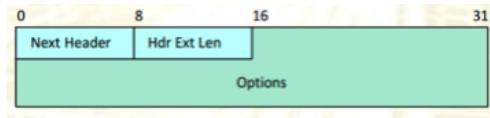


Figure 5: Hop-by-Hop Option Header Format

- **Next Header** (*1 byte*)
- **Header Extension Length** (*1 byte*)
  - Lunghezza dell’header in unità di otto byte (meno 1)
- **Options**: una o più

Per quanto riguarda il campo Options:

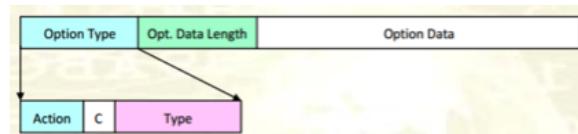


Figure 6: Option field

- **Action** (*2 bits*): Cosa fare se l’opzione non è riconosciuta
  - **00**: Salta e continua a processare.
  - **01**: Scarta il pacchetto.
  - **10**: Scarta il pacchetto e manda un messaggio ICMP Parameter Problem all’indirizzo Source del pacchetto.
  - **11**: Scarta il pacchetto e manda un messaggio ICMP Parameter Problem all’indirizzo Source del pacchetto solo se la destinazione non è un indirizzo multicast
- **C** (*1 bit*)
  - **1**: l’informazione di opzione può cambiare in viaggio
  - **0**: l’informazione di opzione non cambia in viaggio
- **Type**

- **Jumbo Payload** (*Type = 194*): Utilizzato per inviare pacchetti molto grandi la cui lunghezza non può essere codificata su soltanto 16 bits (>64 KB). Quando viene utilizzato, il campo length del payload IPv6 è settato a 0. La lunghezza del pacchetto è codificata con 32-bits, supporta la trasmissione di pacchetti che stanno tra 65.536 e 4.294.967.295 bytes (4GB). Compromesso tra il design iniziale di IPv6 e requirements speciali di networking. Quindi, le informazioni del payload non vengono messe all'interno della sezione payload ma all'interno dell'estensione.
- **Router Alert** (*Type = 5*): Indica ad un router sul cammino di forwarding che il pacchetto contiene informazioni importanti che devono essere processate dal router. Esempio: RSVP utilizza pacchetti di controllo che contengono informazioni che hanno bisogno di essere interpretate o aggiornate da routers sul cammino.

## 2.2 Routing Header

Definisce una lista di nodi intermedi che devono essere visitati sul cammino alla destinazione

- **Routing Type**
  - 0: default
  - 2: Mobile IPv6
  - 3: RPL
- **Segment Left:** nodi rimanenti che devono essere visitati
- **Address** (RT = 0)

Di solito non si ha il controllo sulla rotta utilizzata per raggiungere la destinazione finale. Per avere il controllo sul percorso si usa questa estensione, dove si indica la lista dei router che si vuole che vengano attraversati per consegnare il pacchetto. Non si vanno a rimuovere via via i routers, cosicché il destinatario può sapere il percorso fatto dal pacchetto per raggiungerlo. Non si tiene però conto, in questo modo, del fatto che uno di questi router potrebbe essere congestionato. Soluzione: loose control, ovvero non si specifica tutta la lista ma solo un hop dal quale attraversare.

A volte è necessario costruire router senza routing table. Rilevante nelle capillary network dove il router potrebbe essere un sensore. Accade ad esempio in RPL (Routing protocol for low power and lossy network). Un problema è legato al fatto che alcuni nodi intermedi potrebbero essere malevoli e leggere i pacchetti che poi verranno inoltrati. Un altro possibile problema riguarda il DoS attack: ovvero si possono modificare gli indirizzi dei nodi intermedi e far convergere tutto il traffico verso un solo nodo. Per questo motivo adesso questo header è deprecato.

## 2.3 Fragment Header

Diversamente da IPv4, i routers non frammentano pacchetti IPv6. La frammentazione avviene solo all'host sorgente, che manda il pacchetto. L'host di destinazione gestisce semplicemente il riassamblamento. I pacchetti IPv6 più larghi dell'MTU sul link di forwarding sono scartati dal router.

Gli host IPv6 utilizzano una procedura di discovery di cammini MTU. La minima grandezza IPv6 MTU è 1280 bytes.

Questo header viene usato dal sender nel caso voglia frammentare il pacchetto. Solo il sender può usare la frammentazione e il ricevente si preoccupa di ricongiungere i pacchetti frammentati. I nodi intermedi non usano frammentazione, inoltrano solo gli stessi pacchetti.

Per definizione un link in cui si usa IPv6 deve garantire un minimum MTU pari a 1280 bytes, per cui se un sender manda un pacchetto di dimensione minore a 1280 bytes saprà con certezza che il pacchetto arriverà a destinazione senza essere scartato o frammentato dai router lungo il path. E se un sender volesse mandare pacchetti più grandi di 1280 bytes, deve usare la path MTU discovery per scoprire se lungo il cammino è disponibile un MTU maggiore di 1280 bytes, oppure lo stesso sender può usare la frammentazione.

Path MTU discovery procedure: in questa procedura si usa ICMPv6 per scambiarsi informazioni riguardo l'MTU. Per scoprire la minima MTU si manda un pacchetto di una certa dimensione e se non riceviamo in risposta nessun errore ICMPv6, allora l'MTU minimo è sicuramente maggiore della dimensione del pacchetto. Se il minimo MTU è minore, il router scarterà il pacchetto e manderà un pacchetto ICMPv6 di errore al sender con all'interno il valore del minimo MTU. Se un link non è in grado di garantire una MTU di 1280 bytes, allora non potrà utilizzare IPv6 (sarà un problema con IoT).

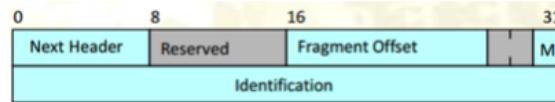


Figure 7: Fragment Header

- **Fragment Offset (13 bits)**: L'offset in unità di 8 byte del dato in questo pacchetto relativo all'inizio del dato nel pacchetto originale.
- **M-Flag (1 bit)**:
  - 1: più frammenti
  - 0: ultimo frammento
- **Identification (4 bytes)**: Generato dall'host sorgente in modo da identificare tutti i pacchetti che appartengono al pacchetto originale.

## 2.4 Altri Extension Headers

- Authentication Header
- Encapsulating Payload Security Header  
IPv6 security support (IPsec)
- No Next Header: No Payload
- Destination Options Header  
Lo stesso di Hop-by-Hop, ma le opzioni devono essere processate solo alla destinazione.  
Utilizzato dal Mobile IPv6

### 3 Categorie di indirizzi IPv6

Non è presente il broadcast in IPv6. Il broadcast veniva usato in IPv4 con DHCP ad esempio. Il Broadcasting in IPv4 è limitato alla stessa subnet (rete locale, ovvero il router non manda verso l'esterno i pacchetti broadcast).

- **Unicast:** identifica in maniera univoca un'interfaccia di un nodo IPv6. Un pacchetto inviato ad un indirizzo unicast è inviato all'interfaccia identificata da quell'indirizzo.
- **Multicast:** identifica un gruppo di interfacce IPv6. Un pacchetto inviato ad un indirizzo multicast è processato da tutti i membri del gruppo multicast.
- **Anycast:** assegnato a interfacce multiple (tipicamente su nodi multipli). Un pacchetto inviato ad un indirizzo anycast è consegnato a soltanto una di queste interfacce (tipicamente la più vicina).

In IPv6 si utilizzano di default dei Multicast address per identificare il servizio necessario. Questo indirizzo sarà associato al server che svolge quel servizio, quindi tutti i server DHCP avranno lo stesso indirizzo multicast IPv6. In questo modo il pacchetto viene ricevuto solo dal server DHCP e tutti gli altri nodi non interessati (dato che non forniscono quel servizio) non riceveranno il messaggio, non dovranno quindi processarlo per poi scartarlo (come in IPv4).

Un indirizzo IPv6 è assegnato ad un'interfaccia, almeno un indirizzo unicast per interfaccia di un nodo. Una singola interfaccia può essere assegnata a più indirizzi IPv6 di ogni tipo. Gli indirizzi IPv6 hanno uno “scope” (codificato come parte dell'indirizzo): lo “scope” è un intervallo topologico tra cui gli indirizzi potrebbero essere usati come identificatori unici. E' una parte dell'indirizzo che indica la parte della rete dove l'indirizzo può essere usato come unico. Ci sono scope globali e non globali.

Non è più presente ARP, il servizio viene fornito direttamente da IPv6. Scope è un sottoinsieme di tutta la network codificata con un indirizzo. In IPv4 c'è solo uno scopo, che è il global-scope, mentre in IPv6 c'è sia il global-scope che local-scope.

#### 3.1 Link IPv6

Identificato da un set di interfacce che possono comunicare direttamente tra di loro senza bisogno di passare da un router (hop singolo). Un link IPv6 è un'astrazione, insieme di interfacce che possono comunicare direttamente senza l'intermezzo di un router. Il protocollo IP introduce un modello per modellare la rete. Un link che pensiamo come ad esempio un filo, è in realtà un'astrazione di ciò che il sottostante link-layer protocol permette di fare. Se, per esempio, si dice che un link è broadcast, significa che la sottostante link-layer technology mette a disposizione di IP la comunicazione broadcast (non si parla quindi del mezzo).

Ethernet supporta broadcast, la tecnologia però, non lo è. È connessa a degli switch in unicast (point-to-point). Tipiche assunzioni riguardo ad un link:

- Stabile (nel tempo)
- Single link-layer broadcast domain
- Transitive (se A-B e B-C, allora A-C)

Implicazioni:

- I prefissi network possono essere usati per determinare se un'interfaccia è attaccata ad un certo link.
- La rilevazione di indirizzi duplicati può essere facilmente indirizzata

Se non vale la proprietà transitiva, non si può usare la on-link determination. Quindi, il discorso è che le funzionalità che otteniamo ad un determinato livello (in questo caso 3, IPv6) dipendono da cosa ottieni dai livelli sottostanti.

La transitività riguarda i link unicast, non broadcast. Se un link è broadcast puoi far girare una procedura (duplicate address detection - stateless address autoconfiguration) per controllare se ci sono altri nodi sullo stesso link che utilizzano lo stesso indirizzo che si vuole assegnare.

### 3.2 Address Scope

- **Global Address scope:** unico nell'intera internet
- **Link-local:** Assomiglia vagamente ai private address di IPv4, ma è più specifico. Grazie al link-local address ed al fatto che gli host possono autoconfigurare le proprie interfacce, la comunicazione su un link può avvenire senza bisogno di altro, senza bisogno di un gestore.  
L'indirizzo quindi è unico sul link al quale è attaccata l'interfaccia corrispondente, e non deve uscire da essa.
- **Unique local:** Indirizzo globalmente unico, ma valido solo in un sottoinsieme dei link. Non dovrebbe essere "routato" nel global Internet.  
In questo modo se 2+ sottoreti si vogliono unire, non ci saranno problemi di sovrapposizione degli indirizzi.

### 3.3 Notazione di indirizzi

**x:x:x:x:x:x:x:x** dove x è un blocco da 16bit rappresentato da 4 cifre esadecimali.

Regole di abbreviazione:

- Zeri iniziali possono essere saltati
- Zeri consecutivi possono essere sostituiti da ":" e questa regola può essere applicata soltanto una volta, altrimenti non sapresti la dimensione dei blocchi rimossi

L'indirizzo con tutti 0: **0:0:0:0:0:0:0:0** (::) indica un indirizzo non specificato.

L'indirizzo **0:0:0:0:0:0:1** (::1) indica l'indirizzo loopback, ovvero corrisponde al localhost in IPv4.

### 3.4 Prefissi: notazione e allocazione

Simile all'IPv4 con CIDR: [Indirizzo IPv6]/[lunghezza prefisso].

Identifica un set di indirizzi che, ad esempio, possono appartenere alla stessa subnet.

Allocation	Prefix binary	Prefix hex	Fraction of address space
<b>Unassigned</b>	<b>0000 0000</b>	<b>::/8</b>	<b>1/256</b>
Reserved	0000 001		1/128
<b>Global unicast</b>	<b>001</b>	<b>2000::/3</b>	<b>1/8</b>
<b>Link-local unicast</b>	<b>1111 1110 10</b>	<b>FE80::/10</b>	<b>1/1024</b>
Reserved (formerly Site-local unicast)	1111 1110 11	FEC0::/10*	* deprecated 1/1024
<b>Unique-local</b>	<b>1111 110</b>	<b>FC00::/7</b>	
Private administration	1111 1101	FD00::/8	
<b>Multicast</b>	<b>1111 1111</b>	<b>FF00::/8</b>	<b>1/256</b>

Figure 8: Allocazione dei prefissi

#### 3.4.1 Global Unicast

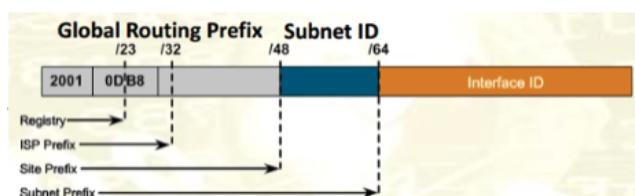


Figure 9: Global unicast address

Il prefisso Global Routing identifica il range di indirizzi allocati ad un luogo. Questa divisione è un possibile uso; l'unica cosa obbligatoria è avere una parte per la rete (subnet) e una parte per l'interfaccia. Questo boundary è 64, la divisione è fissata.

**Interface ID** Perché si hanno così tanti bit per l'Interface ID? In IPv4, ad esempio, si hanno  $2^8 - 1$  indirizzi assegnabili manualmente o automaticamente, ad esempio tramite un server DHCP.

L'idea in IPv6, è fare un'implementazione di questa operazione completamente stateless. Cosa si intende per stateless? E' un pezzo di informazione memorizzato da qualche parte, che sia sul nodo (configurazione manuale) o sul DHCP server. La fai usando il link-layer address. Non c'è uno "state" al livello 3. Il link-layer address è univoco, perché IEEE gestisce lo spazio di indirizzi. È diviso in 2 parti. L'univocità della prima parte in base al produttore, è garantito da IANA, mentre la parte rimanente dipende dal produttore. Si suppone che sia univoco, a livello link-layer.

Questa specifica è stata estesa a 64 bit, per questo in IPv6 si usano 64 bit per l'interface ID: tali bit corrisponderanno al link-layer address.

Ci sono regole per passare da 48 a 64 bit:

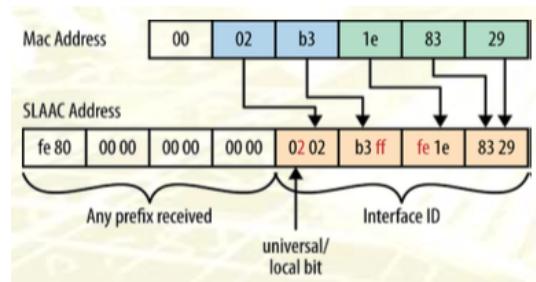


Figure 10: Conversione da 48 a 64

*SLAAC = Stateless Address AutoConfiguration*

Si aggiungono le parti rosse indicate in figura. Adesso, ogni interfaccia può comunicare il proprio Interface ID.

Si crea un problema: si può identificare il dispositivo indipendentemente da dove ti colleghi: per questo, infatti, ci sono diversi modi per generare l'ID dell'interfaccia.

**Problema della privacy** L'accesso ad Internet potrebbe essere tracciato anche tra networks, questo perché l'identificatore è univoco tra le interfacce.

- **Stable privacy addresses:** Non è basato su un identificatore hardware, viene generato in modo random. Non cambia all'interno di una subnet, ma cambia quando l'host si muove da un network all'altro.
- **Temporary transient:** Assegnato utilizzando un numero random che cambia ad intervalli regolari.

**Indirizzi link-local e local** Gli indirizzi link-local sono assegnati di default attraverso auto-configurazione. L'ID Globale di indirizzi univoci IPv6 è generato randomicamente.

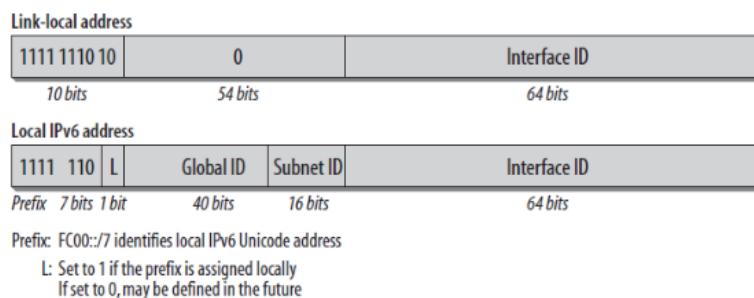


Figure 11: Indirizzi link local

I 64 bit finali sono quelli generati nei modi spiegati precedentemente (Interface ID).  
Prefisso: FC00::/7 identifica gli indirizzi locali IPv7 Unicode.  
Il bit L è settato a 1 se il prefisso è assegnato localmente e se settato a 0, potrebbe essere definito in futuro.

### 3.4.2 Indirizzi Anycast

Un indirizzo Anycast è assegnato ad interfacce multiple (tipicamente su nodi multipli). Un pacchetto inviato ad un indirizzo anycast è consegnato a soltanto una di queste interfacce (tipicamente la più vicina).

Il fatto che un indirizzo sia Anycast è una proprietà della configurazione del network, non dell'indirizzo stesso. Gli indirizzi Anycast sono presi da blocchi unicast. Da un punto di vista del sender, non importa se ci sono più indirizzi dietro un indirizzo. Per esempio, se voglio contattare un server DNS, la cosa importante è che qualcuno mi risponderà, non importa quale server lo farà.

Gli indirizzi Anycast sembrano indirizzi Unicast. Come posso assegnare un indirizzo Unicast a più interfacce? Tipicamente, su più nodi?

Supponendo di avere differenti host attaccati a differenti network, che sono tutti attaccati allo stesso network. Abbiamo un source che vorrebbe inviare il pacchetto con un indirizzo 2001::1. Assegnando questo indirizzo a host diversi, questo diventerà Anycast.

Quale host riceverà il pacchetto? Il più vicino. Questo grazie al routing: per implementare un indirizzo anycast, è necessario che sia configurato in maniera appropriata nel network.

Si ha quindi bisogno di un routing speciale? In realtà no, con IProuting si è già preparati a situazioni in cui si hanno più cammini per raggiungere la stessa destinazione. Il routing, quindi, selezionerà il migliore in quel momento.

Se gli host con lo stesso indirizzo sono 3, avremo 3 cammini possibili alla stessa destinazione virtuale (stesso indirizzo IPv6).

Abbiamo la stessa situazione con i server DNS. Qual è il problema con gli indirizzi anycast? Se c'è una richiesta DNS, questa è trasportata da pacchetti UDP, che non hanno uno stato associato alla comunicazione.

### 3.4.3 Indirizzi Multicast

Quando un pacchetto è inviato a un indirizzo multicast, tutti i membri del gruppo multicast processano il pacchetto. Un nodo può appartenere a più di un gruppo multicast.

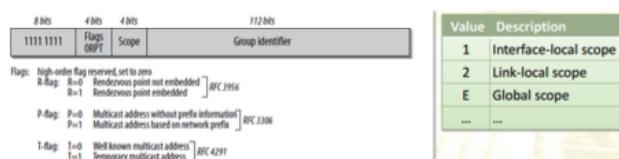


Figure 12: Multicast Address

I primi 8 bit sono tutti a 1, poi ci sono 4 bit di flag e 4 bit che indicano lo scope. Non si ha un Interface ID, ma abbiamo un identificativo per un gruppo di interfacce.

Address	Description
FF02:0:0:0:0:0:1	All-nodes address
FF02:0:0:0:0:0:2	All-routers address
FF02:0:0:0:0:0:5	OSPF/IGP
FF02:0:0:0:0:0:9	RIP routers
FF02:0:0:0:0:0:A	EIGRP routers
FF02:0:0:0:0:0:B	Mobile agents
FF02:0:0:0:0:0:1:2	All DHCP agents
FF02:0:0:0:0:0:1:4	DTCP Announcement
FF02:0:0:0:0:1:FFXX:XXXX	Solicited-node address
...	...

Figure 13: Indirizzi tipici Multicast

## 4 ICMPv6 e Autoconfigurazione

In questa sezione ci sono i veri cambiamenti rispetto ad IPv4: molti protocolli che prima erano "satelliti" ora sono parte di IPv6. Un esempio è ARP: non è una traduzione, ma un'operazione di discovery, di lookup.

### 4.1 ICMPv6

**Internet Control Message Protocol v6** Basato sulla versione ICMP di IPv4, con le dovute modifiche per adattarsi al nuovo protocollo. Fornisce funzioni diagnostiche e di gestione dell'errore.

I messaggi ICMPv6 sono incapsulati in pacchetti IPv6 con il valore *Next Header 58*. Questa nuova versione fornisce anche una nuova funzionalità denominata *Neighbour Discovery Protocol*, che offre procedure per:

- Trovare i router (Router Discovery)
- Risoluzione di indirizzi a livello collegamento (*livello 2*), (Address Resolution).
- Rilevare possibili indirizzi duplicati (Duplicate Address Detection).

Nella versione IPv4 si poteva tener traccia della raggiungibilità dei vicini solo grazie ad ARP.

#### 4.1.1 Formato del messaggio

Come già accennato, questo protocollo genera messaggi che sono trasmessi come payload IPv6 con extension number 58, che identifica ICMPv6.

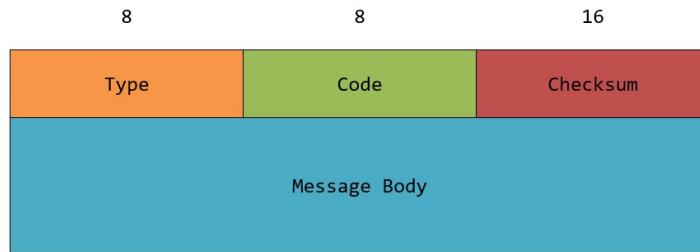


Figure 14: Formato messaggio ICMP

Nel messaggio si possono riconoscere i seguenti campi:

- **Type & Code:**
  - Type = 1, Code = 0: Destinazione non raggiungibile. Quindi, quando un router non ha una route per una certa destinazione, manda indietro al trasmettitore questo ICMPv6.
  - Type = 2: Pacchetto troppo grande. Questo può servire in un protocollo per scoprire il minimo MTU in un path; viene quindi usato nella fase di path MTU discovery e nel body del messaggio viene inserito l'effettivo MTU.
  - Type = 3: Time exceeded. Viene usato quando scade l'hop counter del pacchetto
  - Type = 4: Parameter problem. Usato quando un router rileva problemi nel formato del pacchetto.
- **Checksum:** Utilizzato per controllare se il messaggio è corrotto. Questi 2 byte sono necessari, dato che IPv6 non ha nessun meccanismo di checksum. ICMPv6 lavora ad un livello più alto di IPv6, quindi ha bisogno del checksum per controllare che il messaggio non sia corrotto.
- **Corpo del messaggio:** Qui ci va l'informazione vera e propria che trasporta il messaggio.

Message number	Message type	Description
128	Echo Request	RFC 4443. Used for the ping command.
129	Echo Reply	
...		
133	<b>Router Solicitation</b>	RFC 2461. Used for neighbor discovery and autoconfiguration.
134	<b>Router Advertisement</b>	
135	<b>Neighbor Solicitation</b>	
136	<b>Neighbor Advertisement</b>	
137	<b>Redirect Message</b>	
...		
155	Routing Protocol for Low-Power Network Messages	RFC 6550

Figure 15: Tipi di messaggio ICMP

#### 4.1.2 Neighbour Discovery protocol

E' una procedura definita interamente all'interno di ICMPv6 in modo da soddisfare varie feature:

- Address Resolution: è un set di funzioni che servono a risolvere un indirizzo IP in un indirizzo layer 2. Sostanzialmente possono verificarsi 2 casi:
- Stateless Address Autoconfiguration (SLAAC)
- Router Discovery (RD)
- Neighbour Unreachability Detection (NUD): per scoprire se qualche vicino non è più raggiungibile. Viene implementata con una cache delle traduzioni L2-L3. Ciascuna entrata ha un *lifetime* per controllare se un nodo è ancora raggiungibile.
- Duplicate Address Detection (DAD)
- Redirection

Alcune di queste funzionalità sono disponibili anche per IPv4, ma sono implementate in protocolli differenti da IP e ICMP: ad esempio address resolution implementata con ARP. Altre funzionalità, invece, sono nuove come ad esempio SLAAC ed RD.

#### 4.1.3 Router Discovery

Periodicamente, i router inviano dei messaggi di *Router Advertisement* all'indirizzo *multicast all nodes* (codice FF02::1). Questi messaggi contengono le informazioni necessarie ad un host che si vuole unire alla rete per autoconfigurarsi sulla stessa. I RA possono essere richiesti esplicitamente da un nodo tramite un messaggio *Router Solicitation*, sempre all'indirizzo multicast (codice FF02::2).

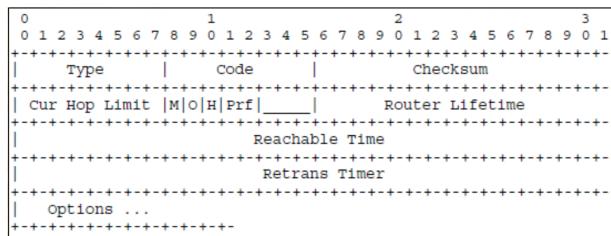


Figure 16: Messaggio router discovery

Si vanno ora ad analizzare alcuni campi all'interno del messaggio di Router Advertisement:

- Type = 134, Code = 0
- Cur Hop Limit: c'è un valore di default per hop limit. Non si può, però, usare un hop limit minore del diametro della nostra rete altrimenti, ovviamente, i pacchetti non arriverebbero al confine. Mentre, non ci sono motivi per impostare un hop limit maggiore del diametro della nostra rete.
- **Time:** parametro di configurazione.
- **M flag:**
  - 0: configurazione stateless: in cui si indica che ci deve essere una configurazione autonoma.
  - 1: configurazione stateful (DHCP): in questo caso non si può scegliere da soli il proprio indirizzo. Si utilizza il servizio DHCP.
- **Prf flag (2 bits):** Preferenza riguardo al default router. (Gateway in IPv4)
- **Options:**
  - Source link-layer address: MAC del gateway per uscire dalla rete.
  - MTU
  - Prefix information: il router inserisce il prefisso della sottorete corrente (link) in modo tale che i nodi che ricevono questo messaggio possano usare il prefisso specificato.
- Prefix information option: informazione relativa al prefix, ce n'è una per ogni prefix. Type e length specificano il tipo e la lunghezza del prefisso. Prefix length è equivalente alla mask in IPv4.
- **L flag:** on-link determination. Se L=1, un nodo sa che se un altro ha lo stesso prefisso allora può considerarlo "vicino" e quindi ci può comunicare direttamente. Se L=0 invece deve passare per forza dal router.
- A flag: se settato, questo flag indica che deve essere eseguito il Duplicate Address Detection.
- **Address lifetime:** periodo di validità dell'indirizzo.
- **Preferred lifetime:** periodo dopo il quale un indirizzo diventa deprecato.

## 4.2 Configurazione dell'indirizzo

La configurazione dell'indirizzo può essere fatta in 2 modi, come già accennato precedentemente:

- **Stateful:** DHCPv6
- **Stateless (autoconfigurazione):** Senza nessuna configurazione manuale da parte dell'host. L'interface ID è generato dall'indirizzo MAC (o scelto randomicamente). Il prefisso è imparato dal messaggio di Routing Advertisement.

L'indirizzo passa attraverso differenti stati:

- **Tentative Address (provvisorio):** L'unicità in un link deve essere verificata. Il nodo in questione manda una *Neighbour Solicitation* contenente il Tentative Address in multicast avente come groupID gli ultimi 24 bit della sua interfaceID per scoprire se esiste un altro host con lo stesso indirizzo.  
Se non riceve risposta, allora l'indirizzo è valido e passa allo stato Preferred. Un'interfaccia scarta pacchetti ricevuti che siano indirizzati ad un indirizzo provvisorio, ma accetta pacchetti di Neighbor Discovery.

- **Preferred Address:** l'utilizzo da parte di protocolli di più alto livello è illimitato. Quando scade il *preferred lifetime* l'indirizzo diventa Deprecato e quindi potrebbe essere riassegnato.
- **Deprecated Address:** l'utilizzo di questo tipo di indirizzo è sconsigliato ma non proibito.
- **Valid Address:** è un indirizzo che può essere Preferred o Deprecated.

Questo tipo di lifetime viene definito in modo da dare il tempo di controllare che non ci siano collisioni sullo stesso link.

Si vanno quindi a definire gli step per configurare un indirizzo IPv6:

- Si genera un indirizzo link-local appendendo l'identificatore di interfaccia al prefisso del link local *FE80::/10*. L'indirizzo link-local è *tentative*.
- Si manda un messaggio Neighbor Solicitation per controllare il Duplicate Address Detection. L'indirizzo link-local diventa *preferred*.
- Si manda un messaggio Router Solicitation all'indirizzo "all-routers".
- Per ogni prefisso nei Router Advertisement con il flag A settato, viene generato un indirizzo. L'indirizzo è *tentative*, dovrebbe essere utilizzato il Duplicate Address Detection.
- L'indirizzo diventa *preferred* fino a che non finisce il lifetime.

## 5 6LoWPAN

**IPv6 over Low-power Wireless Personal Area Networks** Protocollo progettato per trasportare pacchetti IPv6 nello standard 802.15.4 (*LoW Power Area Networks*) definendo uno strato posto a metà fra il livello 2 ed il livello 3 del modello ISO/OSI.

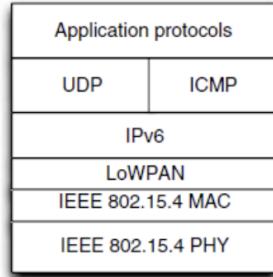


Figure 17: 6LoWPAN protocol stack

Fino ad ora, la ricerca sull'IoT ha implementato protocolli di layer 2 progettati specificatamente per efficienza energetica e nodi vincolati dal punto di vista computazionale. Attualmente, lo standard è 802.15.4, ma ci sono nuove soluzioni che stanno venendo fuori. Come tecnologia su layer 3, le ricerche hanno come obiettivo quello di piazzare IPv6 nello stack IoT, ma questo comporta alcuni problemi. Infatti, IPv6 non è compatibile con 802.15.4. Per superare questo problema, si pone un layer di adattamento nello stack di networking, tra il layer 2 e 3 in modo da fornire compatibilità tra le 2 soluzioni. Questo layer è il **6LoWPAN**.

### 5.1 Architettura 6LoWPAN

Tipi di nodi in 6LoWPAN:

- Hosts
- Routers (6LR)
- Border Routers (6LBR): hanno almeno un link attaccato ad una interfaccia fuori dalla 6LoWPAN. Possono essere link punto a punto oppure broadcast.

I 6LoWPAN sono network detti "stub", ovvero che un tipo di network che non fa passare traffico esterno attraverso essa, tutto il traffico viene generato internamente, a meno che non sia traffico esterno diretto ad un nodo della rete. Le sessioni possono essere iniziate solo da una macchina della rete.

Per quanto riguarda la topologia:

- Semplice: in cui è presente un solo Border Router.
- Extended: in cui sono presenti multipli Border Router. Devono essere connessi allo stesso link fuori dalla 6LoWPAN ed il link deve essere broadcast. Devono quindi poter comunicare fra di loro.
- Ad-hoc: non sono presenti Border Router, quindi tutto il traffico è intra-6LoWPAN.

### 5.2 Multi-hop forwarding in Low Power and Lossy Networks

Il multi-hop forwarding si può implementare su layer 2 o 3. Nell'implementazione a layer-3 i router sono normali routerIP, se il multi-hop è implementato a layer-2, questo è traparente a 6LoWPAN che quindi non se ne deve curare. Non ci sono limiti alla mobilità dei nodi all'interno della stessa 6LoWPAN, finché un nodo può parlare con almeno un router si può mantenere la comunicazione con qualsiasi altro nodo (senza cambiare indirizzo).

Un primo problema da risolvere per ottenere compatibilità tra IPv6 e 802.15.4 è che il primo assume che un subnet è associato ad un singolo link e quindi tutti gli host sullo

stesso subnet, possono comunicare fra di loro. Questa assunzione non è sempre vera un network wireless multi-hop. Per esempio, un host A potrebbe comunicare con B, B con C, ma A potrebbe non comunicare direttamente con C perché sono troppo lontani fra di loro. In questo caso, il sottostante modello di link cambia: ovvero, non c'è un singolo dominio broadcast link-layer, ma multipli domini broadcast sovrapposti. In questo tipo di scenario, per ottenere compatibilità con IPv6, un protocollo L2 per network wireless multi-hop, dovrebbe supportare un forwarding multi-hop link-layer, che 802.15.4 non fa. Ci sono quindi due soluzioni per poter ovviare a questo problema: Mesh-Under e Route-Over.

### 5.2.1 L2 forwarding (Mesh-Under)

La soluzione proposta a livello 2 è chiamata "mesh under".

Un modo possibile di implementare questo protocollo è il seguente: supponiamo di avere un frame diretto da A a C, ma C non è direttamente raggiungibile. (*Src: A, Dst: C, DATA*). Si assume, quindi, che C sia raggiungibile tramite un certo B che è raggiungibile anche da A. Quindi, si incapsula il frame in un altro header contenente B come indirizzo di destinazione. (*Src: A, Dst: B, Src: A, Dst: C, DATA*).

Quando B riceve il frame, rimuove il secondo header e controlla il primo, determinando che il pacchetto è rivolto verso C, quindi lo incapsula in un nuovo header e lo manda verso C. (*Src: B, Dst: C, Src: A, Dst: C, DATA*).

Finalmente, C riceve il pacchetto, rimuove l'header e controlla che il pacchetto sia da A verso C e passa i dati al livello superiore.

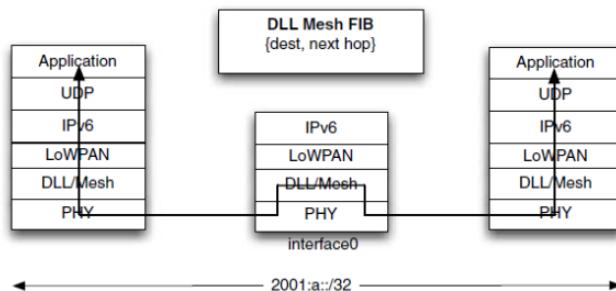


Figure 18: Mesh Under protocol stack

Ovviamente, la scelta del prossimo hop per una certa destinazione, è determinata sulla base di un algoritmo di routing specificato a priori. Si crea un problema che riguarda la mancanza di interoperabilità tra L2 ed L3, quindi la riparazione delle rotte sono lente.

- **Discrepanza topologica tra L2-L3:** a livello 2 abbiamo lo *spanning tree protocol* che gestisce le rotte. Essendo in un contesto wireless (e non cablato) non abbiamo la possibilità di fare controlli elettronici diretti per verificare lo stato dei nodi (che nelle LLNs possono spostarsi, andare in standby ecc). Segue che l'algoritmo non rileva immediatamente i cambiamenti di topologia nella rete e quindi ci sarà una discrepanza tra la topologia vista a L2 e quella a L3.
  - le riparazioni delle rotte a L2 sono lente
- Non c'è un protocollo efficace che possa sostituire lo spanning tree.
- Si potrebbe usare una tabella di routing a L2? In teoria sì, in pratica non c'è un protocollo efficace che possa generare questa tabella.

Alcuni di questi concetti verranno ripresi nei paragrafi successivi.

### 5.2.2 L2 forwarding vs IP model

Definizione di link: un'area topologica limitata da routers (in cui la comunicazione è regolata da un protocollo L2) che decrements l'IPv6 Hop Limit quando si fa il forwarding dei pacchetti.

Definizione di subnet: un'area topologica che utilizza lo stesso prefisso nell'indirizzo, dove quel prefisso non viene ulteriormente suddiviso eccetto per quanto riguarda indirizzi individuali.

Nel modello IPv6: una subnet è associata con un link e subnet multiple potrebbero essere associate allo stesso link. I protocolli L2 dovrebbero essere progettati per realizzare un modello di link consistente con il modello IP.

Il modello a cui si deve far riferimento è quello multi-access, in quanto si hanno più nodi che devono accedere allo stesso link, pacchetti mandati ad un indirizzo link-local multicast possono essere ricevuti da tutti i nodi interessati in quel link. Due nodi sul link devono poter comunicare senza nessun decremento dell'hop limit dell'IPv6. Deve quindi essere tutto trasparente ad IPv6.

**Switched Ethernet** In Ethernet si ha un esempio di comunicazione cablata multi-hop, fornisce a IP un modello di link multi-access. Viene utilizzato uno spanning tree protocol per implementare un modello multi-access. Questo protocollo opera indipendentemente dalla funzione di forwarding, l'albero creato è un sottoinsieme dei link e questo protocollo viene utilizzato per eliminare loop non graditi. Esiste solo un path per raggiungere la destinazione e viene deciso dal protocollo in fase di creazione.

In caso di failure di un nodo, lo spanning tree viene ricreato e ci vogliono pochi secondi per poter ricreare una nuova topologia. Con questo protocollo, si può essere sicuri che gli eventuali messaggi broadcast inviati, arriveranno una sola volta a ogni nodo.

**Multi-hop wireless network** In questo caso, se un nodo trasmette un pacchetto, questa trasmissione sarà ricevuta da tutti i pacchetti nel range della comunicazione, con una sola trasmissione. Quindi in questo caso, a differenza di ethernet, si parla di broadcast medium. Con questa premessa, con broadcast medium, si utilizza una topologia a multicas tree: a differenza di uno spanning tree (che va a coprire tutta la rete), un multicast tree copre solo un sottoinsieme di tutti i nodi della rete, i vari sottoinsiemi di nodi organizzati in multicast tree devono comunicare tra loro per coprire tutta la rete.

Si deve considerare in questo caso, che la topologia di rete varia nel tempo in modo impredibile. I nodi possono essere statici, ma i confini, il range di comunicazione può variare nel tempo a causa di interferenze interne o esterne.

Si può implementare facilmente il modello IPv6 (mesh under), ma ci sono alcuni contro in questo caso: le riparazioni a L2 sono lente, in ethernet ci si può accorgere subito di un link malfunzionante tramite controlli elettronici, con la comunicazione wireless questo diventa più difficile e non se ne può accorgere immediatamente, ma dopo un determinato numero di trasmissioni non andato a buon fine. Essendoci una discrepanza da una rete vista a L2 e a L3, si crea un problema in caso di fallimento dei nodi.

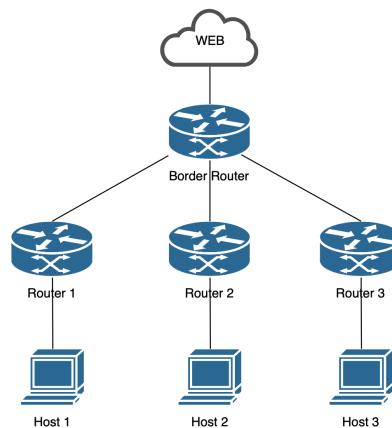


Figure 19: Topologia a L2

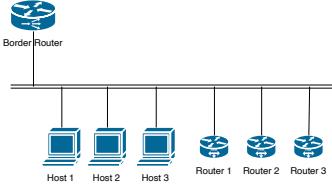


Figure 20: Topologia a L3

### 5.2.3 L3 forwarding (Route-Over)

In questo caso, le decisioni di routing vengono prese in base alle informazioni di IPv6 (quindi a livello 3), non ci si aspetta supporto dal livello 2. In 6LoWPAN il routing viene fatto a livello 3, usando delle estensioni per implementare quelle funzioni che IPv6 considera svolte solamente a livello sottostante. Siccome vogliamo implementare il routing *multi-hop* a livello 3, abbiamo bisogno di tutte le informazioni (per esempio gli extension header). Se il pacchetto è frammentato allora devo aspettare tutti i frammenti per ricomporlo e solo allora avrò tutte le informazioni necessarie per inoltrarlo. Prima di fare ciò bisognerà ovviamente ri-frammentare il pacchetto. Quindi si crea un alto overhead per questo tipo di approccio.

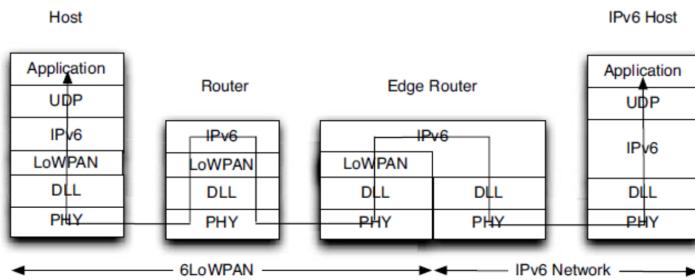


Figure 21: Route-over protocol stack

### 5.2.4 L3 forwarding vs IP model

Le domande che vengono poste a questo punto sono: il modello IPv6 funziona? Il modello per cui, una subnet è associata ad un solo link. E poi, qual è il sottostante modello link? Non può essere un modello punto-punto e nemmeno un multi-access, in quanto, nel secondo caso, non ci sarebbe modo di capire cosa si intenda per "stesso link". Un 6LoWPAN è composto da domini di broadcast multipli e sovrapposti. Quindi, l'approccio IP tradizionale non può più essere applicato in questo contesto.

## 5.3 Modello link 6LoWPAN

In 6LoWPAN non viene fatta nessuna assunzione sul link model (undetermined link model), ogni nodo della subnet condivide lo stesso prefisso.

Ogni nodo possiede un link-local scope, che è formato dai nodi che possono comunicare direttamente con esso, ovvero le interfacce che si trovano nel range del segnale radio. In questo modo, nella stessa subnet ci possono essere differenti link-local scope. Quindi, due nodi nella stessa subnet possono condividere lo stesso prefisso, ma avere un link local scope la cui intersezione è nulla. Quindi i nodi nella 6LoWPAN, condividono lo stesso global IPv6 prefix. Questo implica che non si può più utilizzare il prefisso del nodo per inoltrare un pacchetto, perché adesso lo stesso prefisso non significa più stesso link, quindi un nodo con un certo prefisso potrebbe anche non poter raggiungere il nodo destinatario che possiede lo stesso prefisso. Neighbour discovery deve essere esteso e il DHCP non funziona.

## 5.4 Addressing

I prefissi IPv6 sono ottenuti attraverso il Router Advertisements.

Gli interface ID IPv6 sono ottenuti in accordo alla procedura *Stateless Address Autoconfiguration (SSA)*. E' obbligatorio avere un mapping tra il link-layer (layer 2) e l'indirizzo IPv6

(layer 3), in questo modo non c'è bisogno di fare address resolution e otteniamo una compressione migliore dell'header. Infatti, l'indirizzo a livello 2 è uguale ad un pezzo dell'indirizzo IPv6. Si ottiene quindi una buona compressione, mettendo tale parte nell'indirizzo.

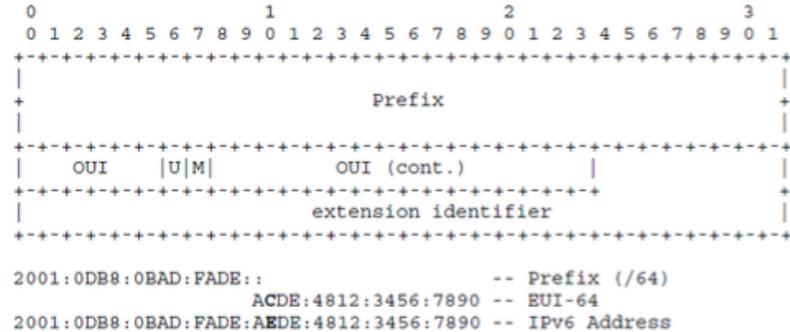


Figure 22: 64-bit IEEE EUI-64 link layer address



Figure 23: 16-bit short address

## 5.5 6LoWPAN adaptation layer

La dimensione di MTU in 802.15.4 è piccola (127 bytes) per far fronte alle limitate capacità dei buffer e per limitare il rate di errori dei pacchetti.

Considerando l'overhead del protocollo, nel caso peggiore rimangono solo 81 bytes per il payload. E' necessario quindi un adaptation layer per supportare la frammentazione e il riassemblamento (il minimo MTU di IPv6 è 1280 bytes), implementare la compressione dell'header e ridurre l'overhead del protocollo IPv6.

### 5.5.1 6LoWPAN header stack

L'header in 6LoWPAN è organizzato come uno stack (come in IPv6), ma con meno flessibilità. Le combinazioni possibili sono ben definite dallo standard:

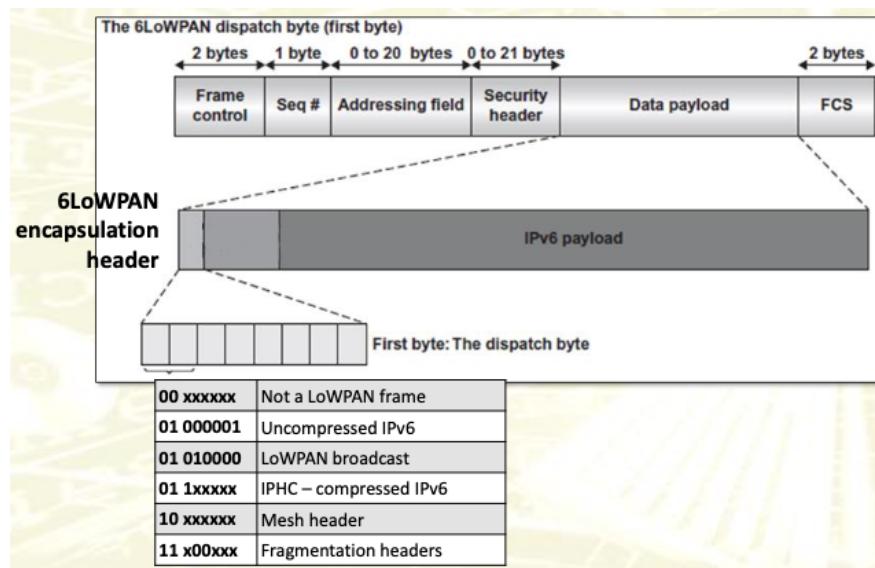


Figure 24: 6LoWPAN header stack

### FCS: Frame Sequence Check

Guardando i primi 2 bit si capisce come guardare il resto del payload, in questo modo si definiscono tutte le possibili 8 combinazioni dei bit.

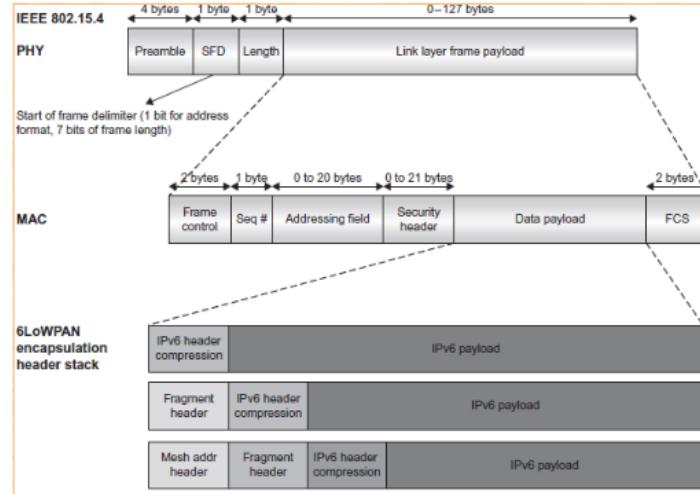


Figure 25: 6LoWPAN header stack

#### 5.5.2 Mesh addressing layer

6LoWPAN supporta il forwarding mesh a livello 2 quando il link-layer utilizza il multi-hop. Ma non è definito nessun protocollo di routing. *Mesh-Under*.

Se usato, il Mesh Header deve essere il primo in cima allo stack visto che contiene l'indirizzo necessario per fare il forwarding del pacchetto.

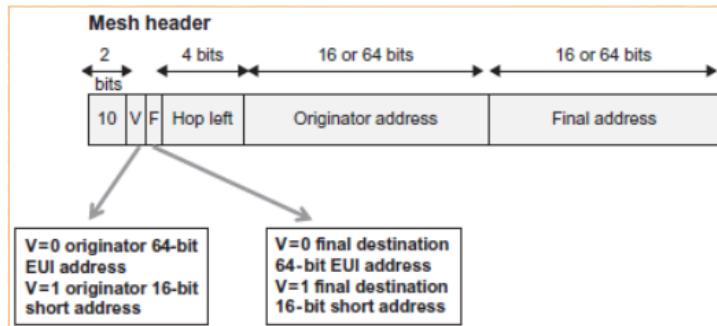


Figure 26: Mesh header

Nel caso di broadcast Mesh Under, se non si usa uno spanning tree (e quindi possono verificarsi loop), si ha bisogno di un numero di sequenza che si trova nel link layer, tale che se manda un pacchetto broadcast con un certo numero di sequenza e lo si riceve con lo stesso numero, lo si scarta. (non è presente la corrispondente immagine per il Broadcast Header)

#### 5.5.3 Fragment header

È necessario quando il payload originale IPv6 non può essere trasportato in un singolo frame 802.15.4 perché eccede le dimensioni di MTU.

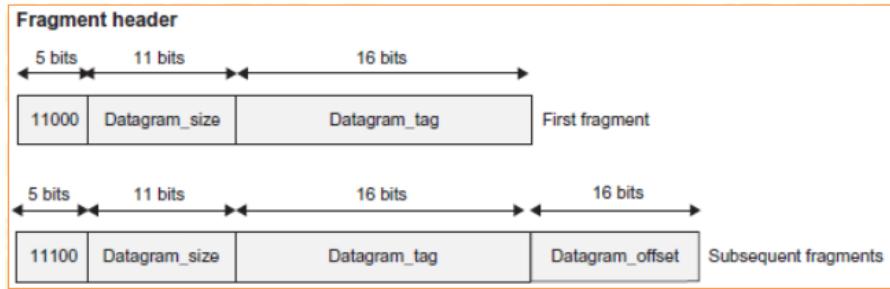


Figure 27: Fragment Header

Per ogni header successivo al primo, si inserisce l'offset per identificare il frammento relativo al pacchetto originale.

La dimensione del datagram è importante in quanto non è detto che i pacchetti arrivino nell'ordine corretto e quindi avere la dimensione del datagram in ogni frammento, consente di allocare memoria qualunque sia il primo pacchetto ricevuto.

#### 5.5.4 Header compression

Sono state definite tecniche di compressione IP stateful che si basano sull'ottimizzazione di flow a lunga durata (lifetime). Queste tecniche però, non sono compatibili con i network 6LoWPAN. In generale, ad esempio in uno streaming video, i pacchetti vanno da una sorgente a una destinazione; in questo caso alcuni campi dell'header vengono mandati tutti insieme (nello stesso flusso). Nel caso attuale, questo approccio non va bene, si vuole una compressione pacchetto-pacchetto ed è sufficiente effettuare una compressione solo nella capillary network, quando il pacchetto arriva al Border Router, non serve più che sia compresso.

**Stateless Header Compression** Nell'header base IPv6, facendo alcune assunzioni, si possono rimuovere alcuni campi.

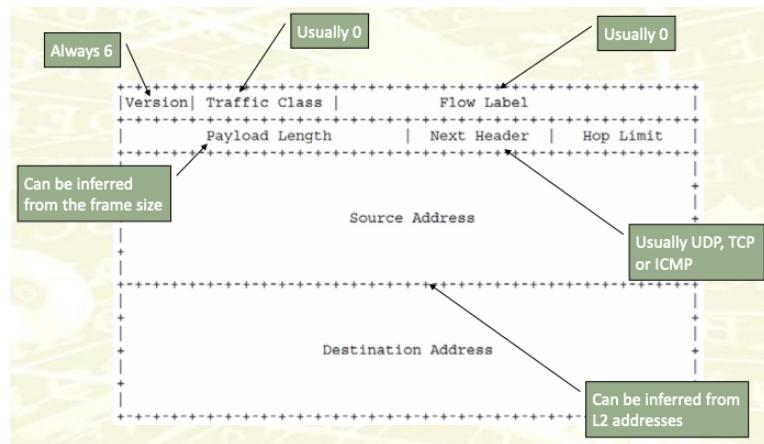


Figure 28: Header compression

Il tema è che si rimuove e non si trasmettono campi il cui contenuto può essere identificato in qualche modo (ridondanti):

- Il campo *Version* è sempre 6.
- *Traffic Class* è spesso 0.
- *Flow Label* tipicamente non viene considerato.
- Il campo *Next Header* è tipicamente UDP, TCP o ICMP.
- Il campo *Payload Length* può essere ricavato da il frame di layer 2.

- I campi *Source Address* e *Destination Address* possono essere ricavati dal frame di layer 2.

Questo approccio stateless, però, non è ritenuto soddisfacente per comunicazioni oltre i link locali, dato che per pacchetti che devono uscire dalla 6LoWPAN, soltanto 8 di 32 bytes possono essere salvati. Questo approccio, quindi, tipicamente non viene usato.

**Context-based Header Compression - IPHC** Si utilizzano contesti condivisi tra source e destination per ottenere una compressione migliore. 6LoWPAN, quindi, sostituisce l'header IPv6 con il seguente header compresso:

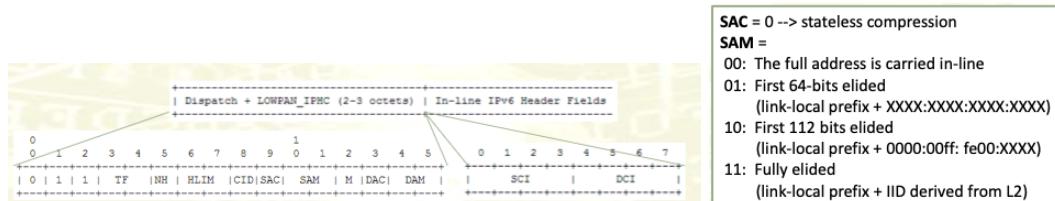


Figure 29: Stateless Header Compression IPHC

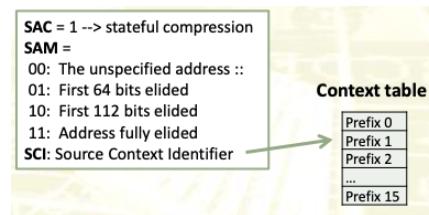


Figure 30: Stateful Header Compression IPHC

- Il campo *TF* specifica se *Traffic Class* e *Flow Label* hanno valori di default; se non ne hanno i loro valori sono passati come *In-line IPv6 Header Fields*.
- NH* specifica la solita cosa per il campo *Next Header*.
- HLIM* specifica possibili valori per *Hop Limit*
- Source Address Compression (SAC)* specifica quando avviene una compressione statefull o stateless.
- Source Address Mode (SAM)* specifica quanti bit sono stati toliti (elided) e quanti altri sono passati "in-line".

C'è un ulteriore campo *SCI* (*Source Code Identifier*) che punta allo slot della Tabella di Contesto contenente il prefisso cercato.

In caso di compressione stateful, il prefisso può essere trovato in una tabella di contesto (di 16 entries) condivisa tra il sender e il receiver. Il Border Router diffonde questa informazione nella rete. Occorre semplicemente specificare l'ID del contesto, non l'indirizzo. Si parla, quindi, di stateful perché in diverse 6LoWPAN possiamo avere diverse context table.

## 5.6 6LoWPAN Neighbour Discovery (ND)

La Neighbour Discovery in multicast non è efficiente poiché i nodi potrebbero essere in *standby* per risparmiare energia. Se stiamo usando una *route-over solution* (quindi abbiamo effettivamente più link che si sovrappongono), con la procedura di ND classica posso controllare i duplicati solo all'interno del raggio di trasmissione.

Per questo motivo la ND è stata ri-progettata per il 6LoWPAN in modo che ceda della flessibilità in cambio di semplicità. E' stato ridisegnato il modo in cui interagiscono router e host in modo tale che fosse indipendente dal tipo di soluzione adottata (mesh-under o route-over). Qui usiamo un approccio ottimistico: si assume che non ci siano duplicati (quindi non viene eseguito alcun controllo sul DAD - Duplicate Address Detection).

- Non si ha broadcast degli advertisement: se vogliamo sapere qualcosa riguardo i nostri vicini dobbiamo inviare un messaggio di *Router Solicitation*.
- Un host trasmette un *Router Solicitation* all'indirizzo multicast link-local all-router.
- Ciascun router risponde con un messaggio *Router Advertisement* in unicast.
- Informazioni utili a tutta la rete 6LoWPAN (prefissi & info contesto) vengono diffuse (flooding) dal cosiddetto Border Router (6LBR).

### 5.6.1 Address Configuration

Lo scambio di messaggi RA e RS avviene tra gli host ed i nodi vicini (entro il raggio di trasmissione). Nella versione originale del Neighbour Discovery ogni router invia periodicamente un RA. In 6LowPAN questo crea troppo overhead, per cui ogni router invia un RA solo quando riceve un RS message da un host. Quindi, si evince che un host invia un RS message per scoprire quali sono i routers nel suo raggio di comunicazione. Il RA è unicast in modo che l'unico nodo che lo processi sia l'host che ha inviato il RS message.

All'avvio, i router si comportano come host per raccogliere la configurazione iniziale, inoltrando i messaggi ai nodi più distanti dal Border Router. Il BR propaga le informazioni (prefissi e context) a tutta la rete. I messaggi che arrivano dal BR possono essere marcati con un'opzione aggiuntiva che serve per indicare il livello di credibilità del messaggio (*Authoritative Border Router Option*). Un messaggio del BR per la configurazione di rete avrà priorità maggiore rispetto alle informazioni salvate sui router. Questo serve per poter correggere eventuali errori sui router.

### 5.6.2 Address Registration

E' un'altra opzione trasportata dai messaggi RS per le reti LLN (low-power lossy networks) che permette di garantire affidabilità e risparmio energetico.

Alcuni nodi possono essere mobili quindi, mandano un avviso al router per comunicargli la loro presenza (approccio pro-attivo). Il router mantiene una lista dei nodi che può raggiungere costantemente, con indirizzo a L2 ed L3. In questo modo un router non ha bisogno di verificare la raggiungibilità di un host ogni volta che vuole iniziare una comunicazione.

A questo punto è compito del router assicurarsi che non ci siano indirizzi duplicati (DAD). Tuttavia, ci possono essere diversi router (e quindi diverse liste di nodi). Come si può risolvere il problema del DAD in questo caso? Si può risolvere tramite una Tabella DAD centralizzata presso il Border Router. Per supportare questa funzionalità vengono introdotti due nuovi messaggi ICMPv6:

- Duplicate Address Request (DAR)
- Duplicate Address Confirmation (DAC)

In questo modo, quando un host vuole registrarsi presso un router, quest'ultimo verificherà presso il BR se l'id dell'interfaccia di rete dell'host è già presente nella Tabella. In caso contrario, procede alla registrazione.

Nel caso multi-hop, la registrazione avviene attraverso i seguenti passaggi:

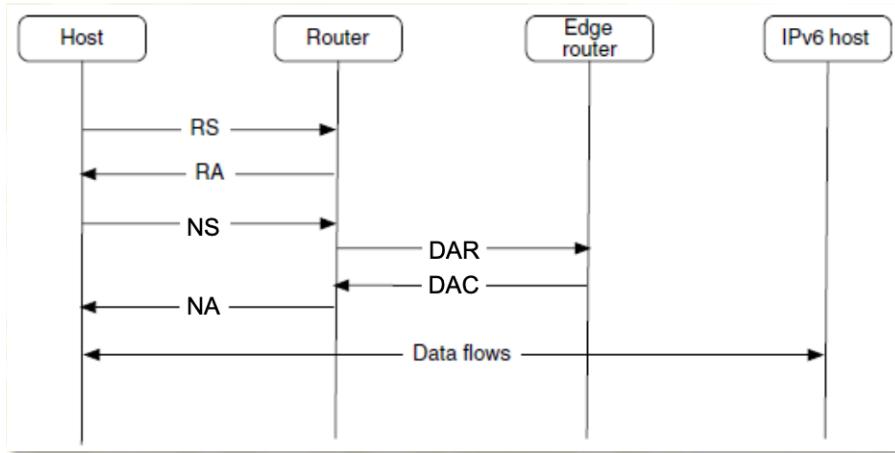


Figure 31: Multi-hop registration

Si va quindi a fare una richiesta alla repository per eventuali duplicati, quest'ultima tiene traccia dei nodi nel network.

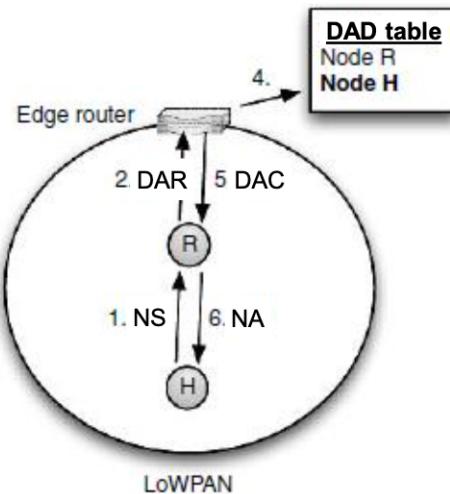


Figure 32: DAD table

Non viene più effettuata la address resolution all'interno dell'host.  
Per quanto riguarda la trasmissione dei pacchetti:

- Indirizzi considerati *on-link*: link-local unicast (prefix fe80::) o multicast (multicast link-local address).
- Tutti gli altri prefissi sono considerati *off-link*.

Per quanto riguarda la "address resolution": la address registration fornisce supporto per il mantenimento dello stato "host-to-router". La risoluzione per indirizzi basati su multicast non è necessaria.

## 6 RPL

### 6.1 Premessa

I tradizionali protocolli di routing, come RIP o OSPF, non sono più adatti per l'IoT, perché ci sono molti nuovi fattori da considerare:

- Undetermined Link Model: i protocolli di routing tradizionali, sono progettati per tradizionali modelli IP (ipotesi 1 link-1 subnet). Se viene utilizzata una soluzione route-over, questa ipotesi non vale più e il protocollo di routing si deve adattare conseguentemente.
- Nuove metriche di costo, non più soltanto il cammino più corto: ad esempio si può considerare anche la percentuale di perdita di comunicazione nel mezzo, efficienza energetica, e vincoli di ritardo.
- Adattabilità a cambiamenti dinamici nella topologia del network.
- Bisogno di cammini multipli verso una destinazione, per migliorare la robustezza della comunicazione.
- Nodi limitati dal punto di vista computazionale.

### 6.2 Concetti generali

Protocollo di routing L3 per le LLNs (low-power lossy networks). Basato sull'algoritmo *Distance-Vector*, introduce alcune ottimizzazioni oltre alla nozione dei vincoli di routing ovvero la possibilità di non considerare solo il percorso a costo minore quando si fa routing ma imporre anche dei vincoli ai percorsi di rete.

In questo protocollo supponiamo che i router non si scambino informazioni sullo stato dei link: nessuno ha conoscenza della topologia completa della rete, si conosce indirettamente sapendo la distanza dalle varie destinazioni.

RPL costruisce un DODAG (Destination-Oriented Directed Acyclic Graph) ovvero un grafo in cui gli archi sono direzionati ed è privo di cicli. Ciascun nodo ha almeno un arco uscente. Il motivo per il quale si è scelto di implementare un grafo piuttosto che un albero è molto semplice: vogliamo che ci siano più percorsi alternativi verso un determinato nodo. In questo modo si può sopprimere ad eventuali malfunzionamenti dei nodi.

Nel caso in cui si abbia una rete 6LoWPAN composta da più Border Router (entrambi connessi ad una stessa *backbone* - in questo modo si avrà un unico DODAG virtuale), i nodi vengono divisi in n sottoinsiemi dove n = numero di BR. Ogni nodo può appartenere ad un solo sottoinsieme (DODAG) ed eventuali link che attraversano i sottoinsiemi non saranno più utilizzabili.

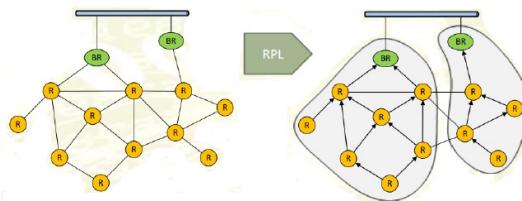


Figure 33: Multi-sink DODAGs

### 6.3 Istanze RPL

In una rete 6LoWPAN si possono avere differenti tipologie di traffico con differenti esigenze/requisiti. Per esempio, ci possono essere messaggi critici che devono essere consegnati il più in fretta possibile in maniera affidabile, oppure messaggi poco utili, che possono anche essere persi o interpolati e, pertanto, devono essere mandati consumando meno energia possibile. Per questo motivo nella stessa rete possiamo avere differenti istanze di RPL con differenti tabelle di routing. Per ogni istanza RPL quindi, avremo un DODAG distinto che,

all'interno della stessa istanza avrà le solite metriche di routing e gli stessi vincoli. Alcuni esempi di vincoli/metriche sui nodi sono:

- Stato del nodo e attributi.
- Energia (batteria) del nodo.
- Numero di hops.

Per i link invece avremo:

- Throughput.
- Latenza.
- Affidabilità del link.
- Protocollo di routing usato nel link.

#### 6.4 Funzione Obiettivo (FO)

La FO definisce quali metriche/vincoli usare per trovare i percorsi a costo minimo in un'istanza RPL. Per realizzare ciò, i nodi devono conoscere sempre quali sono i suoi genitori (ovvero conoscere la loro posizione all'interno della rete) e il loro *grado/rank* (uno scalare che indica la distanza di un nodo dal BR). Grazie al grado di un nodo, si può sapere se è genitore o figlio di un altro nodo.

Il Rank, nel dettaglio:

- Deve decrescere monotonicamente su ogni cammino verso il BR; in altre parole, il rank di un nodo è sempre più grande del rank dei suoi nodi genitori nel grafo.
- Quando un nodo si unisce al network, riceve i messaggi dai suoi vicini che contengono i loro rank, quindi sceglie alcuni parenti tra questi vicini e deve settare il suo rank in modo che sia più grande del rank dei parenti.
- Il rank fornisce un ordine parziale tra i nodi del grafo, quindi dati due nodi A e B sullo stesso cammino verso il sink, il nodo A è più vicino del nodo B se  $\text{rank}(A) < \text{rank}(B)$  (però se i due nodi non sono sullo stesso cammino verso il sink, non possono essere tratte conclusioni a riguardo, per questo si parla di ordinamento parziale).
- Come viene calcolato è definito dalla Funzione Obiettivo.
- I rank possono essere usati per evitare cicli.
- Non sono intesi come una misura del costo di un cammino verso il root: un nodo potrebbe avere più di un cammino verso il root, quindi una singola misurazione come il rank non è adatta a rappresentare il costo di un cammino. In più, è meglio mantenere queste due metriche separate; infatti, il costo del percorso è una metrica che potrebbe variare frequentemente, mentre il rank è una misura che vorremmo rimanesse stabile.

#### 6.5 Formazione del DODAG

Il Root DODAG comincia facendo advertising della sua presenza mandando dei messaggi DIO (*DODAG Information Object*) sull'indirizzo link-local multicast; questi messaggi contengono differenti parti di informazione, tra i quali:

- Identificatore dell'istanza DODAG
- L'identificatore DODAG nell'istanza RPL
- Istanza RPL dell'objective function
- Rank del sender
- ecc...

I nodi RPL ascoltano i messaggi DIO e li inoltrano per fare l'advertising della loro presenza e propagano le informazioni. Ricevendo un messaggio DIO, un nodo RPL impara il set di nodi nel vicinato con al più 1 hop. In questo set, basandosi sull'istanza RPL della funzione obiettivo, il nodo:

- Determina un set di vicini candidati tra cui scegliere i genitori.
- Sceglie un genitore preferito da impostare come percorso predefinito verso il DODAG root.
- In base al grado ricevuto dai genitori, il nodo determina il suo grado personale.

I messaggi DIO sono periodicamente mandati in broadcast nella rete per mantenere le informazioni di routing aggiornate. La propagazione dei messaggi DIO viene gestita dall'algoritmo *Trickle*.

### 6.5.1 Algoritmo Trickle

Trickle è un algoritmo di tipo gossip che consente di propagare informazioni con un flooding controllato.

Questo viene ottenuto tramite due tecniche:

- **Broadcast Suppression:** vengono selezionati solo alcuni nodi che ritrasmettono, mentre gli altri non possono.
- **Adaptive Periodicity:** modifica la frequenza che indica ogni quanto vengono mandati i messaggi DIO.

Ciascun nodo ha un contatore  $c$  ed un timer  $\tau \in [\frac{I}{2}, I]$ , quando il timer scade un nodo fa il broadcast del messaggio: ogni nodo conta il numero di messaggi *consistenti* che vengono ricevuti nell'intervallo corrente e quando il timer scade, il nodo attiva il broadcast solo se il numero è minore di un *threshold di ridondanza*.

Nello specifico, vediamo il **funzionamento di Trickle**:

1. Inizialmente  $I = I_{\min}$ ,  $c = 0$ ,  $\tau \in [\frac{I}{2}, I]$  e aspetto i metadati.
2. Se i metadati sono consistenti (rispettano le regole dei gradi/rank):
  - (a)  $c++$
  - (b) al tempo  $\tau$ , se  $c < K$ : il nodo invia in broadcast i metadati.
  - (c) quando l'intervallo  $I$  scade:
    - i.  $I = 2I$  fino ad un massimo di  $I_{\max}$
    - ii.  $c = 0$
    - iii.  $\tau = \tau' \in [\frac{I}{2}, I]$
3. altrimenti (metadati inconsistenti), resetto tutto.

La costante  $K > 0$  è detta soglia di ridondanza e indica il numero massimo di messaggi DIO che possono essere ascoltati in un determinato intervallo (misurato da Trickle). Oltre questa soglia si ha la soppressione dei messaggi.

**Inconsistenza di Routing** Trickle utilizza il concetto di messaggi *consistenti* e *inconsistenti*. Nel nostro caso, i messaggi inconsistenti si riferiscono a cambiamenti nella topologia del network. Il sink potrebbe rilevare inconsistenza di route, in quel caso può mandare nuovi messaggi DIO per costruire un nuovo DODAG.

I messaggi DIO includono un altro pezzo di informazione: la versione DODAG. Quindi, i nuovi messaggi DIO includono un nuovo numero della versione DODAG; inizia così la procedura per unirsi ad un nuovo DODAG comportandosi come se avesse ricevuto un nuovo

messaggio inconsistente, resettando l'intervallo Trickle, per consentire una costruzione veloce di un nuovo DODAG.

Inoltre, un nodo stesso potrebbe autonomamente rilevare inconsistenze di route: ad esempio, potrebbe osservare che il rank di un parente è cambiato e se il nuovo rank è maggiore del proprio rank, si è rilevata un'inconsistenza. In questo caso, il nodo si adatta alla nuova topologia e l'intervallo di Trickle è resettato. Informazioni riguardanti il rank del nodo possono essere incluse nel messaggio generato dal nodo, inserendolo nell'header IPv6 nell'opzione *hop-by-hop*.

**Rilevazione dei loop** Può essere eseguito dai nodi mentre osservano inconsistenze nel rank dei parenti. Considerando la seguente situazione d'esempio:

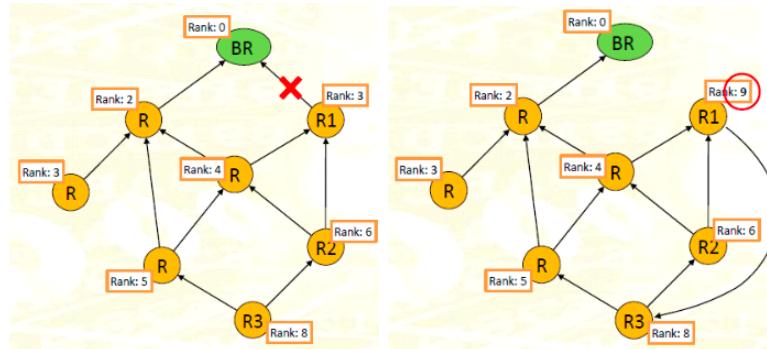


Figure 34: Esempio topologia in loop

In questo esempio, il link tra BR ed R1 si rompe. Quindi, R1 si aggancia ad R3, cambiando il suo rank a 9. Tuttavia, R1 è ancora parente di R2, che è, a sua volta è parente di R3: si è formato un loop.

In questa condizione, il loop può essere rilevato quando R2, per esempio, manda un messaggio al root, utilizzando R1 come prossimo hop. R1, infatti, riceverà il messaggio in cui il rank sarà più piccolo del suo; questa però risulta essere un'inconsistenza perché messaggi che attraversano un cammino verso il root dovrebbero sempre trovare rank decrescenti.

Si crea anche un problema di "conteggio all'infinito": dall'esempio precedente, R1 inizierà a mandare i messaggi DIO con il suo rank aggiornato; questi messaggi saranno ricevuti da R2. Quest'ultimo non sa che è presente un loop e andrà ad incrementare il suo rank in modo da mantenere la regola che il rank del parent deve essere più piccola di quella dei figli. A sua volta, R2 inizierà a mandare messaggi DIO, con il suo rank aggiornato, che verranno ricevuti da R3. Questo, farà la solita cosa, incrementando il rank e mandando ancora messaggi DIO. Questi messaggi saranno ricevuti da R1 che incrementerà ancora il rank.

Questa procedura si ripeterà all'infinito e i nodi continueranno a incrementare il rank in maniera indefinita.

I loop possono essere rotti e il problema del conteggio all'infinito può essere evitato andando a definire un valore *Max Depth*: i rank non potranno essere più grandi di questo valore e quando un nodo vede che il suo parent ha raggiunto il massimo valore del rank, che potrebbe accadere dopo che si è verificato il fenomeno sopracitato, si stacca dal parent, rompendo il possibile loop.

Il "conto all'infinito" può essere evitato imponendo che il nodo setti il suo rank al massimo valore appena questo osserva che un cammino verso il root è fallito (questo è simile al "root poisoning", ovvero quando un router setta il costo verso una destinazione a infinito). Nel precedente esempio, R1 dovrebbe settare il suo rank al valore massimo quando il link verso BR fallisce.

**Global Repair** Il BR nota che c'è un'inconsistenza nella rete e invia messaggi DIO con numero di versione DODAG diverso rispetto a quello attuale. In questo modo i router capiscono che la rete deve essere resettata.

**Local Repair** La riparazione delle inconsistenze viene fatta localmente dai singoli router (come abbiamo visto per la loop detection).

## 6.6 Comunicazioni Point-to-Point e Point-to-Multipoint

Fino ad ora abbiamo considerato solo comunicazioni da nodi verso la radice del DODAG. In RPL è possibile avere comunicazioni **punto-a-punto** (messaggi dalla radice ai nodi) oppure **punto-a-multipunto** (da un nodo ad un altro in un altro DODAG). Queste caratteristiche sono supportate da messaggi speciali chiamati **DAO - Destination Advertisement Object**, che sono usati dai nodi per avvisare che sono raggiungibili tramite un certo percorso. Esistono due modalità di implementazione:

**Storing mode** Assumiamo che i router che compongono il DODAG abbiano abbastanza memoria per memorizzare una tabella di routing.

1. Messaggi DAO contengono indirizzo IPv6 del nodo sorgente e sono mandati in unicast verso i nodi "genitori".
2. Un nodo che riceve un DAO aggiunge un'entrata alla sua tabella di routing che associa l'indirizzo contenuto nel DAO al nodo "figlio" dal quale il messaggio è arrivato. Il nodo "figlio" sarà quindi un *next-hop* per la destinazione specificata nel DAO.
3. Il DAO viene inoltrato in su per l'albero fino alla radice.

**Non-Storing mode** Non avendo memoria a disposizione per la routing table, memorizziamo le stesse informazioni all'interno dei pacchetti.

1. Tutti i messaggi DAO convergono alla radice. Ogni messaggio contiene la lista dei nodi traversati lungo il loro percorso per arrivare alla radice. In questo modo il nodo root conosce tutti i possibili percorsi verso una qualsiasi destinazione.
2. Il traffico Point-to-Multipoint viene indirizzato dalla root verso il percorso desiderato.
3. Il traffico Point-to-Point viene mandato al nodo root e poi indirizzato come nel caso P2MP.

## 7 CoAP

### 7.1 Concetti generali

Constrained Application Protocol, è un protocollo web generico orientato a requisiti specifici (esempio quelli delle LLNs). Il suo scopo principale è quello di realizzare un protocollo applicazione (L5) per servizi web (in comune con HTTP) ma ottimizzato per le applicazioni *Machine-to-Machine (M2M)*. Non si usa HTTP perché è troppo pesante per i nodi IoT. Il livello CoAP è intermedio fra UDP e il livello 5 (applicazione) composto da due sottolivelli: uno gestisce le richieste/risposte, l'altro fornisce il supporto allo scambio di messaggi asincroni. I messaggi asincroni servono per poter inviare un messaggio al client anche quando questo non ha effettuato alcuna richiesta e questo non è possibile in TCP in quanto serve una connessione tra i due peer.

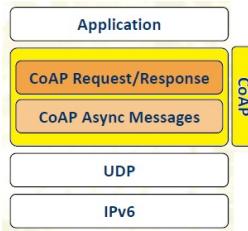


Figure 35: Posizione del livello CoAP

CoAP utilizza un modello di interazione Request/Response (simile ad HTTP) asincrona, con alcune opzioni per l'affidabilità.

Divide il layer applicazione in 2 sotto-layer:

- Il top layer è il *Request/Response Layer*: i messaggi sono codificati come richieste o risposte (similarmente a HTTP) e specificano l'identificatore della risorsa target e ulteriori informazioni.
- Sotto questo layer, è presente il *Asynchronous Message Layer*, che fornisce 4 tipi di messaggi in cui le richieste e risposte possono essere avvolte:
  - *Confirmable Messages (CON)*: richieste e risposte che richiedono un acknowledgement.
  - *Non-Confirmable Messages (NON)*: richieste e risposte che non richiedono acknowledgement.
  - *Acknowledgement Messages (ACK)*
  - *Reset Messages (RST)*

Una trasmissione non-affidabile viene iniziata dal mittente che invia un messaggio NON al quale non riceverà risposta. Una trasmissione affidabile invece, viene iniziata tramite l'invio di un messaggio CON. Il ricevente può:

- Rispondere con ACK oppure
- Rifiutare con RST.

Il mittente ripete il messaggio CON finchè non riceve ACK o RST oppure finchè non scade un timeout (solitamente 2 secondi).

CoAP utilizza UDP al layer di trasporto, ma al Layer dei messaggi asincroni è implementato un meccanismo leggero di affidabilità:

- ACK e ritrasmissioni con un approccio Stop-and-wait e back-off esponenziale per il timer di ritrasmissione per messaggi CON
- Rilevazione di duplicati basata su identificatori a 16 bits assegnati a messaggi (tipo numeri di sequenza), chiamati Message IDs, sia per messaggi CON che NON.

## 7.2 Trasmissione dei messaggi

### Trasmissione affidabile

- Iniziata mandando un messaggio CON.
- Il destinatario può: mandare un ACK oppure rifiutare il messaggio, sia esplicitamente mandando un RST oppure implicitamente, ignorandolo.
- I messaggi CON e ACK hanno lo stesso Message ID di 16 bits.

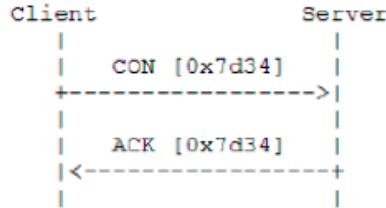


Figure 36: CoAP - Trasmissione affidabile

**Trasmissione inaffidabile** Iniziata mandando un messaggio NON. Il destinatario non deve fare acknowledgement del messaggio.

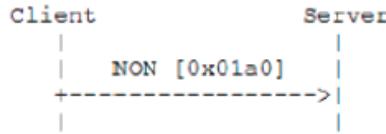


Figure 37: CoAP - Trasmissione inaffidabile

**Modello Request/Response** Modello simile all'HTTP.

Una richiesta include un metodo, l'URI di una risorsa e, opzionalmente, metadati (opzioni CoAP).

Una risposta specifica il codice, payload e altre informazioni opzionali. I *Tokens* fanno match tra richieste e risposte (da non confondere con i Message ID). In HTTP i Token non sono necessari dato che le richieste e risposte sono già "taggati" dalla connessione TCP.

I possibili metodi per una CoAP Request sono:

- GET: richiede una rappresentazione dello stato di una risorsa.
- POST: creazione di una risorsa.
- DELETE: richiede la cancellazione di una risorsa.
- PUT: creazione o aggiornamento di una risorsa.

Le CoAP Response contengono un codice, il payload e, al solito, meta-dati opzionali. La corrispondenza tra Richieste-Risposte è garantita mediante l'utilizzo di un token. Le possibili classi di codici di risposta sono:

- Successo (codice 2.xx)
- Client-error (codice 4.xx)
- Server-error (codice 5.xx): il server è consapevole di aver sbagliato oppure è impossibilitato ad eseguire la richiesta ricevuta.

Se una richiesta è trasportata con un messaggio CON, la risposta potrebbe essere trasportata nel corrispondente ACK:

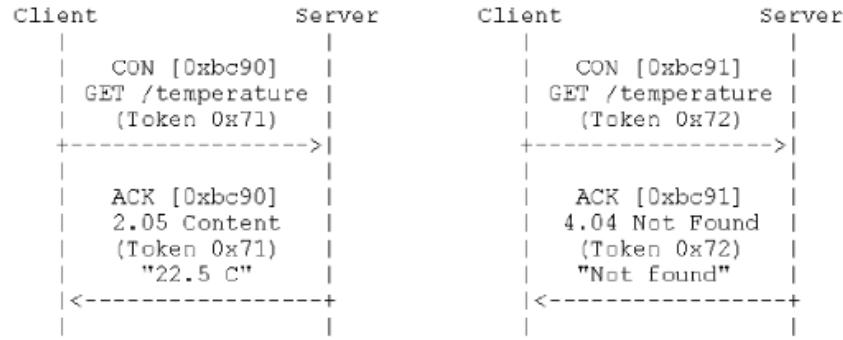


Figure 38: CoAP - Risposta a richiesta tramite ACK

Oppure, potrebbe essere inviata in un differente messaggio CON:

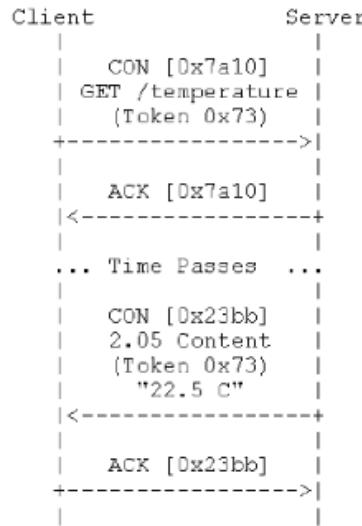


Figure 39: CoAP - Risposta a richiesta in differente CON

Se una richiesta è trasportata in un messaggio NON, anche la risposta sarà trasportata in un messaggio NON:

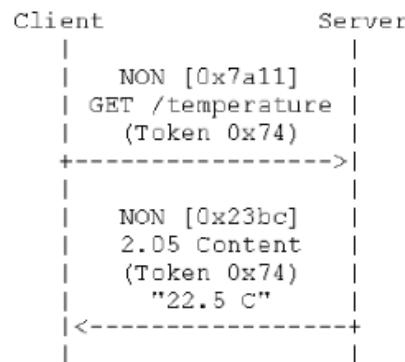


Figure 40: CoAP - Richiesta e Risposta NON

C'è un terzo tipo di messaggio: *Vuoto*. I messaggi vuoti non sono né una Richiesta né

una Risposta, sono identificati da un codice speciale. Un messaggio vuoto in un CON viene usato per "richiedere" un RST: questa feature è chiamata *CoAP ping*.

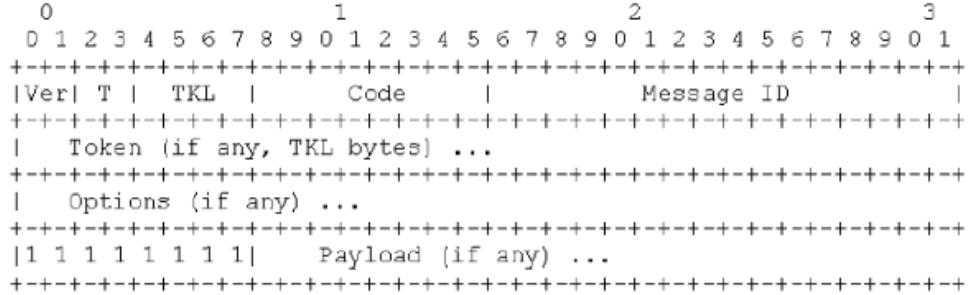


Figure 41: CoAP - Formato Messaggi

**Formato del Messaggio** Header di grandezza fissata di 4 byte.

- **Ver:** versione del protocollo (attualmente = 1).
- **T:** Tipo del messaggio
  - 0: CON
  - 1: NON
  - 2: ACK
  - 3: RST
- **TKL:** Lunghezza Token (i token possono avere grandezza variabile)
- **Code:**
  - *Class* (primi 3 bit): Request = 0, Success = 2, Client Error = 4, Server Error = 5.
  - *Detail* (ultimi 5 bits): specifica il metodo (ad esempio 0.01 è GET) o il codice di risposta (ad esempio, 2.04 è "Created", che notifica il successo di una PUT).
  - Il codice speciale 0.00 denota un messaggio *Empty*.
- **Message ID**

Il campo **Options** codifica informazioni aggiuntive: tra cui, il target URI: "*coap://host[:port]/path?query-options*". L'URI è codificato in 4 campi:

- *Uri-Host*
- *Uri-Port*
- *Uri-Path*
- *Uri-Query-Options*

Utilizzare 4 campi, è più efficiente di utilizzare un solo campo, dato che dovremmo parsare il contenuto per ottenere pezzi separati di informazione. Mentre, in questo caso, abbiamo già campi separati: non abbiamo costi di parsing.

**CoAP Proxy** Altre informazioni codificate nel campo *Options* riguardano il proxying.

- *Proxy-URI*: identifica il proxy server che si vuole utilizzare.
- *Proxy-Scheme*: codifica sia "CoAP" che "HTTP": se il valore si differenzia dal tipo della richiesta corrente, significa che stiamo chiedendo al proxy di fare una traduzione da un protocollo all'altro (cross-protocol proxy).

Potremmo voler implementare un intermediario (proxy) che si pone fra i client ed i server il cui compito principale sarebbe quello di inoltrare richieste/risposte ai destinatari. Un Proxy CoAP potrebbe anche svolgere funzioni di *caching*, traduzione protocolli oppure traduzioni nomi.

Esistono due classi di proxy:

- *Forward-proxy*: effettua richieste al posto del client. Endpoint selezionato dal client ed effettua le necessarie traduzioni.
- *Reverse-proxy*: soddisfa le richieste al posto di uno o più serer. Trasparente al client ed effettua le necessarie traduzioni.

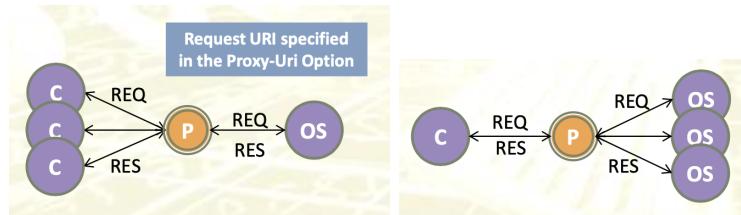


Figure 42: CoAP - Forward e Reverse Proxy

### 7.3 Resource Observing

Client e Server si scambiano la rappresentazione (stati) delle risorse grazie ad un modello publish/subscribe. Un Client interessato a ricevere aggiornamenti su una risorsa si "iscrive" (observe) presso il server che gestisce quella risorsa. L'approccio è di tipo *best-effort*. Definiamo alcuni termini:

- Soggetto: una risorsa su un CoAP server.
- Observer: un CoAP client.
- Registrazione: una richiesta GET *estesa*.
- Notifica: una risposta CoAP aggiuntiva inviata ai client registrati ogni qualvolta lo stato di una risorsa cambia.

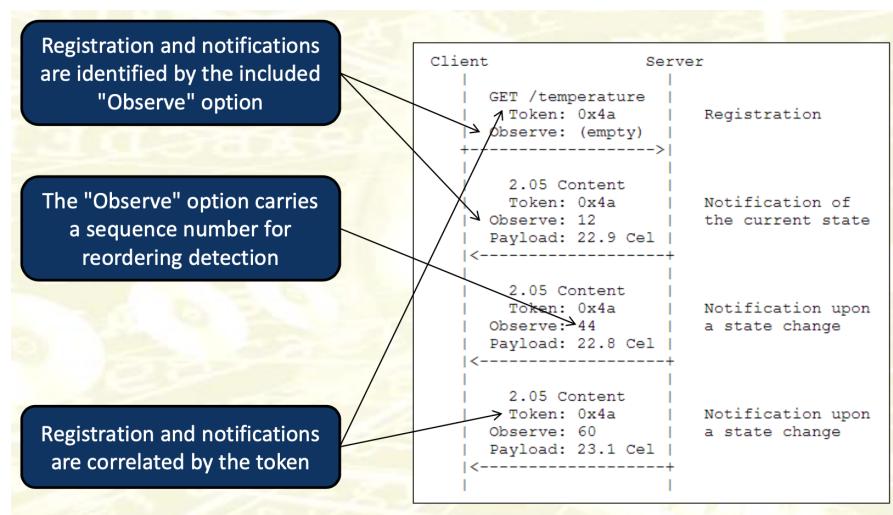


Figure 43: CoAP - Esempio Observe

Una volta che un client si è registrato per l'osservazione di una risorsa, lo standard non specifica come de-registrarsi, ma ci sono differenti possibili strategie:

- Una possibile soluzione sarebbe quella di mandare una nuova richiesta per la stessa risorsa, includendo una nuova opzione che specifichi che il client vorrebbe de-registrarsi.
- Un'altra soluzione, è quella di mandare un messaggio RST che viene interpretato come una richiesta di de-registrazione.
- Le notifiche sono seguite da ACK: basandosi su questa feature, si potrebbe implementare una logica che interpreti la situazione in cui si ricevano ACK multipli come un'implicita de-registrazione (potrebbe anche accadere, per esempio, per un fallimento di un cliente). Questa soluzione potrebbe comunque portare ad errori: un osservatore potrebbe sempre rinnovare la sua registrazione dopo un po' di tempo che non si ricevono notifiche. Con questo intento, è possibile utilizzare un'opzione che consenta all'osservatore di ottenere una notifica ogni X secondi, anche se la risorsa osservata non è cambiata: questo consente al cliente di sapere che non è stato rimosso per errore dal set degli osservatori dal CoAP server.  
Più in generale, è possibile avere opzioni QoS per ricevere notifiche temporizzate.
- È inoltre possibile utilizzare un parametro *Max Age* per settare a priori la durata massima di una registrazione

Nodi intermedi possono fare la observing di risorse al posto dei clients: un client può quindi registrarsi per una observe presso ad un proxy e quest'ultimo, a sua volta, si registra al CoAP Server finale. Il Proxy può gestire client multipli contemporaneamente, ma si registra al nodo CoAP solo una volta, andando a ridurre il carico sul nodo (vincolato).

## 7.4 Resource Discovery

All'inizio, è possibile scoprire server CoAP in un network di sensori wireless mandando un messaggio CoAP speciale: *Multicast CoAP Discovery Request*. Questo messaggio è indirizzato all'indirizzo IPv6 multicast speciale "*All CoAP Nodes*" con porta 5683. Questa operazione di Discovery è limitata allo scope link-local.

A questo punto è possibile scoprire quali risorse sono esposte da parte di un CoAP server. Attraverso il well-known relative URI: *"/.well-known/core"*, che è l'entry point di default per richiedere la lista di risorse hostate da un server.

La lista di risorse è rappresentata con uno speciale linguaggio chiamato *Link Format*, progettato dal gruppo CoRE (Constrained Restful Environment).

Link Format è una collezione di link (definito come relative URI di una risorsa) più un numero di attributi separati da punti.

Esempio di Link Format: */sensor/temp; if="sensor"*. Simile al formato dei valori degli header HTTP. Attributi possibili:

- **Resource Type ('rt')**: Stringa opaca utilizzata per assegnare specifici tipi semantici ad una risorsa.
- **Interface Description ('if')**: Stringa opaca utilizzata per fornire specifiche definizioni di interfacce utilizzate per interagire con la risorsa target.
- **Maximum Size Estimate ('sz')**: un'indicazione della massima grandezza della rappresentazione della risorsa.
- **Content Format ('ct')**: fornisce un indizio riguardo al formato dei contenuti che dovrebbe ritornare la risorsa.
- **Observable ('obs')**: un indizio che indica che la destinazione di un link è utile per l'osservazione.

Per attributi opachi, si intende che lo standard non definisce specifici valori per alcuni attributi e vengono definiti dalla specifica applicazione.

È possibile filtrare i risultati ritornati dopo una richiesta al well-known relative URI. Per esempio, *"GET /.well-known/core? href=/foo"* ritornano tutte le risorse il cui nome comincia

per foo. ”*GET /.well-known/core?rt=temperature*” invece, ritorna tutte le risorse associate al tipo di risorsa ”temperature”.

Le risorse possono essere raggruppate in containers, per esempio:

”*REQ: GET/.well-known/core*”

”*RES: 2.05 Content*

”*</sensors>;ct=40*”

Il container di tutte le risorse si indica attraverso: */.well-known/core*.

L’ultima linea contiene l’attributo di link *Content Type* (’*ct*’) che specifica il formato del contenuto ritornato dalla risorsa (per esempio ”text/plain” o ”application/json”). Il valore 40 significa ”application/link-format”, ovvero che la risorsa ritorna una lista di altre risorse nel Link Format. In altre parole, la risorsa */sensors* è un container di altre risorse: una query ad un container ritorna una descrizione Link Format della risorsa nel container.

Un altro esempio:

”*REQ: GET/sensors*”

”*RES: 2.05 Content*

”*</sensors/temp>;rt=”temperature”;if=”sensor”*,

”*</sensors/light>;rt=”light”;if=”sensor”*”

Questo raggruppamento di risorse in container e sotto-container porta ad un’organizzazione gerarchica delle risorse (in realtà, non sono presenti gerarchie nei CoAP server, le risorse sono piatte, la gerarchia è soltanto come viene vista dal punto di vista della query).

## 8 QoS - Quality of Service

Vogliamo gestire il modo in cui i pacchetti viaggiano nella rete. In particolare, dobbiamo considerare che alcuni pacchetti devono essere trattati in maniera diversa da altri. Questo perché esistono diverse tipologie di applicazioni con requisiti diversi (es. real-time e non).

### 8.1 Algoritmi QoS

Le proprietà desiderabili di un algoritmo QoS sono:

**Isolamento dei flussi** L'algoritmo deve essere in grado di isolare un flusso dagli effetti indesiderati degli altri. Idealmente, l'algoritmo dovrebbe fornire al flusso un link dedicato con capacità uguale al rate minimo garantito per quel flusso. In questo modo, le prestazioni di un flusso non sono influenzate dalla presenza di altri flussi.

**Fairness** La capacità di un link dovrebbe essere divisa fra i flussi richiedenti in maniera equa. A questo scopo si definisce l'Indice di Jain.

Ipotesi: se fair share = equal share e tutti i percorsi hanno la stessa lunghezza, allora l'Indice di Jain  $\in [0, 1]$  è così definito:

$$f(x_1, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

dove  $x_i$  è il throughput del flusso  $i$ .

Se invece come definizione di fairness sceglieremo di allocare la massima capacità disponibile di un link ad un flusso, avremo la **Max-Min fairness**. In questo modello si alloca ad un flusso la massima capacità disponibile di un link ovvero  $\phi_i = \frac{c}{n}$  dove  $n$  è il numero di flussi che attraversano quel percorso.

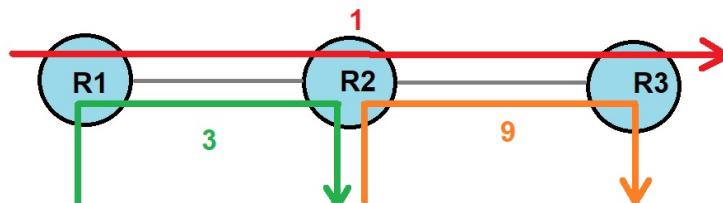


Figure 44: Max-Min fairness

Supponiamo, per esempio, di avere tre percorsi attraverso tre router come mostrato in figura. Avremo un set 0 composto da 1 flusso che attraversa tutti i router, un set 1 composto da 3 flussi che attraversano R1 ed R2 ed infine un set 2 composto da 9 flussi che attraversano R2 ed R3. Quindi tra R1-R2 abbiamo  $3+1=4$  flussi mentre tra R2-R3 abbiamo  $1+9 = 10$  flussi.

Segue che  $\phi_1 = \frac{c}{4}$ ,  $\phi_2 = \frac{c}{10}$  mentre  $\phi_0 = \min(\phi_1, \phi_2)$ . Questo perché il set 0 attraversa tutti i router e quindi la sua porzione di banda allocata sarà la più piccola fra le disponibili. In questo caso si dice che  $\phi_2$  è un *bottleneck* per  $\phi_0$ .

Non sempre una divisione in parti uguali è *fair*. Nell'esempio precedente, il set 0 sfrutta una capacità di  $0.1c$  sul link R1-R2 a fronte di una capacità allocata di  $0.25c$ . In questo modo, il set 1 si trova con una capacità totale di  $0.75c$  (anziché  $0.9c$ ). Per ottimizzare questa allocazione si può usare un algoritmo noto come *Progressive Filling*.

L'algoritmo Progressive Filling prevede che inizialmente tutti i set abbiano  $\phi_i = 0$ . Dopodiché, vengono tutti incrementati della stessa quantità finché non si raggiunge la saturazione di un link. A questo punto, tutte le bande allocate ai flussi che attraversano il link saturo vengono fissate. Si procede dunque all'incremento dei rimanenti set finché tutti i link non sono saturi.

Applicando il Progressive Filling all'esempio precedente si ottiene  $\phi_0, \phi_2 = \frac{c}{10}$  e  $\phi_1 = \frac{1}{3}(c - \frac{c}{10}) = 0.3c$

Si può avere inoltre la **Proportional Fairness** quando la capacità del link viene allocata ad un flusso in base alla richiesta di capacità dello stesso.

**DEFINIZIONE** di Flusso *backlogged*: Data  $A(t)$  funzione di input ad un server e  $W(t)$  funzione di output del server, la funzione  $Q(t) = A(t) - W(t)$  è definita backlog e rappresenta il lavoro incompleto (lunghezza della coda nel buffer). D'ora in avanti i flussi *backlogged* saranno occasionalmente definiti arretrati.

Un altro indice di fairness è l'Indice di Golestani, noto più semplicemente come *Service Fairness Index*, definisce la massima differenza tra il servizio normalizzato ricevuto da due flussi arretrati, su un intervallo in cui entrambi sono continuamente arretrati.

$$RF_{(i,j)}(t_1, t_2) = \left| \frac{W_i^S(t_1, t_2)}{t_i} - \frac{W_j^S(t_1, t_2)}{t_j} \right|$$

dove  $W_i^S(t_1, t_2)$  denota il servizio ricevuto dal flusso  $i$  misurato in unità di traffico (bit), nell'intervallo  $(t_1, t_2)$ , sotto la disciplina di scheduling  $S$ .

**Garanzie sul Ritardo end-to-end** Il tempo di ritardo di un pacchetto su una rete può essere visto come una variabile aleatoria. Possiamo quindi caratterizzare il ritardo in due modi:

- Deterministico: vincolato al ritardo, variazione ritardo.
- Probabilistico: media, percentili ecc.

## Piena Utilizzazione

**Semplicità** Infine, bisogna considerare la semplicità di implementazione. Vogliamo che, alla peggio, l'algoritmo di selezione del pacchetto abbia una complessità computazionale  $\theta(N)$ , dove  $N$  è il numero dei flussi.

## 8.2 Algoritmi di Scheduling

Gli algoritmi di scheduling possono essere:

- **Conservativi:** iniziano sempre una trasmissione quando il link è disponibile e c'è qualcosa da trasmettere nel buffer.
- **Non-Conservativi:** potrebbero non trasmettere anche se il link è disponibile. Perché si fa questo? Perchè potremmo essere in attesa di un pacchetto importante da trasmettere immediatamente e che sappiamo che sta per arrivare.
- **Priorità Ordinata:** ai pacchetti viene applicato un timestamp al loro arrivo per poi essere trasmessi in ordine.
- **Basati sui Frame:** il tempo è diviso in *frame*. I pacchetti vengono trasmessi in frame (es. Round Robin).

Per i nostri scopi considereremo solo gli algoritmi conservativi poiché nell'Internet la cosa più importante è massimizzare l'utilizzo dei link.

### 8.2.1 GPS - Generalized Processor Sharing

E' un algoritmo **conservativo** ispirato alla dinamica dei fluidi in cui i pacchetti sono considerati infinitamente divisibili e il server è in grado di processare più flussi simultaneamente. Questo algoritmo è uno scheduler puramente ideale che viene usato solo come modello di riferimento per quelli reali. Perciò, in seguito, andremo a definire altri algoritmi che cercheranno di avvicinarsi il più possibile a GPS.

In GPS, il link ha una certa capacità  $r$  che viene divisa fra i vari flussi, indichiamo la porzione di ciascun flusso con  $\phi_i$ .

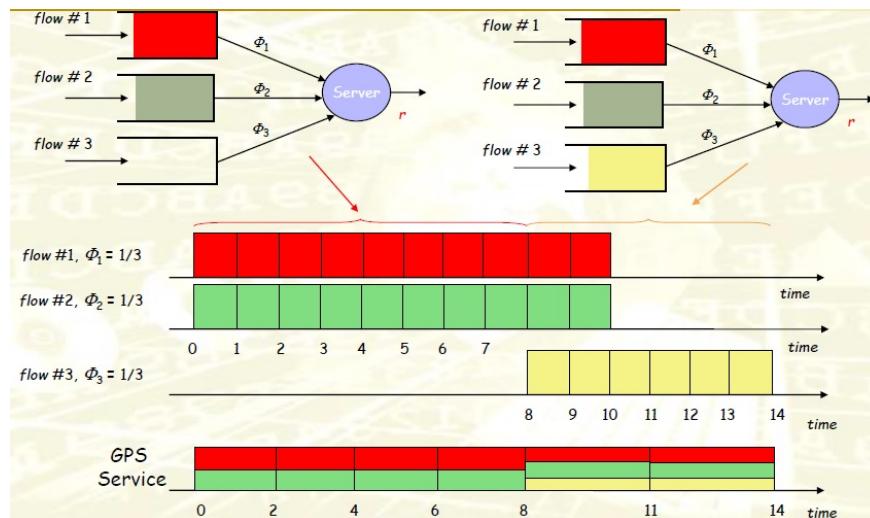


Figure 45: GPS - funzionamento

In figura possiamo vedere il funzionamento di GPS. In particolare, ogni quadretto sulla linea temporale rappresenta un pacchetto. Da  $t=0$  a  $t=7$  abbiamo solo due flussi in arrivo al server, ciascuno occupando 1 unità di tempo per pacchetto. Da  $t=8$  in poi, i flussi diventano tre (quindi 3 unità di tempo).

GPS è un algoritmo conservativo per cui la diseguaglianza:

$$\frac{W_i(t_1, t_2)}{\phi_i} \geq \frac{W_j(t_1, t_2)}{\phi_j}, j = 1, 2, \dots, N$$

è verificata per qualsiasi flusso  $i$  che è continuamente arretrato (backlogged) nell'intervallo  $[t_1, t_2]$ . In altre parole, se la coda del buffer è sempre non-vuota, allora il servizio relativo

(relativo alla mia frazione di banda) che ricevo non può essere minore del servizio normalizzato ricevuto da qualsiasi altro flusso, nello stesso intervallo.

Se entrambi i flussi  $i$  e  $j$  sono entrambi arretrati (backlogged) in  $[t_1, t_2]$  allora si ha:

$$\frac{W_i(t_1, t_2)}{\phi_i} = \frac{W_j(t_1, t_2)}{\phi_j}, j = 1, 2, \dots, N$$

A questo punto se calcoliamo la *fairness* tramite l'indice di Golestani (o Relative Fairness Index), otteniamo che  $RFI = 0$  e quindi GPS è perfettamente fair.

Ad un flusso  $i$  viene garantito un certo **Rate Minimo Garantito**  $g_i$  calcolabile come:

$$\frac{W_i(t_1, t_2)}{W_j(t_1, t_2)} \geq \frac{\phi_i}{\phi_j}$$

$$W_i(t_1, t_2)\phi_j \geq W_j(t_1, t_2)\phi_i$$

$$\sum_{j=1}^N W_i(t_1, t_2)\phi_j \geq \sum_{j=1}^N W_j(t_1, t_2)\phi_i$$

$$W_i(t_1, t_2) \sum_{j=1}^N \phi_j \geq \phi_i \sum_{j=1}^N W_j(t_1, t_2)$$

A questo punto ci interessa notare che il termine  $\sum_{j=1}^N W_j(t_1, t_2)$  indica la somma dei servizi forniti a tutti i flussi nell'intervallo  $[t_1, t_2]$  ma noi sappiamo che c'è almeno un flusso ( $i$ ) che è sempre arretrato (*backlogged*). Quindi durante questo intervallo, sapendo che GPS è conservativo, possiamo dire che il server utilizza tutto il rate  $r$  disponibile.

Detto questo, possiamo scrivere che:

$$\sum_{j=1}^N W_j(t_1, t_2) = r(t_2 - t_1)$$

Sostituendo nella diseguaglianza iniziale:

$$W_i(t_1, t_2) \sum_{j=1}^N \phi_j \geq \phi_i r(t_2 - t_1)$$

e quindi:

$$W_i(t_1, t_2) \geq \frac{\phi_i}{\sum_{j=1}^N \phi_j} r(t_2 - t_1)$$

dove il termine  $g_i = \frac{\phi_i}{\sum_{j=1}^N \phi_j} r$  è il **rate minimo garantito** che cercavamo.

Pertanto si può dire che il servizio  $W_i$  è limitato inferiormente da  $g_i$ .

In figura possiamo osservare come varia il servizio ricevuto dal flusso  $i$ : la curva è limitata superiormente dalla capacità del link  $r$  ed inferiormente dal rate minimo garantito  $g_i$ . La pendenza non può mai essere in contrasto con le pendenze di queste due rette.

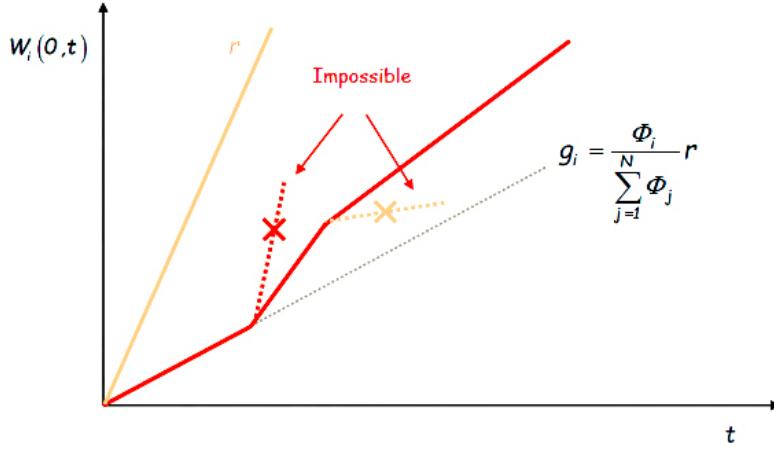


Figure 46: GPS - funzionamento

La banda allocata in eccesso a dei flussi può essere redistribuita tra i flussi arretrati (backlogged) in proporzione al loro peso. In questo caso si parla di rate istantaneo:

$$r_i(t) = \frac{\phi_i}{\sum_{j \in B(t)} \phi_j} r \geq g_i \implies r_i(t) \geq g_i$$

dove  $B(t)$  è il set di flussi *backlogged*.

GPS fornisce anche determinate garanzie sul ritardo. In particolare:

$$\text{delay}_i \leq \frac{Q_i}{g_i}$$

dove  $Q_i$  è il numero di pacchetti *backlogged*.

La garanzia fornita è che il **ritardo nel caso pessimo** è limitato se e solo se il traffico in arrivo è limitato. In conclusione, GPS non è realizzabile per davvero poiché introduce delle assunzioni che non sono realistiche: i pacchetti non sono infinitamente divisibili e non possono essere trasmessi simultaneamente. Ma se implementassimo un algoritmo di scheduling che crea gli stessi output con lo stesso ordine (ma con tempistiche diverse)?

### 8.2.2 WFQ - Weighted Fair Queueing

Algoritmo conservativo che simula il comportamento di GPS mandando i pacchetti in uscita nello stesso ordine, ma con tempistiche diverse. Anche questo algoritmo non è realizzabile praticamente.

Trasmette pacchetti scegliendo, tra quelli in coda all'istante  $t$ , quello che completerebbe il servizio per primo se il sistema fosse GPS e nessun altro pacchetto arrivasse dopo l'istante  $t$ . In altre parole, sappiamo che in GPS i pacchetti possono essere serviti contemporaneamente mentre in WFQ no. Quindi l'algoritmo servirà i pacchetti singolarmente ma nello stesso ordine di GPS. Questo significa che l'output sarà il solito (nello stesso ordine) ma il tempo a cui usciranno i vari pacchetti sarà diverso.

**Esempio** Supponiamo che un pacchetto  $p$  sia in servizio presso un sistema GPS ed uno WFQ. Ad un istante generico  $t$  arriva un nuovo pacchetto  $p'$ . Nel caso di GPS il pacchetto inizia subito il servizio (simultaneamente a  $p$  ed uscirà all'istante  $t + n$ ). Con WFQ il pacchetto inizierà il servizio non appena terminerà quello di  $p$  ed uscirà ad un istante  $t + m$  con  $m > n$ . L'ordine di uscita sarà sempre  $p, p'$  ma il tempo a cui escono i due pacchetti sarà diverso.

Si può dire quindi che WFQ approssima il comportamento di GPS, ma fino a che punto? La differenza tra i due algoritmi è data dalla differenza dei tempi d'uscita per un singolo pacchetto, ed è limitata dalla MTU (Maximum Transfer Unit):

$$\text{out}_{\text{WFQ}} - \text{out}_{\text{GPS}} \leq \text{MTU}$$

Possiamo avere 2 casi: o siamo in ritardo rispetto a GPS, e quindi possiamo esserlo al più del tempo necessario a trasmettere un pacchetto di dimensione = MTU, oppure siamo in anticipo rispetto a GPS (ma questo non è possibile per questioni di fairness). Vediamo un esempio di come WFQ non sia *fair*.



Figure 47:

In figura possiamo osservare l'ordine di arrivo dei pacchetti (sx) e l'ordine di servizio in un sistema GPS (dx). Il servizio in uscita offerto dal sistema WFQ sarà:

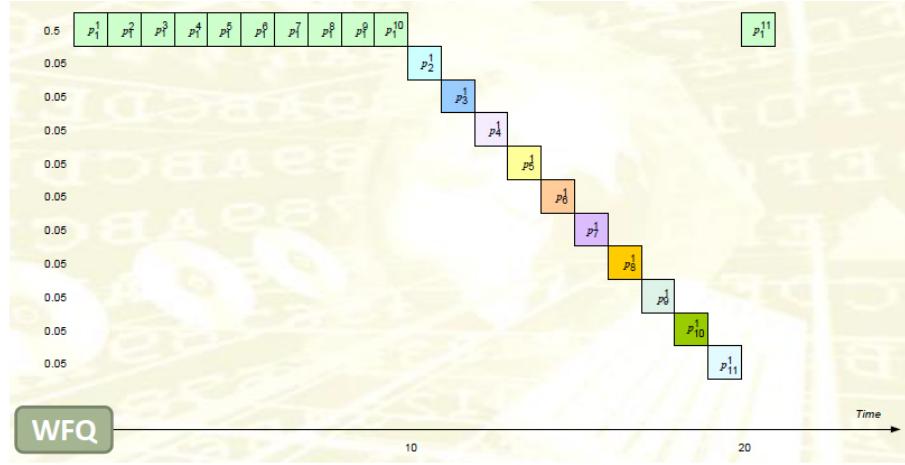


Figure 48: Servizio del sistema WFQ

### 8.2.3 $(WF)^2Q$ - Variante di WFQ

Worst-case Fair WFQ è una variante di WFQ che, quando il server è pronto a trasmettere un nuovo pacchetto ne sceglie uno fra quelli che, al tempo  $t$ , avrebbero già cominciato ad essere serviti da un sistema GPS equivalente. Tornando all'esempio fatto nel capitolo precedente, vediamo come cambia il servizio del sistema WFQ:

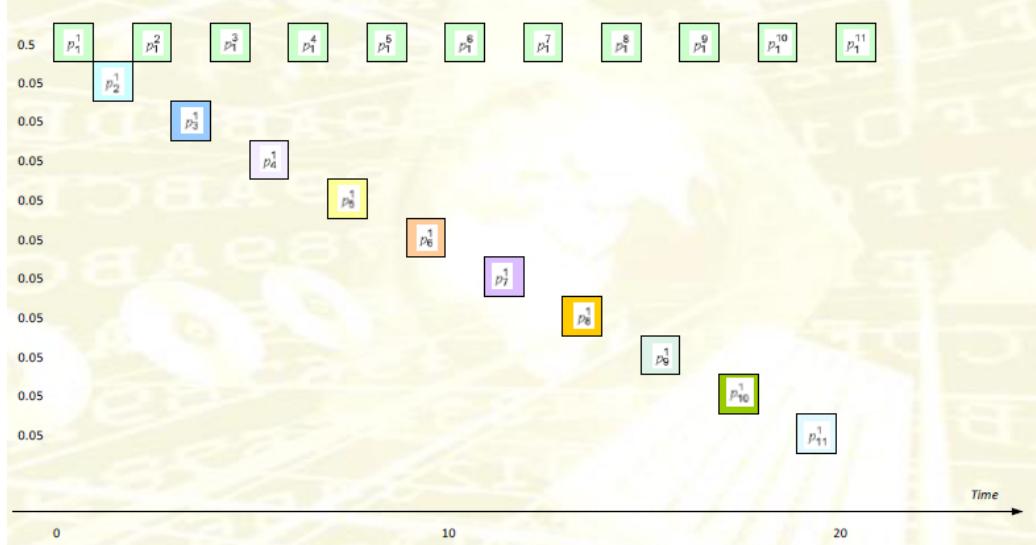


Figure 49: Servizio del sistema  $WF^2Q$

Come si può osservare, il primo pacchetto ad uscire è sempre  $p_1^1$  in quanto è il primo ad arrivare. Tuttavia, quando  $p_1^1$  viene trasmesso il prossimo ad essere preso in lavorazione non sarà  $p_1^2$  in quanto non ancora arrivato, bensì  $p_2^1$ . In questo modo i pacchetti del flusso 1 si alternano con tutti gli altri.

#### 8.2.4 DRR - Deficit Round Robin

Algoritmo di packet-scheduling conservativo, basato su frame (temporali). I flussi sono organizzati in una lista circolare e vengono controllati (in ordine) uno alla volta per vedere se hanno pacchetti da trasmettere. Ogni flusso può trasmettere un numero massimo di pacchetti per ogni round.

Ciascun flusso  $i$  è caratterizzato da:

- un quanto  $Q_i$  di unità di traffico (bit, byte ecc.) che misura la quantità di traffico che il flusso  $i$  dovrebbe *idealmente* trasmettere. Perché idealmente? Perché un flusso potrebbe avere meno unità di traffico da trasmettere al momento in cui viene interrogato, oppure la dimensione del pacchetto potrebbe non coincidere perfettamente con  $Q_i$  (cioè il quanto non è multiplo della dimensione del pacchetto).
- un Deficit Counter ( $DC_i$ ): serve a tener traccia dei byte totali inviati da un flusso, in questo modo si possono recuperare eventuali byte non sfruttati nel round precedente (a causa del fatto che il quanto non era multiplo della dimensione del pacchetto).

Funzionamento:

1. Quando un flusso è *backlogged*,  $DC_i = 0$
2. Ad ogni round,  $DC_i = DC_i + Q_i$
3. Il flusso  $i$  può trasmettere fino a  $DC_i$  unità di traffico.
4. Se il flusso è ancora *backlogged* dopo il servizio,  $DC_i = DC_i - \text{byte(trasmessi)}$
5. Altrimenti, se  $i$  non è più *backlogged*,  $DC_i = 0$

Notare che  $DC_i$  è limitato superiormente dalla dimensione massima di un pacchetto. Se un flusso deve trasmettere un pacchetto con  $\text{dim}(pkt) > Q_i$  allora accumula tutto  $Q_i$  nel deficit. Se al prossimo round  $\text{dim}(pkt) > Q_i$ , accumula di nuovo e così via fino a che, eventualmente,  $DC_i = \text{dim}(pkt)$  e quindi potrà trasmettere.

## 9 DiffServ

Soluzione che fornisce la possibilità ai router di differenziare i flussi in classi in modo tale che, all'interno del dominio DiffServ, ogni pacchetto appartenente alla stessa classe venga trattato allo stesso modo. C'è un numero fisso di classi QoS. La classificazione del traffico viene fatta al **confine** del dominio DiffServ, ossia all'interfaccia d'ingresso di un "border router".

Non si potrebbe delegare questo compito all'applicazione? In teoria sì, ma non vogliamo che questo accada perché significherebbe standardizzare le classi QoS in tutta la rete internet.

### 9.1 Traffic Classification

Processo di identificazione dei pacchetti e di assegnamento di una classe, con lo scopo di applicare un'azione comune ad essi. Esistono diversi metodi per classificare i pacchetti:

- Complesso: detta anche multi-campo, si basa sul valore di campi specifici all'interno dell'header IP (per esempio sorgente/destinazione o porta UDP o TCP).
- DPI (Deep Packet Inspection): controlla anche i dati del pacchetto.
- Semplice: riguarda campi dell'header progettati appositamente per la classificazione QoS.

### 9.2 Traffic Conditioning

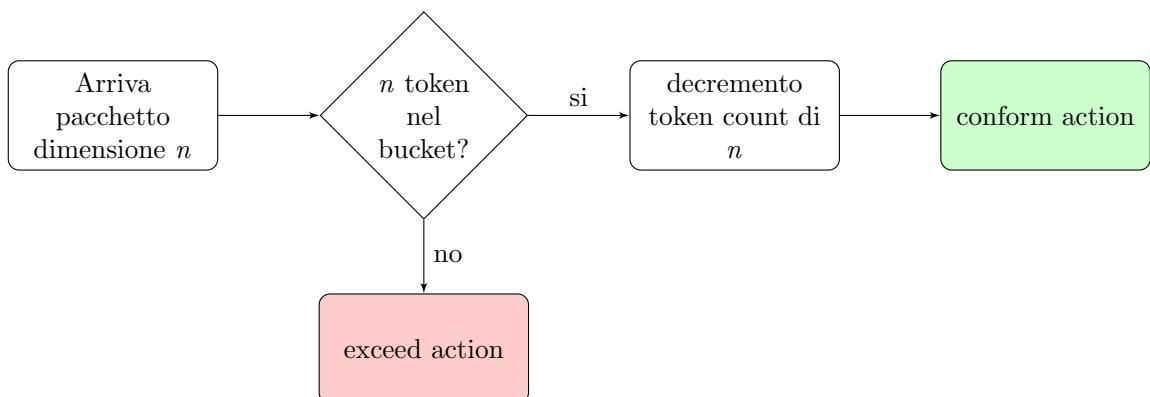
Meccanismo tramite il quale possiamo:

- Policing: Imporre regole al flusso di traffico per far rispettare un determinato profilo di burst/rate. Solitamente i pacchetti non conformi sono scartati.
- Shaping: Riformare (ritardare pacchetti) in un flusso di traffico per far rispettare quanto sopra. I pacchetti non conformi vengono modificati in modo da esserlo.

Concretamente questo significa che nel router andremo a definire i vari gruppi (classi) e come classificare i flussi di traffico nei gruppi. Dopodiché, per ogni gruppo, stabiliamo un "profilo" che ogni flusso appartenente a quel gruppo deve rispettare per essere ammesso nel dominio DiffServ.

#### 9.2.1 Policing

Un metodo per implementare il policing è il *token bucket*, ossia un contenitore dentro al quale disporremo dei "gettoni" virtuali. Inizialmente il bucket è pieno. Un pacchetto, per poter trasmettere, ha bisogno di un numero di token pari alla sua dimensione.



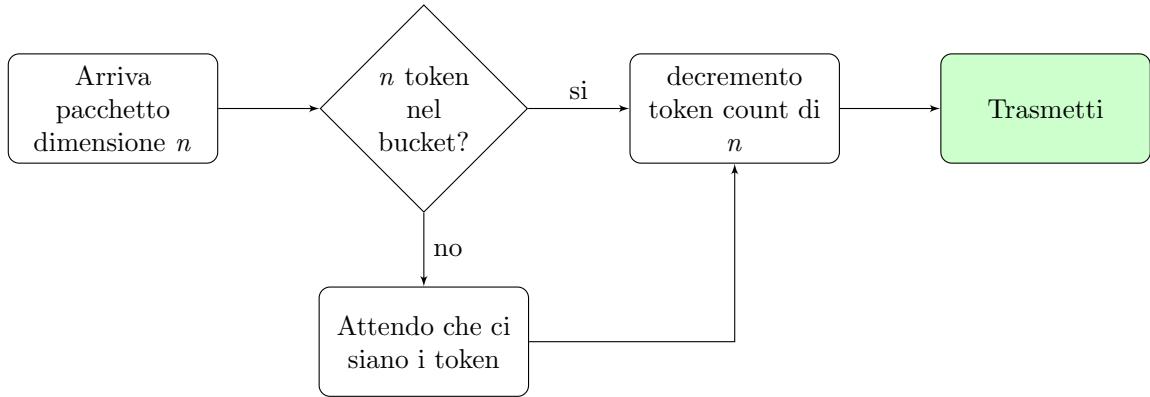
A questo punto bisognerà definire in cosa consistono le azioni di *exceed* e *conform* in base al comportamento che vogliamo ottenere.

Con il policing si va ad eliminare il ritardo a scapito di una maggiore perdita pacchetti. Questo perché丢弃ing i pacchetti all'ingresso della coda impongo un limite alla funzione degli arrivi e quindi la rete non sarà mai congestionata.

**Multiple Rates** Un caso del token bucket è quello in cui abbiamo 2+ bucket concorrenti. Un pacchetto che vuole viaggiare nella rete dovrà ottenere un numero di token sufficienti sia nel primo bucket che nel secondo.

### 9.2.2 Shaping

In questo caso, se non ci sono abbastanza token nel bucket il pacchetto non viene scartato, bensì viene messo in attesa fino a che non ci saranno token a sufficienza.



## 9.3 DiffServ Code Point

Nuovo campo da 6bit introdotto nei pacchetti IP per permettere la classificazione dei pacchetti in classi corrispondenti ad un PHB (Per-Hop Behaviour). Questi 6 bit possono essere usati liberamente all'interno di un dominio DiffServ purché pacchetti appartenenti allo stesso flusso abbiano la stessa classe PHB. Di default, i pacchetti hanno la classe **CS0** detta anche best-effort.

Codepoint	DSCP		
Default/CS0	000000		
EF PHB	101110		
CS1	001000		
CS2	010000		
CS3	011000		
CS4	100000		
CS5	101000		
CS6	110000		
CS7	111000		
AF PHB Group	Drop Precedence		
AF Class	Low	Medium	High
AF1x	001010	001100	001110
AF2x	010010	010100	010110
AF3x	011010	011100	011110
AF4x	100010	100100	100110

Figure 50: Codici DSCP

**Expedited Forwarding (EF)** Se un pacchetto è marcato con questa classe EF, allora a quel pacchetto è garantito un rate minimo indipendente dal carico del traffico non-EF. Questa classe è particolarmente utile per applicazioni che necessitano basso ritardo e poca perdita di pacchetti (esempio: VoIP).

**Assured Forwarding (AFxy)** Set di classi pensate per un tipo di traffico che non ha particolari requisiti in termini di latenza. Per questo, il traffico out-of-profile verrà lasciato passare ugualmente (a meno che non crei congestione nella rete, in quel caso verrà rimosso). Abbiamo 12 possibili gruppi per AF: i primi 3 bit x identificano la classe (1-4) mentre i successivi 2 bit y servono a stabilire la precedenza di drop all'interno della stessa classe

(low, med, high).

## 9.4 Active Queue Management

In che modo vengono scartati i pacchetti? Tradizionalmente ciò avviene quando un pacchetto arriva all'ingresso della coda ma non c'è abbastanza spazio per contenerlo. Questa strategia è chiamata **tail drop**. Il motivo per cui si scartano i pacchetti non è perché non c'è abbastanza memoria per gestire le code ma perché code più lunghe implicano ritardi maggiori dovuti al tempo di accodamento.

Per fornire un trattamento diverso alle classi di pacchetti, potremmo usare una tecnica nota come **tail-drop pesato**. Supponiamo che la coda abbia due limiti  $q_1$  e  $q_2$  con  $q_1 < q_2$ . Quando un pacchetto a bassa priorità arriva, viene scartato se la dimensione della coda è maggiore di  $q_1$ . Quando arriva un pacchetto ad alta priorità, si applica invece la soglia  $q_2$ . Quando avviene il drop del pacchetto, il mittente (tipicamente) reagisce diminuendo il suo rate di trasmissione che, teoricamente, porterebbe ad un'ottimizzazione del throughput dell'applicazione. Questo in realtà non è sempre vero poiché se tutti i mittenti spediscono pacchetti simultaneamente, questi vengono scartati. A questo punto i mittenti reagiscono diminuendo il rate di trasmissione e si arriva ad uno scenario in cui si è passati ad una bassa utilizzazione della rete. Questo problema è noto come *global synchronization*.

Per questi motivi viene sviluppato l'Active Queue Management, un insieme di tecniche volte a gestire le code di un router in modo da limitare il ritardo e ottimizzare il throughput.

### 9.4.1 RED - Random Early Detection

Tecnica pro-attiva pensata per rilevare preventivamente una possibile congestione e per fornire feedback alle applicazioni. La congestione è rilevata tramite l'osservazione della lunghezza (profondità) media della coda. Raggiunta una certa soglia di riempimento, se arriva un nuovo pacchetto, viene deciso in maniera casuale se scartarlo o meno. In questo modo solo uno (o pochi più) dei flussi diminuirà (diminuiranno) il rate. Abbiamo quindi evitato il problema della *global synchronization*.

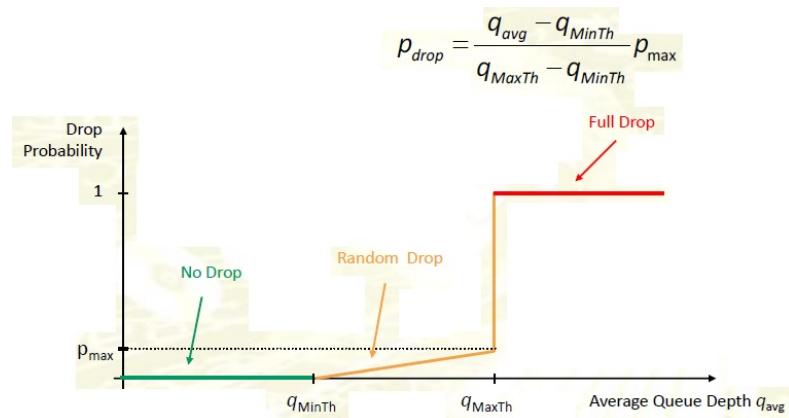


Figure 51: Random Drop

Come possiamo vedere in figura, ad ogni soglia  $q$  della coda corrisponderà una certa probabilità di drop. Possiamo rendere l'algoritmo più o meno *aggressivo* andando a modificare la pendenza della retta di random drop. Questo può essere utile nel caso delle classi DSCP perché ci permette di avere un comportamento di RED diverso per ogni classe, a seconda della loro precedenza di drop.

## 10 Core Network Protocols and Architectures

### 10.1 MPLS - Problema

Con l'espansione della rete, sono nati problemi di scalabilità di gestione delle reti. Ogni router contiene tabelle di routing/forwarding per l'inoltro dei pacchetti, generate dagli algoritmi. Per inoltrare un pacchetto a destinazione, dovrà essere presente un'entrata nella tabella di routing per la destinazione.

Il dominio della rete è composto da border router e core. Questo dominio deve consegnare i pacchetti a diverse destinazioni, l'inoltro avviene ad ogni hop. Se un pacchetto arriva ad un certo border router e deve uscire da un altro border router, dovrà seguire un determinato percorso nel dominio. L'inoltro del pacchetto è basato sull'indirizzo destinazione contenuto nel pacchetto. Ad ogni router viene scelto il next hop in base alla tabella di routing, che è conosciuta solo al router che la possiede e che mappa DEST, ADD e NEXT HOP. Quando un pacchetto arriva ad un router è necessario che vi sia corrispondenza, cioè l'indirizzo di destinazione indicato nel pacchetto deve essere presente nella tabella. Più si è interni al core della rete, più la tabella di routing sarà grande.

Quindi, più sono i nodi più saranno le entrate necessarie nella tabella di routing. Si crea quindi un problema di scalabilità. Quando un router interno deve mandare un pacchetto all'esterno, seleziona il miglior border router tra quelli disponibili e gli inoltra il pacchetto. Nella tabella di routing ci saranno le entrate per i nodi del dominio e per le network esterne che si annunciano al BR.

Il problema principale, quindi, risiede nel fatto che per ogni "nuova" rete esterna che si annuncia, si deve aggiornare la tabella di routing di ogni router all'interno al dominio. Questo approccio, come si può immaginare, non scala bene in quanto si ha un aumento della dimensione della tabella di routing senza che nel dominio siano aggiunti nuovi nodi.

Si avrà quindi bisogno di uno strumento nei core router che ci dica, per ogni destinazione esterna, quale sia il BR a cui inoltrare il pacchetto. In questo modo solo i BR aggiorneranno la loro tabella con gli annunci esterni al dominio. Non c'è alcun modo di fare questo con IP.

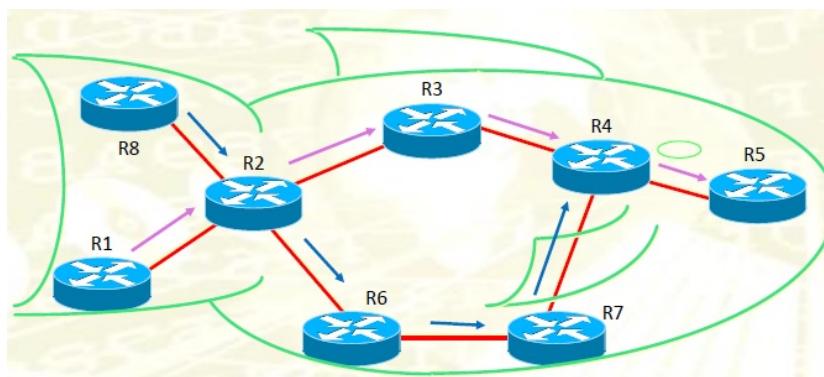


Figure 52: Esempio di *fish-network*

Un esempio lampante per capire il problema è quello della "fish network" mostrato in figura, in cui si hanno 2 flussi distinti che devono raggiungere un certo border router. Assumendo che uno dei percorsi per raggiungere il border router sia più lento e l'altro più veloce, vorremmo decidere di differenziare i 2 flussi nei due percorsi differenti. Questo con IP non è possibile, perché il path in IP viene deciso solo tramite il destination address. Tutti i flussi di dati vengono trattati ugualmente, non è possibile modificare questo comportamento del protocollo IP. Quindi, l'obiettivo è che un percorso non venga scelto solo attraverso la destinazione, ma anche sulla base della sorgente.

Per risolvere il problema, si deve fare *reverse-engineering* su OSPF, modificando il costo del link, in modo tale che il risultato dato in output da OSPF sia uguale a quello che abbiamo calcolato esternamente. Tramite questo lavoro si ottimizzano le performance della rete.

MPLS (Multi-Protocol Label Switching) si pone come soluzione per risolvere problemi di questo tipo.

## 10.2 MPLS - Funzionamento

### 10.2.1 Concetto Label Switching

Si va a modellare il processo di forwarding all'interno dei router. Può essere diviso in 2 fasi:

- **Partizionamento:** I pacchetti vengono divisi in classi dette FEC (Forwarding Equivalent Class). La FEC di un pacchetto nel classico forwarding IP viene scelta guardando solamente il Destination Address (che è quello che vogliamo cambiare con MPLS). Pacchetti che appartengono alla stessa FEC vengono trattati allo stesso modo dal router.
- **Mappatura:** Ogni FEC è mappata dal router verso un determinato next hop creando così una corrispondenza ben precisa.

In tutto questo assumiamo che tutti i router del dominio abbiano la stessa visione su come partizionare i pacchetti in FEC, ovvero siano d'accordo su quali siano le classi FEC.

Il partizionamento e la mappatura, nel forwarding IP convenzionale, vengono effettuati ad ogni hop. Nel nostro approccio si vanno a dividere, il partizionamento viene effettuato negli edge router (router di ingresso) mentre la mappatura viene fatta ad ogni hop.

Questo perché, la fase di partitioning è molto complessa, dato che potenzialmente coinvolge l'ispezione di molti campi di un pacchetto. Come conseguenza di questa suddivisione di ruoli, si deve assegnare un label ai pacchetti in modo che i core router possano identificare a quale FEC appartengono i pacchetti. Un valore della label L corrisponde ad una specifica classe e viene inserito all'interno del pacchetto.

Questo approccio cambia la struttura delle tabelle di forwarding, che adesso hanno un mapping tra il label e il next hop. È molto conveniente, dato che consente di ripetere il partitioning ad ogni router che, come detto precedentemente, è ritenuta un'operazione molto dispendiosa. Il label è un campo "piccolo" e quindi di facile ispezione, rispetto ad eventuali altri campi (ad esempio il destination address).

Questa soluzione, inoltre, migliora il problema della scalabilità: quando diventa raggiungibile un nuovo network esterno, abbiamo bisogno di aggiornare soltanto i router di ingresso, in modo tale che pacchetti indirizzati alla "nuova" destinazione, siano mappati con una FEC associata al cammino che porta all'appropriato router di uscita. Mentre, i router interni non hanno bisogno di nessun update, perché sono già in grado di mappare quella FEC al prossimo hop, ovvero esiste già un cammino associato a quella specifica FEC.

Per implementare questa funzionalità descritta, abbiamo bisogno di un modo per consentire ai router di accordarsi a quale label associare a ogni FEC. Questo task potrebbe risultare abbastanza difficile, infatti si è trovato un modo per "aggirare" questo problema: si consente ad ogni router di scegliersi i label per rappresentare i FEC. I router si scambiano le informazioni con i propri vicini riguardo la rappresentazione usata, cosicché quando uno di questi deve trasmettere un pacchetto ad un vicino, semplicemente cambia il label associato al pacchetto giusto prima di effettuare la trasmissione, in modo da conformarsi con la rappresentazione utilizzata dal vicino.

### 10.2.2 Architettura MPLS

**Piano dati MPLS** La figura 53 rappresenta un esempio pratico di come viene gestito MPLS dai vari router, sul piano dati. Nell'immagine, i nodi ai bordi vengono chiamati "edge" e quelli interni "core".

I router coinvolti in questa operazione vengono definiti con il termine *Label Switching Routers (LSR)*.

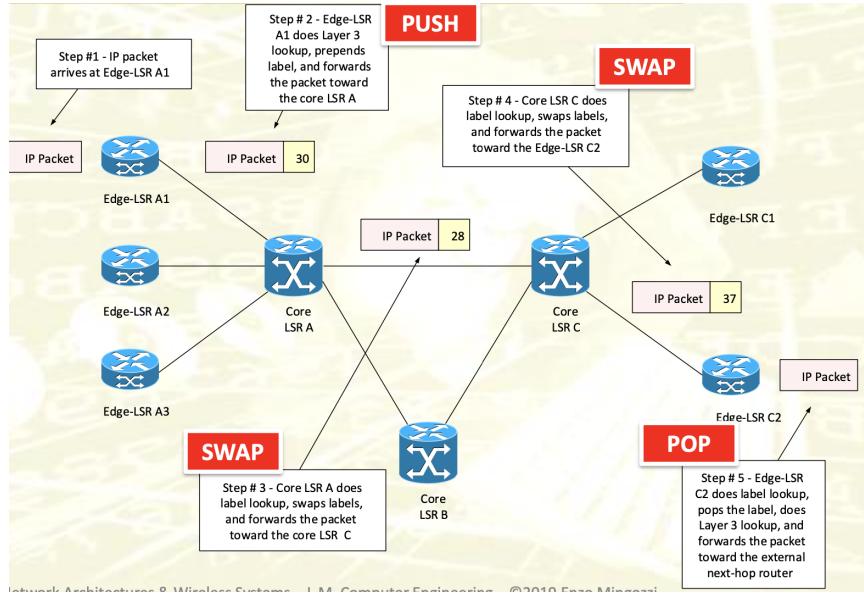


Figure 53: MPLS data plane

Supponendo che arrivi un pacchetto al router A1. Questo è inizialmente privo di etichetta; deve quindi essere specificata la classe di inoltro del pacchetto, quindi al successivo nodo viene mandato il pacchetto IP ed in più la nuova informazione riguardante la classe. Questa operazione di etichettamento viene chiamata *PUSH* (il cui nome viene preso dalle tipiche operazioni effettuate su uno stack). Quando il pacchetto arriva al Core LSR A, questo riceve un pacchetto che già possiede un'etichetta, quindi lo inoltrerà, a seconda dell'etichetta, al prossimo hop. Prima di inoltrare un pacchetto, però, viene sostituito il valore dell'etichetta del pacchetto ricevuto con un valore utile a permettere al prossimo nodo di identificare la classe del pacchetto: operazione di *SWAP*. Stessa cosa per il Core LSR C, e così via fino a giungere al router finale: se il pacchetto deve uscire dal dominio, l'ultimo router farà un'operazione di *POP*, ovvero rimuove l'etichetta al pacchetto, facendolo tornare come era prima di entrare nel dominio.

**Codifica del label** Il label viene codificato in una stringa di bit lunga almeno 32 bit. Se sono presenti più etichette, ci saranno altrettanti blocchi da 32 bit, posizionati in una sorta di stack (da qua, i nomi relativi alle operazioni menzionate nel paragrafo precedente).

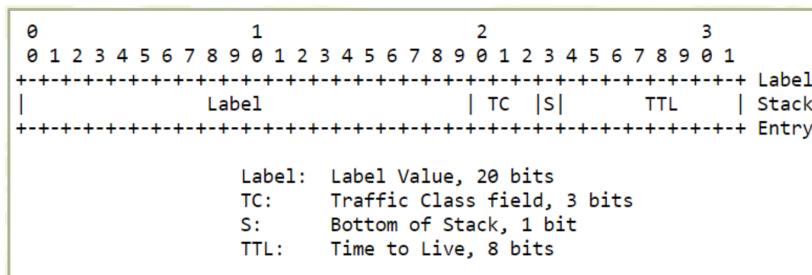


Figure 54: Formato dell'header

Nello specifico, i campi vengono definiti nel seguente modo:

- **Label:** è un identificatore corto, di lunghezza fissata e significativo solo localmente. Viene utilizzato per identificare un FEC.
- **Traffic Class (TC):** identifica la classe di traffico di appartenenza del pacchetto, per introdurre una differenziazione di trattamento dei pacchetti con uguale etichetta.
- **Time-to-Live (TTL):** nello stile di IPv4 o IPv6. Serve a garantire che in percorsi ciclici il pacchetto non giri all'infinito.

- **S:** Flag che serve a gestire la pila. Nello specifico avremo **s=0** se oltre alla prima label ne esistono altre successive oppure **s=1** se è stato raggiunto il fondo dello stack.

Per ogni protocollo di livello 2, c'è un modo per il ricevitore di sapere che i successivi 4 byte sono MPLS e non occorre leggere altro oltre questi 4 byte per sapere come effettuare il forwarding. Si parla di layer 2.5, perché l'header si pone, appunto, in mezzo ai layer 2 e 3. Quindi, per ogni tecnologia di livello 2, bisogna specificare un valore per il tipo di payload che identifichi che il prossimo layer è MPLS.

Questo, però, non succede nel layer di MPLS, cioè non esiste un campo che identifichi quale protocollo sia il successivo, l'informazione viene considerata implicita. Ad esempio, si suppone che tutti i router usino IP. In alternativa si possono utilizzare classi diverse per specificare differenti tipi di payload.

**Modello funzionale MPLS** Composto da 3 strutture dati:

- **Next Hop Label Forwarding Entry (NHLFE):** che contiene le informazioni che riguardano l'azione da eseguire sul pacchetto (push, pop, swap), i parametri relativi a quella azione (se faccio un'azione devo conoscere il valore da inserire) e il next hop.
- **Incoming Label Map:** viene usata quando abbiamo un pacchetto in arrivo con un'etichetta già applicata. In questa mappa ci sono dei puntatori alla NHLFE e a seconda della label nell'etichetta del pacchetto vado a fare una delle azioni della NHLFE.
- **FEC-to-NHLFE Map:** fa un mapping tra FEC e label. A seconda della classe scelta, andiamo a fare un push dell'etichetta con il corrispondente valore del label.

Se il next hop è il router *LSRx*, si fa una POP, ovvero si toglie l'etichetta. Il pacchetto ritorna allo stesso router (*LSRx* infatti indica lo stesso router che sta processando il pacchetto al momento), che, se non ci saranno più etichette, lo inoltrerà al Next Hop come un normale pacchetto IP.

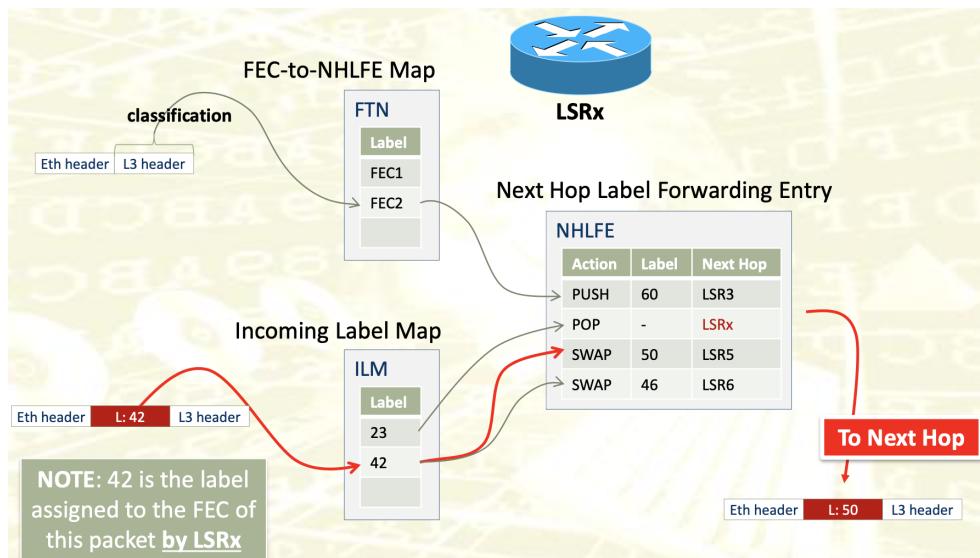


Figure 55: Strutture dati di MPLS

Quello presentato è uno schema generale, le implementazioni reali potrebbero utilizzare strutture dati organizzate in modi differenti o con nomi diversi. In generale, si identificano con il nome generico di *Label Forwarding Information Base*.

**Label Switched Path (LSP)** Un LSP è un percorso unidirezionale composto da LSR sul quale viaggiano i pacchetti appartenenti ad una certa FEC per raggiungere la loro destinazione. Identifica l'insieme delle informazioni correlate distribuite nelle varie Label

Forwarding Information Base e che definiscono il percorso da ingresso a uscita. LSP non introduce meccanismi legati alla connessione e non è un tunnel.

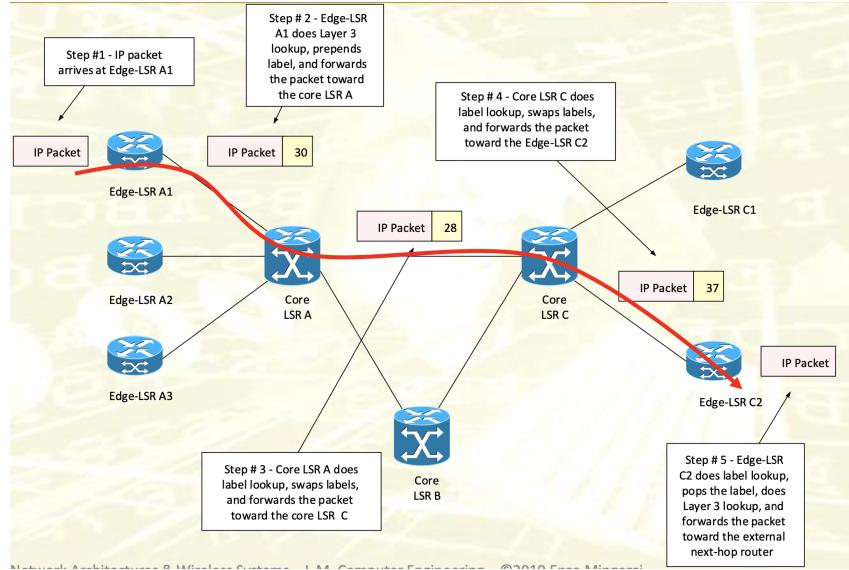


Figure 56: Label Switched Path

**Penultimate Hop Popping (PHP)** Una piccola ottimizzazione può essere effettuata tramite PHP. Considerando cosa accade quando un pacchetto raggiunge l'ultimo hop di LSP: l'ultimo router fa un look up nella sua tabella interna e scopre che deve fare un POP del label: *LFIB: rimuovi il label*. Dopo questa operazione, il router utilizza la routing table IP per effettuare il routing come farebbe normalmente: *FIB: forward del pacchetto IP sulla base dell'indirizzo del prossimo hop*. Quindi, normalmente, si fanno 2 look-ups su tabelle diverse. L'ottimizzazione proposta da questo approccio, consente di fare una sola look-up. Come suggerisce il nome, quindi, si effettua l'operazione di POP non all'ultimo nodo, ma al penultimo. Quindi, il penultimo router effettua la POP ed invia il pacchetto al next hop senza l'etichetta.

**Label Stacking** I concetti espressi precedentemente, sono validi quando è presente un solo label nel pacchetto. Come già detto, è consentito avere più label e questi saranno organizzati a stack (pila). Quindi un pacchetto etichettato, porterà un numero di etichette  $m \geq 0$ .

In questo stack, i pacchetti saranno organizzati in "last-in, first-out", il label 1 è considerato l'ultimo e l' $m$ -esimo il top. Quindi, le azioni nella NHLFE si applicano sempre al top del label nello stack.



Figure 57: Organizzazione degli header con stack

Operazioni possibili su uno stack:

- **SWAP Lx:** sostituisce l'etichetta al top dello stack con uno specificato label Lx. Non è possibile fare uno swapping di un'etichetta non in cima alla pila.
- **POP:** effettua l'operazione di POP sullo stack. Come nel caso precedente, non è possibile effettuare questa operazione in un'etichetta non in cima alla pila.
- **PUSH Lx:** effettua l'operazione di push con il nuovo label Lx nello stack. Sono ammesse push multiple. Questa operazione è un po' più elaborata, dato che è un'azione combinata di SWAP e PUSH. Quando arriva un pacchetto nel dominio e devo fare una push dell'etichetta m, faccio sempre una swap all'etichetta  $m-1$ , e poi faccio una push

dell'etichetta m. Nel caso il pacchetto non abbia altre etichette, viene fatta solo la push senza la swap.

L'utilizzo di etichette multiple organizzate in stack, può essere utile per effettuare il tunneling delle LSP.



Figure 58: Organizzazione degli header con stack

L'esempio seguente mostra l'utilizzo delle etichette per creare una sorta di gerarchia di LSP e come vengono gestite le 3 operazioni (swap, pop, push).

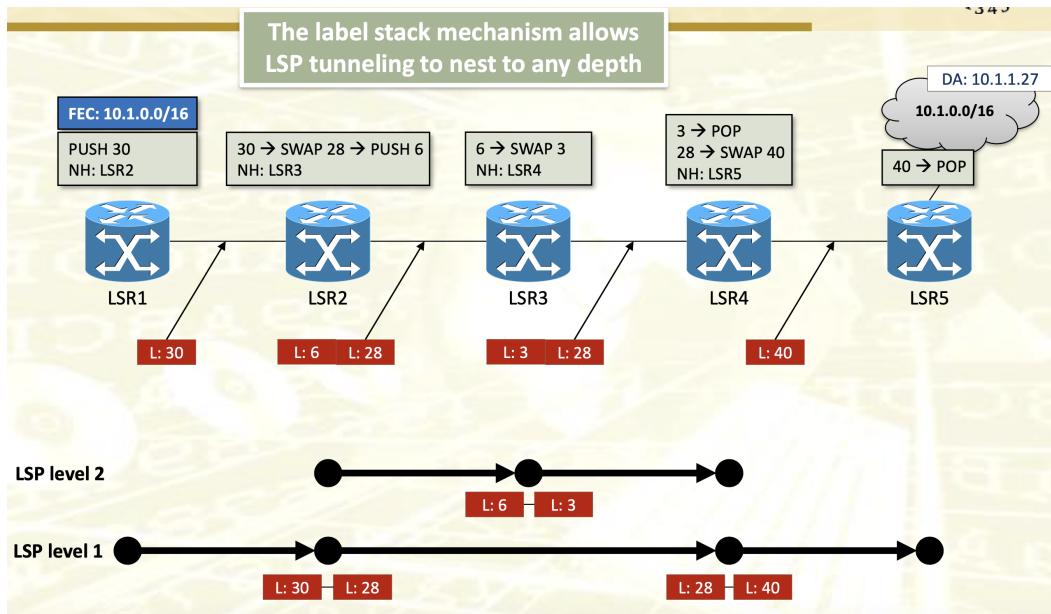


Figure 59: Esempio di LSP tunneling

L'etichetta prima di entrare nel dominio era 30, quando arriva al dominio diventa 28 (viene fatto lo swap) e poi successivamente viene fatta la push dell'etichetta relativa al dominio in cui siamo appena entrati (con label 6). Nel percorso all'interno del domain, l'etichetta con label 28 rimane invariata, viene usato il meccanismo del PHP. Quando si arriva all'ultimo router del dominio, viene fatta la POP della label del dominio, e poi la SWAP della label esterna da 28 a 40. Prima di fare la push, ho bisogno di fare la swap, dato che la label 30 ha valore solo per il router R2, mentre per il router R3 quella classe non corrisponde al label 30, ma al 28. Quindi è necessario che R2 faccia la swap.

**Piano controllo MPLS** Con il Control Plane un LSR ha a disposizione delle procedure per:

- creare bindings tra labels e FEC
- informare gli altri LSR dei bindings effettuati
- costruire e mantenere le strutture dati del data plane

**LSR label scope** L'associazione tra label e FEC è un'associazione locale: due LSR vicini utilizzeranno il medesimo label per lo scambio dei pacchetti appartenenti alla stessa FEC, dopodiché ognuno dei due utilizzerà un diverso label per i pacchetti appartenenti alla stessa FEC, ma destinati ad altri LSR.

Non è detto che una label corrisponda ad una sola FEC:

- **per-interface label scope:** dato un LSR  $R_D$  questo può: associare  $L_1$  ad una FEC F e distribuirla ad un LSR  $R_{U1}$  oppure associare  $L_2$  ad una FEC F e distribuirla ad un LSR  $R_{U2}$ . Se e solo se  $R_D$  può determinare la provenienza da  $R_{U1}$  o  $R_{U2}$  di un pacchetto con label L, dunque solitamente se valgono entrambe le seguenti:  $R_{U1}$  e  $R_{U2}$  sono connessi mediante una connessione punto punto a  $R_D$  e  $R_{U1}$  e  $R_{U2}$  sono gli unici peers con cui  $R_D$  scambia informazioni di binding che riguardano L. Quindi, quando si riceve un pacchetto con un dato label, il mapping ad un'azione dipende sia dallo stesso label ma anche dall'interfaccia da cui stiamo ricevendo il pacchetto. Ad esempio, ricevendo lo stesso label da due router diversi, il router tratterà in maniera differente i due pacchetti perché sono stati ricevuti da due interfacce diverse, dato che sono mappate in due FEC differenti.
- **per-LSR label scope:** dato un LSR  $R_D$  questo può: associare  $L_1$  ad una FEC F e distribuirla ad un LSR  $R_{U1}$  oppure associare  $L_2$  ad una FEC F e distribuirla ad un LSR  $R_{U2}$ . Quindi, ogni label all'interno del router, indipendentemente dal sender del pacchetto, è una differente chiave per mappare FEC differenti. In questo approccio abbiamo bisogno di più labels per fare il mapping.

**Assegnamento ed Allocazione dei Labels** L'assegnamento dei labels è downstream based: sono quindi i downstream LSR a comunicare agli upstream LSR quale label assegnare ad una data FEC. Questo è fatto per evitare la non univocità con peers che fanno uso di un per-interface label space. Quel che può variare è l'allocazione dei label, ossia quando il downstream decide l'associazione label-FEC.

**Downstream On Demand** Con questa modalità un downstream LSR alloca un label ad una FEC solo su richiesta da parte di un upstream LSR. Tale associazione ha validità solo per una specifica interfaccia.

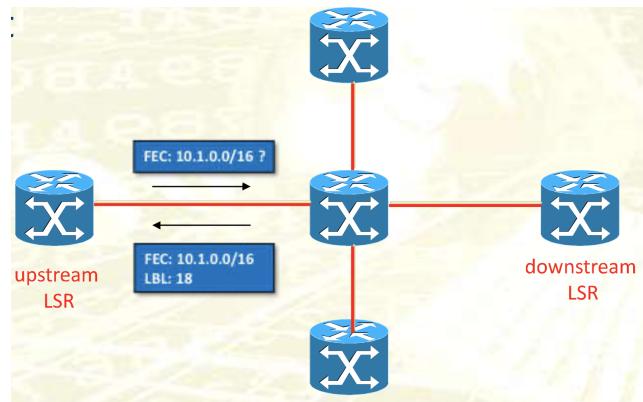


Figure 60: Downstream on demand

**Unsolicited Downstream** Con questa modalità un downstream LSR alloca spontaneamente un label ad una FEC ed invia tale associazione a tutti gli upstream LSR adiacenti, non c'è bisogno di alcuna richiesta esplicita.

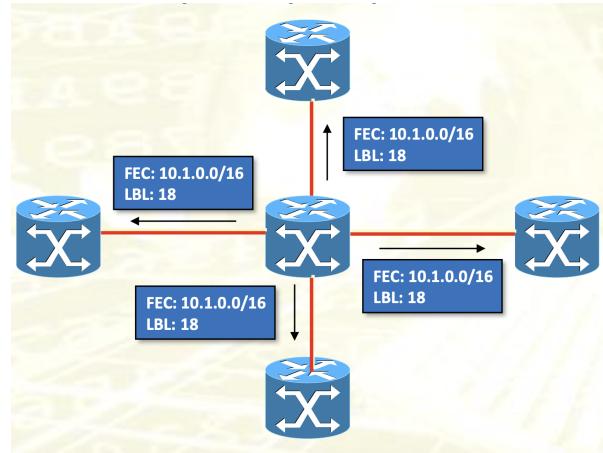


Figure 61: Unsolicited Downstream

Non è detto, però, che tutti i vicini siano interessati a tale binding, legato all'utilizzo di quel router come downstream LSR per una comunicazione verso la FEC di cui fa advertisement. Si configurano quindi 2 possibilità:

- tenere comunque traccia dell'associazione per usi futuri, nel qual caso si parla di *liberal label retention mode*. Tale approccio è consigliato se si vuole avere una reattività ai cambiamenti nel routing, ad esempio per avere sempre a disposizione una backup route nel caso del fallimento di un link.
- scartare il binding non tenendone di conto, nel qual caso si parla di *conservative label retention mode*. Tale approccio è consigliato se il problema del salvataggio delle informazioni è piuttosto sentito.

#### 10.2.3 LSP setup control

I bindings FEC-to-label sono distribuiti solo con l'intento di stabilire cammini label-switched. Si creano quindi 2 differenti modalità che rispondono sostanzialmente a due domande: quale FEC utilizzare per avvertire di un binding? La scelta di una FEC determina quale LSP sono costruiti. Quando avvertire questi binding? Questo determina chi ha il controllo sul setup della LSP.

- **Ordered control:** Egress (o ingress) LSR hanno il controllo sul setup di un LSP. Fornisce una prevenzione dai cicli ma ha come svantaggio la necessità che i bindings si propaghino per tutta la rete prima che si possa stabilire un LSP. Con questa modalità, un LSR effettua un binding tra FEC e label solamente se è un egress router per quella FEC e ha già ricevuto un'informazione di binding per quella FEC.
- **Indipendent control:** Tutti i router fanno l'advertising della FEC indipendentemente (ma devono farlo in una modalità consistente). Il comportamento di default è che si mappi ogni FEC per tutti i prefissi della tabella di routing.

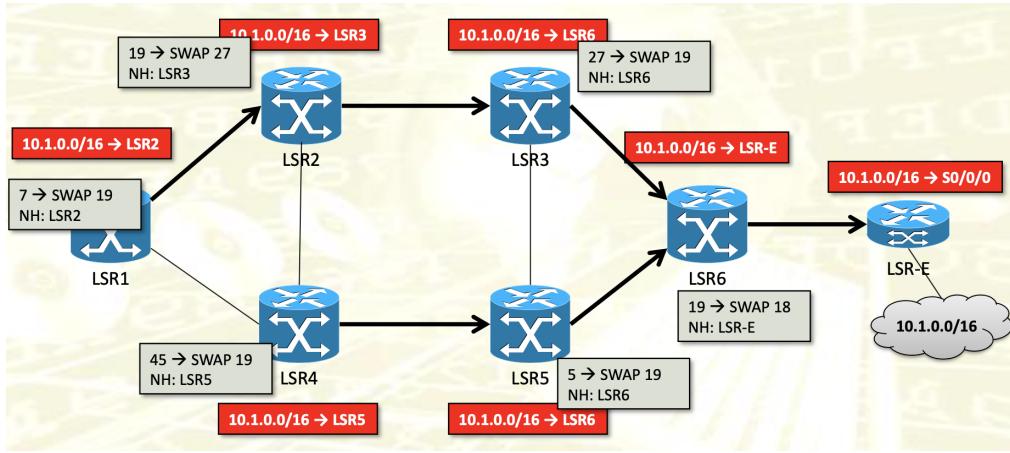


Figure 62: Ordered Control - Unsolicited Downstream

#### Ordered Control - Unsolicited Downstream

- Il setup del cammino parte dall'egress router che costituirà l'ultimo hop di LSP.
- Questo router inizia a fare un broadcasting di messaggi di binding ai suoi vicini per un certo FEC, seguendo un approccio unsolicited downstream.
- Ogni altro router, appena ricevuto un messaggio di binding da un router downstream, utilizza la routing table L3 per scegliere quale binding usare e quale router selezionare come hop successivo. Una volta completata la scelta, genera la NHLFE e inizia a fare broadcasting del suo binding ai suoi vicini.
- Il processo prosegue fino al router ingress da cui il traffico sarà "iniettato" e il cammino viene così formato.

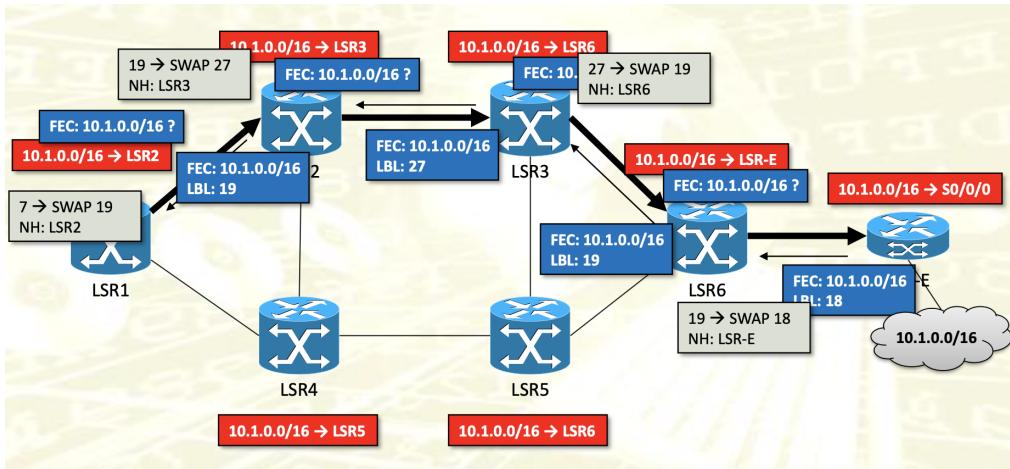


Figure 63: Ordered Control - Downstream On-demand

#### Ordered Control - Downstream on Demand

- Il setup del cammino inizia dal router ingress da cui il traffico è iniettato, appena ha bisogno di scoprire il cammino per un dato FEC.
- Utilizzando le tabelle di routing L3, determina il prossimo hop per quello specifico FEC; poi al prossimo hop chiede che binding vuole usare per quel FEC, seguendo un approccio downstream on-demand.
- Quando riceve la richiesta con le informazioni di binding dal router upstream, il router downstream, a sua volta, ha bisogno di trovare un cammino per il suddetto FEC.

Quindi, a sua volta, determina il prossimo hop dalla sua tabella di routing L3 e gli chiede che binding vuole utilizzare per questo FEC.

- Il processo è ripetuto per tutti i router fino a raggiungere il router egress, che costituirà l'hop finale di LSP. Questo router sa come raggiungere la destinazione finale, seleziona un label per la FEC e comunica il binding al router upstream che lo ha richiesto precedentemente.
- A questo punto l'ultimo router, a sua volta, decide un binding e lo comunica al router upstream e così via, fino al router iniziale: il cammino è finalmente costruito.

Si osserva, quindi, come l'Ordered Control associato all'Unsolicited Downstream: generi un reverse tree e fornisca aggregazione.

Mentre, l'Ordered Control associato ad un Downstream On Demand generi un LSP per ogni ingress router.

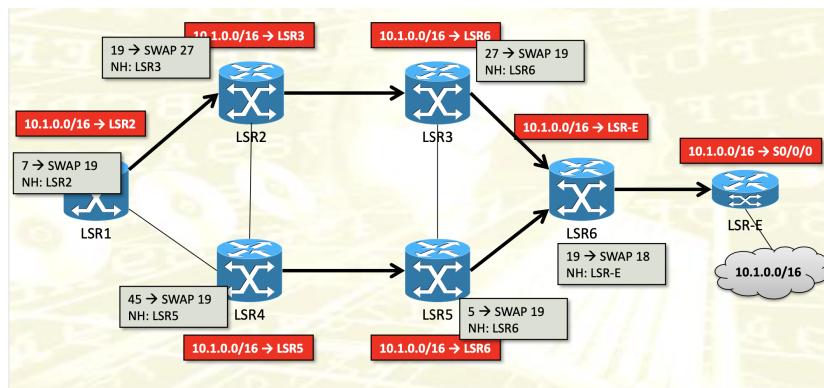


Figure 64: Indipendent Control - Unsolicited Downstream

### Indipendent Control - Unsolicited Downstream

- Ogni router crea e distribuisce i suoi binding indipendentemente.
- All'inizio, quando non c'è ancora nessun binding, un FEC è associato ad un'azione POP e l'indirizzo del prossimo hop è l'indirizzo di loopback. In questo modo, il label MPLS di un pacchetto è rimosso e il pacchetto è passato al livello 3.
- Quando il router downstream decide quale binding usare e lo comunica al router upstream, quest'ultimo cambia l'azione POP con una SWAP del label deciso dal router downstream e setta il downstream come prossimo hop.

Dovendo confrontare Indipendent Control ed Ordered Control, possiamo asserire che il secondo fornisce una prevenzione dai loops, ma ha come svantaggio la necessità che i bindings si propaghino per tutta la rete prima che si possa stabilire un LSP.

Si aggiunge anche una nota che riguarda il modo in cui i router vengono a sapere che sono il penultimo hop e che quindi possono effettuare il PHP. L'ultimo router lo dice esplicitamente, questo è fatto imponendo all'ultimo router di utilizzare un label invertito per il binding. Questo label numero 3, detto *null label*, ogni volta che un upstream router riceve un binding per il label 3, significa che il prossimo router è l'ultimo nell'LSP.

**Aggregazione (MP2P)** È possibile trattare l'unione di più FEC a sua volta come una FEC, ciò risulta utile quando dei LSP condividono lunghi tratti di percorso. In tal caso le N FEC sono dette aggregabili e su di esse si può agire in 3 modi:

- Aggredarle in un'unica FEC.
- Aggredarle in un insieme di M FEC (con  $M < N$ ).
- Non aggredarle

Se si effettua aggregazione, non si hanno solo risvolti positivi nel processo di forwarding, che si troverà a dover operare con tabelle di dimensioni minori, ma si otterrà anche un ulteriore alleggerimento del processo di distribuzione dei label. Si ha quindi la possibilità di scegliere un livello di granularità compreso fra 2 estremi: *coarsest granularity* (aggregazione massima) e *finest granularity* (nessuna aggregazione). Si osservi che: se si utilizza l'ordered control, ogni LSR deve usare la granularità del next hop e se si utilizza l'indipendent control, ogni LSR può usare la granularità che desidera.

#### 10.2.4 Label Distribution Protocol (LDP)

Progettato specificatamente per la distribuzione dei label. Si basa su IGP (Interior Gateway Protocol, il protocollo utilizzato all'interno di un sistema autonomo, ad esempio OSPF), per tutte le decisioni relative al routing. LSR A che riceve un mapping per il label L per il FEC F, dal suo peer LDP; LSR B utilizzerà il label L per il forwarding se e solo se B è nel cammino minimo IGP per la destinazione F dal punto di vista di A.

Maggiori funzionalità:

- Discovery dei vicini (UDP)
- Creazione e mantenimento della sessione (TCP)
- Advertisement del label
- Notifiche

### 10.3 RSVP per la distribuzione dei label

Resource Reservation Protocol. È stato definito per QoS, ma è stato esteso per la distribuzione dei label.

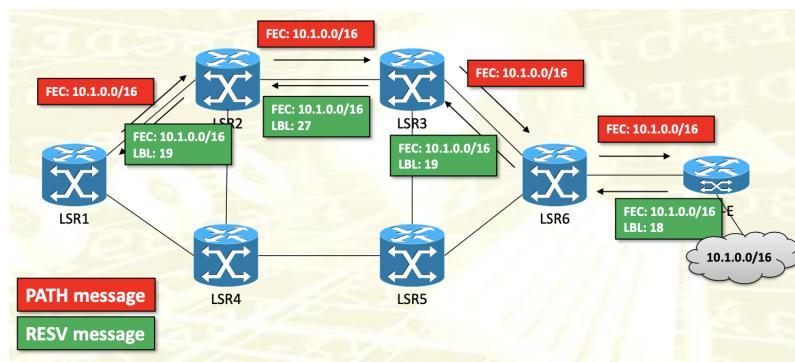


Figure 65: RSVP for label distribution

Ci sono 2 tipi di messaggi: PATH message e Reservation (RESV) message. I messaggi PATH viaggiano dall'upstream al downstream (from HEAD-END a TAIL-END) in un dominio e propagano l'info riguardante la FEC che si vuole stabilire. I messaggi RESV viaggiano poi all'indietro sullo stesso cammino, seguiti da messaggi di path che forniscono informazioni riguardo i binding dei label.

Il protocollo garantisce che RESV seguano esattamente il cammino dei PATH.

#### 10.4 Supporto MPLS per DiffServ

MPLS è una soluzione per fare il forwarding dei pacchetti IP alla loro destinazione, questo è per prendere decisioni riguardo il prossimo hop.

DiffServ è un'architettura network e soluzione di molti problemi, indipendentemente dalle funzioni di forwarding. MPLS è relativo al data plain, ma anche il per hop behaviour è relativo al data plain. MPLS ha il suo header, QoS utilizza quello di IP (campo DSCP per scegliere PHB). Vorremmo un sistema per evitare di doverlo consultare.

Nell'header MPLS, ci sono 3 bit per il traffic class (TC), quindi si va ad utilizzare questo

campo, se ci sono soltanto 8 PHB nel dominio (si codificano i 6 bit di DSCP in questi 3). Che succede se ho più di 8 PHB? Si utilizza lo stesso label: il label indica il prossimo hop e TC come gestire il pacchetto. Quindi si configurano 2 casi:

- Caso fortunato: PHB < 8: Posso utilizzare TC. Quando un pacchetto arriva ad un router, si fa il lookup del label per identificare il prossimo hop, si controlla TC per capire il PHB. Per esempio, quando il label = 001, il router sa che è assicurata la classe di forwarding 1 e quindi tratterà il pacchetto di conseguenza.
- Se abbiamo più di 8 PHB, dobbiamo usare i label anche per distinguere i PHB

Al fine di supportare DiffServ, è necessario che gli LSR possano desumere il PHB da applicare ad un determinato pacchetto appartenente ad un BA (Behaviour Aggregate) esaminando il solo header.

Esistono 2 modalità, il cui impiego è condizionato dal numero di PHB supportati dalla rete:

- **E-LSP:** il PHB viene desunto osservando il solo campo EXP (CoS) nello shim header. Non richiede modifiche alle operazioni di signaling.
- **L-LSP:** il PHB viene desunto osservando i campi label ed eventualmente EXP dello shim header. Richiede modifiche alle operazioni di signaling.

**E-LSP** Qualora siano implementati nel dominio DiffServ/MPLS, solamente 8 PHB, si può utilizzare la modalità *Experimental LSP*. Infatti, nell'architettura DiffServ, il PHB da applicare ad un determinato BA viene deciso ad un router DiffServ mediante l'analisi DSCP all'interno del campo DS: esso è un valore a 6 bit.

In tale implementazione, invece, si riservano solamente 3 bit (del campo Exp) avendo quindi uno spazio di possibili PHB ridotto. Quindi, potrebbero non bastare se i PHB da implementare sono più di 8.

Il pregio di tale soluzione risiede nel fatto che non sono necessari ulteriori meccanismi di signaling: il protocollo di Label Distribution lavora come di consueto; durante le operazioni di forwarding, gli LSR desumono all'interno dello shim header: il PHB da applicare ad un pacchetto consultando il campo Exp e dove effettuare il forwarding mediante il campo label. Si necessita solamente di una preventiva configurazione: così come i PHB venivano mappati su un DSCP, adesso vengono mappati sul campo Exp.

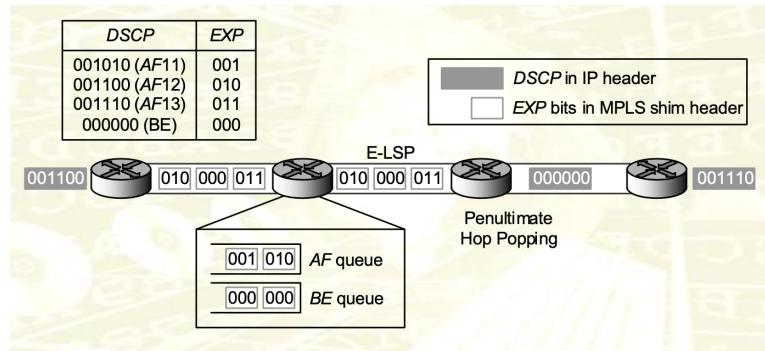


Figure 66: EXP-Inferred LSP

**L-LSP** Qualora siano implementati nel dominio DiffServ/MPLS più di 8 PHB, si può utilizzare la modalità *Label LSP*. Il PHB da applicare viene dunque inferito dal label del pacchetto, si necessita quindi di un enhancement del meccanismo di label distribution. Il protocollo LDP deve essere esteso affinché i messaggi che richiedano o comunichino il binding per una determinata FEC, includano anche le informazioni sul PHB ad essa associato.

Per quanto riguarda il gruppo PHB AF, i pacchetti che condividono un PHB comune vengono riferiti come appartenenti alla medesima PHB scheduling class: essi percorreranno il medesimo LSP in MPLS e la relativa drop precedenza è indicata all'interno dei 3 bit del campo Exp. Questo nel frame mode, mentre nel cell mode si utilizza il campo CLP dell'header della cella.

Per tutti gli altri PHB, ogni LSP trasporterà pacchetti appartenenti ad un singolo PHB. In tali casi, i bit del campo Exp non sono utilizzati.

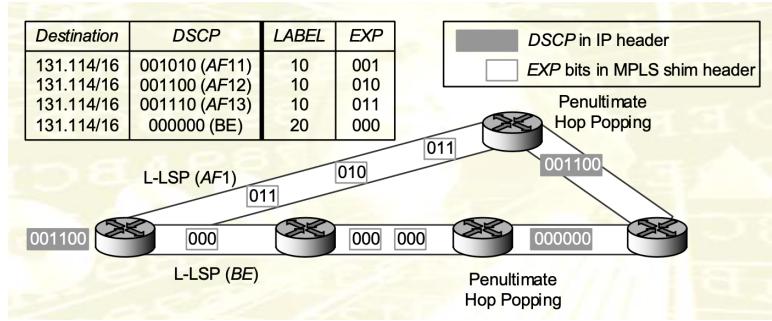


Figure 67: Label-Only-Inferred LSP

# 11 Traffic Engineering

Vogliamo controllare il percorso seguito dal traffico attraverso una rete in modo tale da massimizzare l'utilizzazione di tutti i link cercando, allo stesso tempo, di evitare la congestione della rete. Evitare la congestione significa diminuire le perdite di pacchetti e quindi aumentare l'utilizzo della rete. Per fare ciò abbiamo bisogno di avere il controllo esplicito sul traffico: vogliamo sapere ingresso e uscita dalla rete, decidere il percorso che quello specifico flusso deve seguire ecc. Questo non si può fare con IP classico.

Abbiamo bisogno di algoritmi di routing *constraint-based*, ovvero degli algoritmi e protocolli che minimizzano il costo (secondo una metrica scalare) e che tengano anche conto di determinati vincoli. Alcuni esempi di vincoli sono:

- Performance: trovare un percorso con una certa banda minima disponibile.
- Amministrativi: possibilità di escludere determinati link per questioni amministrative.

Si può anche avere una combinazione di vincoli.

## 11.1 Link Characterization

Se vogliamo modificare un protocollo di routing esistente (tipo OSPF) per supportare il Traffic Engineering, ci serviranno ulteriori informazioni oltre al costo dei vari link. Alcuni dei nuovi parametri sono:

- Max Bandwidth: la capacità del link.
- Max reservable Bandwidth: la quantità riservabile di un link.
- Unreserved Bandwidth: banda ancora disponibile su un link.
- Administrative Group: indica un "gruppo" a cui un link appartiene. In questo modo si può discriminare i vari link se non si vogliono attraversare.

Gli attributi dei link dovranno quindi essere diffusi come parte delle informazioni di routing dal protocollo di routing. OSPF-TE è la versione estesa di OSPF per supportare il Traffic Engineering. Ogni router è a conoscenza degli attributi correnti di tutti i link in una determinata area. Questi dati vengono salvati nel Traffic Engineering Database (TED).

Lo scambio di queste informazioni può essere periodico oppure in risposta ad eventi (cambiamenti dello stato dei link). Tuttavia, se i cambiamenti dei link diventano frequenti, si rischia di sovraccaricare la rete con messaggi di controllo. In questi casi è più opportuno usare un approccio periodico, anche se meno preciso sarà comunque più scalabile.

In OSPF-TE ci sono alcuni attributi che sono statici mentre l'attributo *unreserved bandwidth* cambia frequentemente perché varia ogni qualvolta un flusso riserva della banda. Per questo motivo l'approccio event-based non è scalabile e, in particolare, si utilizza un approccio ibrido.

## 11.2 Protocolli di Routing Estesi

### 11.2.1 CSPF

E' una versione dell'algoritmo di Dijkstra (SPF) estesa per il TE in cui abbiamo i vincoli (Constraint SPF). Inizialmente applichiamo i vincoli a tutti i link nel TED, dopodiché applichiamo l'algoritmo di Dijkstra (SPF) per calcolare il grafo dei percorsi minimi. SPF viene calcolato a partire dal router d'ingresso della rete perché quest'ultimo conosce la topologia.

### 11.2.2 RSVP-TE

Algoritmo che serve per permettere a nuovi flussi di riservare le risorse e allo stesso tempo costruire un percorso. RSVP viene esteso per supportare LSP dopo che è stato ottenuto il grafo dei percorsi minimi tramite CSPF. Il pacchetto contiene la lista dei router da attraversare.

L'ingress router dell'LSP che vogliamo costruire si chiama *head-end* ed ha il compito di iniziare il protocollo mandando un *Path-message* al quale riceverà, come risposta, una *Resv-message*. La differenza rispetto a RSPV tradizionale è che questi due messaggi trasportano informazioni aggiuntive. In particolare, per il path-message abbiamo:

- Label Request Object (LRO): indica che vogliamo creare un percorso e fare label binding ad ogni hop.
- Explicit Route Object (ERO): contiene il percorso da seguire in termini di hops. Questo percorso può essere calcolato direttamente dallo *head-end* in modo da evitare computazioni ad ogni hop. Esiste un bit all'interno dell'ERO che indica se gli hops contenuti sono *strict* o *loose*. Nel primo caso i nodi adiacenti sono direttamente connessi e quindi il percorso sarà esattamente quello descritto dall'ERO. Nel secondo caso lo *head-end* può fornire una lista di nodi tale che due nodi successivi possano non essere direttamente connessi. In questo caso ogni nodo del percorso può calcolare il percorso del next node della lista in maniera autonoma usando CSPF.
- Record Route Object (RRO): contiene lo stato relativo a questa operazione.
- Sender TSpec: contiene la traffic demand.

Per il resv-message abbiamo:

- Record Route Object (RRO)
- Label Object: specifica la label scelta da un router per questo LSP che deve essere comunicata al router soprastante (upstream) nel percorso.

Per evitare situazioni di conflitto sulle risorse, esiste l'Admission Control su ogni hop. Quando un percorso scelto tramite CSPF viene scelto, gli viene garantito che ci siano abbastanza risorse lungo il percorso. Questo nella realtà non è sempre vero in quanto le informazioni contenute nella TED potrebbero non essere aggiornate.

Le LSP forniscono 8 livelli di priorità. In realtà si utilizzano solo 2 priorità:

- **setup priority:** per percorsi in costruzione.
- **hold priority:** per percorsi che sono già attivi.

Se due flussi vogliono riservare delle risorse contemporaneamente, ma non ci sono abbastanza risorse per entrambi, la setup priority è usata per discriminare quale dei due sarà accettato. Poiché le condizioni della rete cambiano in maniera dinamica, il percorso ottimale per un flusso potrebbe cambiare nel tempo. Per questo motivo è necessario ri-eseguire CSPF in modo tale da aggiornare le LSP. Questo processo è chiamato ri-ottimizzazione.

### 11.2.3 Fast Re-Routing

Esempio di applicazione di MPLS-TE. In caso di fallimenti di rete, nelle reti tradizionali quello che succede è che l'elemento che ha rilevato il problema informa tutti gli altri che, a loro volta, ri-calcoleranno i percorsi. Il problema di questo approccio è che impiega molto tempo.

Grazie a fast re-routing si può trovare un percorso alternativo e temporaneo per sopperire (seppur parzialmente) al malfunzionamento della rete fino a che non verrà calcolato un nuovo percorso ufficiale. Tutto questo viene realizzato avendo un LSP di backup pre-calcolato per ogni link: quando viene rilevato un fallimento, l'LSP di backup viene usato immediatamente sfruttando il tunneling MPLS per bypassare il link che è fallito.

## 12 BGP - Border Gateway Protocol

L'architettura attuale di Internet prevede diverse autorità che controllano differenti porzioni della rete in modo indipendente (Autonomous Systems). Data questa struttura complessa con differenti blocchi autonomi, il nostro obiettivo è, dato un pacchetto con indirizzo di destinazione, trovare un metodo per inviare quel pacchetto al dominio (AS) che contiene l'indirizzo destinazione. Si può ottenere tramite BGP.

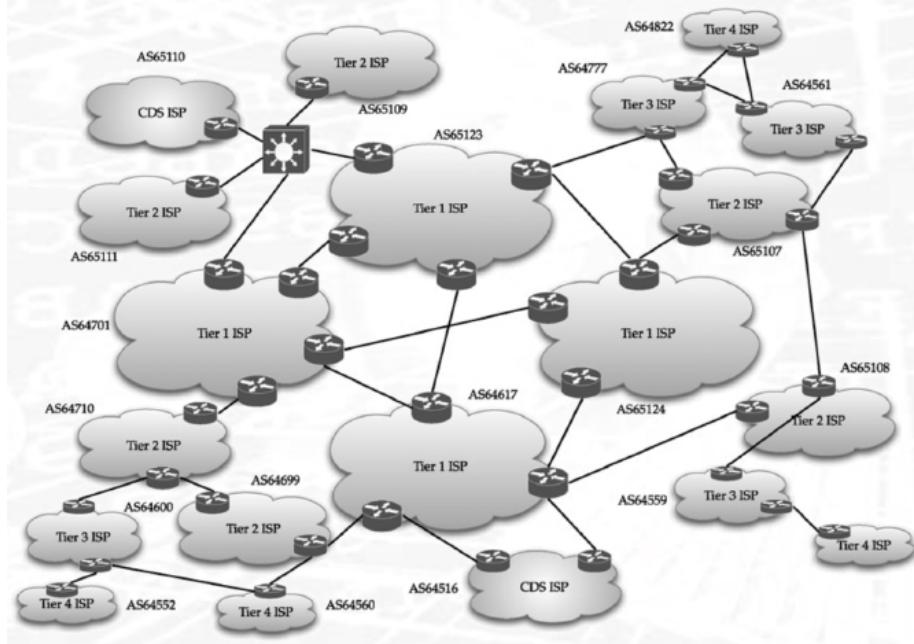


Figure 68: Architettura di Internet con AS

BGP è un'istanza dell'algoritmo *Path Vector Routing*. Non è un algoritmo di tipo link-state né di tipo distance-vector in quanto:

- (link-state) costruire la topologia dell'Internet globale ad ogni nodo e calcolare il percorso minimo è una soluzione non-scalabile.
- (distance-vector) c'è una buona probabilità che emergano cicli che sarebbero pericolosi in una topologia così grande.

In BGP consideriamo ogni AS come un nodo della rete, che può essere collegato ad altri AS tramite archi (link che collegano i border-router). Quello che vogliamo trovare è quindi il "next-hop" che in questo caso sarà il next-AS del percorso.

Ogni nodo AS scambia con i suoi vicini dei messaggi che contengono una destinazione nota e la lista di nodi (AS) che formano il miglior percorso conosciuto verso quella destinazione. Questo è una differenza importante rispetto al distance-vector perché aiuta a prevenire i cicli: se un nodo  $x$  riceve un *path-vector* in cui  $x$  stesso è incluso, il messaggio deve essere scartato altrimenti si forma un ciclo.

Il costo di ogni hop è sempre pari ad 1 anche se, tramite un trucco, è possibile aumentarlo: basta ripetere l'id di un nodo più volte nel percorso. In questo caso si parla di *path manipulation*.

Queste idee vengono messe in pratica nel protocollo BGP-4. I nodi (AS) della rete non s'parlano direttamente bensì sono i border router (BR) a scambiarsi tutti i messaggi tramite una connessione TCP. Possiamo avere più BGP peers (border-router che scambiano messaggi BGP) all'interno di uno stesso AS oppure in due AS diversi:

- **E-BGP (esterni):** i due BR appartengono ad AS diversi.
- **I-BGP (interni):** i due BR appartengono allo stesso AS. Questo serve perché i BR potrebbero essere connessi a due AS diversi pertanto, ci sarà necessità di scambiarsi i differenti pezzi di informazione raccolti.

Fra i vari messaggi BGP che si scambiano i router, di particolare interesse sono gli UPDATE, scambiati periodicamente. Un messaggio di tipo BGP-UPDATE contiene una lista di *Path Attributes* che codificano le informazioni che rappresentano un percorso verso una certa destinazione. Alcuni di questi attributi sono:

- ORIGIN: da dove è partito il messaggio.
- AS\_PATH: lista di ID degli AS che compongono il percorso verso la destinazione.
- NEXT\_HOP (next-as).
- LOCAL\_PREF: serve in caso vi siano più percorsi verso una destinazione. Grazie a questo campo possiamo impostare una rotta "preferita".

Un'altra informazione inclusa nei messaggi BGP-UPDATE è il **Network Layer Reachability Information** che contiene la lista delle reti raggiungibili tramite il percorso descritto nel campo AS\_PATH.

## 13 VPN

Il problema delle cosiddette "Corporate WANs" consiste nel voler collegare fra loro delle reti che si trovano molto distanti l'una dall'altra. Vogliamo che il collegamento le faccia risultare vicine. Il problema è che, essendo distanti, non esiste un collegamento diretto fra queste reti pertanto, sarà necessario implementare un meccanismo che simuli la "vicinanza".

Tale rete non potrà essere pubblica, ma privata, ovvero deve essere utilizzabile solo dalla compagnia e deve avere un meccanismo di routing ed uno spazio degli indirizzi indipendenti dal resto delle reti con cui condivide l'infrastruttura mediante la quale è realizzata.

### 13.1 Concetti generali

**Overlay VPN** In questo modello, ogni sito ha un router che è connesso mediante un link punto punto virtuale ad altri routers locati in altri siti. La gestione dell'overlay è fornita dal service provider. In questa connessione punto-punto, i router si scambiano pacchetti del routing protocol in modo da scambiarsi informazioni riguardo la network reachability. Questo si può fare a più livelli: stendendo linee fisiche (livello 1, anche se in questo caso non si tratterebbe più di rete virtuale), oppure al livello 2 (con, ad esempio, una tecnologia Frame Relay, packet switched). Questo tunneling può essere fatto anche al livello 3 (IP over IP tunneling, usando protocolli come Generic Route Encapsulation o IPsec).

Si ottengono risultati sperati, ovvero: connettività tra i siti, indirizzi privati (no vincoli nell'addressing) e traffico privato. Si ottengono tuttavia svantaggi da questo approccio. I Customer Edge Routers (CE) possono essere gestiti dal cliente, però devono saperli connettere e configurare, cosa non scontata. Problemi di scalabilità: occorre un router per sito, quindi può non essere facile fare manutenzione della configurazione su decine di migliaia di router, che si trovano in luoghi diversi. Se si deve aggiungere un nuovo sito, su richiesta del cliente, si deve configurare non solo il router appartenente al nuovo sito, ma anche i router precedenti, che devono quindi essere in grado di raggiungere il nuovo sito. È quindi complicato mantenere l'overlay.

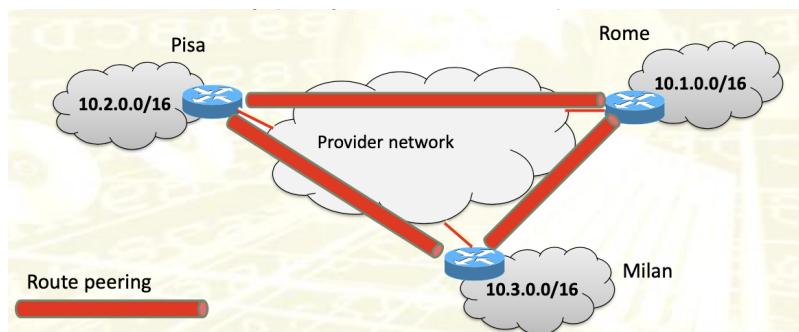


Figure 69: Overlay VPN

**Peer-to-peer approach** I peering sono stabiliti fra router CE e solo con il router edge del provider (PE). Quindi, ogni router del customer, scambierà informazioni di routing solo con il provider edge router a cui sono connessi. Quindi, indipendentemente dal numero di "siti" del customer, il numero di peer di ogni customer edge router sarà uno. Ora la configurazione di CE è estremamente semplice: all'esterno del sito del customer, hai un unico next hop possibile. Prima ne avevamo tanti, ovvero quanti il numero di virtual tunnel/virtual link, tramite i quali eri connesso agli altri siti.

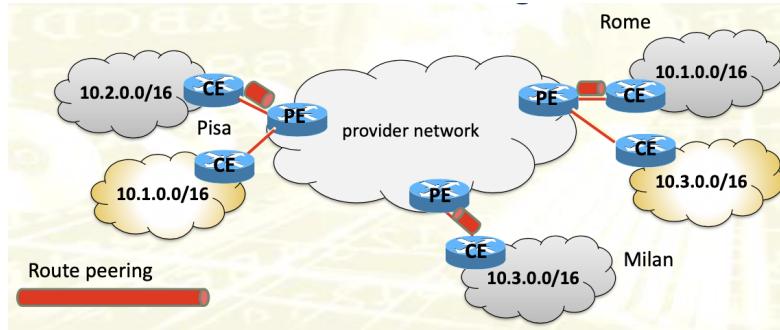


Figure 70: Peer-to-peer approach

Il problema di questo approccio riguarda la figura 70, in cui i colori gialli delle nuvole indicano che appartengono ad altri clienti rispetto a quelli delle nuvole grigie: 5 siti, 2 clienti. Questi clienti fanno un indirizzamento indipendente, ci sono delle sovrapposizioni. Quindi, i problemi sono: come realizzare private addressing e isolamento del traffico.

Una soluzione potrebbe essere quella di inserire dei filtri nei Provider Edge Routers. Un pacchetto originato, ad esempio da 10.2.0.0 "grigio", se è destinato ad un sito "grigio" viene permesso, altrimenti viene filtrato. In pratica, si configura una grande ACL all'interfaccia di ingresso di ogni PER, per controllare se il traffico può entrare nella rete o meno: questo in base al fatto che sia indirizzato ad uno dei siti consentiti o meno (assumendo che si sappia la source). Questa non è una soluzione buona, dato che si considera la network del provider come un "single routing domain", non si possono avere più site con lo stesso indirizzo: si perderebbe così il private addressing oppure, al più, dovremmo implementare delle complicate traduzioni. Si raggiungerebbe comunque un isolamento del traffico, ma ad un costo molto alto, dato che si dovrebbero mantenere dei complessi ACL ad ogni PE. Questa configurazione dovrebbe poi essere continuamente revisionata, dato che ogni volta che cambia la topologia di rete, si devono riconfigurare i PE (le loro ACL). Le ACL, inoltre, rallentano il routing e aggiungono latenza. Quindi, questa soluzione è di difficile applicazione. Cercando un'altra soluzione, l'effetto che si vorrebbe ottenere, è quello che se c'è un pacchetto che va da CE a PE, questo venga consegnato solo se la routing table che è usata per quel pacchetto, che include solo gli indirizzi di destinazione degli altri siti dello stesso customer, sa qual è il next hop di quel pacchetto. Quindi, sempre nella figura 70, se consideriamo il site 10.3.0.0/16, appartenente ad un cliente "giallo", se questo manda un pacchetto con destinazione 10.2.0.0/16, ci si aspetterebbe che il PE a cui è attaccato filtri questo pacchetto, in quanto nella rete del cliente giallo, non ci sono sites con tale indirizzo. Come si può ottenere questo effetto?

La soluzione è tenere diverse routing table nel PE, una per ogni customer. E' come se si considerassero più router logici, uno per ogni cliente. Si controlla nella tabella giusta, in base al cliente che ti sta inviando quel pacchetto. Questo si chiama *Constrain Routing*, perché i PE si scambiano informazioni, che saranno diverse per diversi clienti, quindi durante lo scambio delle informazioni occorrerà mettere informazioni relative ad un certo cliente, nella sua routing table. C'è ancora un problema di overlapping, quando una rete manda un pacchetto con un indirizzo, presente in 2 clienti diversi, si crea un problema di forwarding per siti con indirizzi in overlapping. La soluzione quindi è quella di utilizzare un tunneling tra provider edge router. È una soluzione standard che troviamo negli RFC, chiamata *BGP/MPLS IP Layer 3 VPN*.

## 13.2 BGP/MPLS IP VPN

Considerando lo scenario presente nella seguente figura:

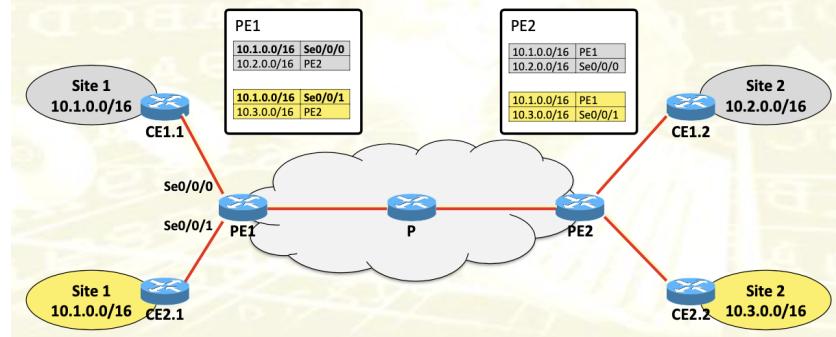


Figure 71: Rete BGP/MPLS IP VPN

Il customer deve solo avere un link al PE e notificare com'è fatta la sua rete. Nella figura 71 si ha P che è un Core Router. In ogni Provider Edge Router si hanno diverse tabelle, una per ogni Customer Edge Router connesso. Si distinguono i pacchetti dei diversi clienti in base all'interfaccia sulla quale arrivano. Queste tabelle sono chiamate *Virtual Routing Forwarding Table (VRF)*. Quindi nel PE1, troveremo una VRF per CE1.1 e CE2.1, con questo approccio è come avere molteplici router virtuali, c'è poi una routing table per il global forwarding, ovvero per forwardare traffico non destinato a siti customers.

Quindi, se il site1 (grigio), manda un pacchetto all'indirizzo 10.3.1.1, il pacchetto va prima a CE1.1, poi a PE1 che, in base alla interfaccia di provenienza, sa che deve guardare la tabella di routing per il cliente 1. Non troverà un'entrata per 10.3.0.0/16 e quindi scarterà il pacchetto. Come fa PE1 a sapere che, ad esempio, 10.2.0.0/16 può essere raggiunta tramite PE2? Glielo deve dire PE2, occorre un dynamic routing protocol, ovvero PE2 avverte PE1 che quell'indirizzo è raggiungibile attraverso di lui, come fanno i normali protocolli di routing. Tuttavia, adesso si devono scambiare informazioni anche riguardo la proprietà dell'indirizzo, customer giallo o grigio. Questo può essere fatto facendo girare un'istanza del protocollo di routing per VPN. In tal modo è come avere n reti separate, anche se questo approccio non scala, dato che si devono configurare tante istanze di routing protocol per quanti customer si hanno.

Si vuole quindi avere una sola istanza del protocollo di routing. In questo caso entra in gioco BGP.

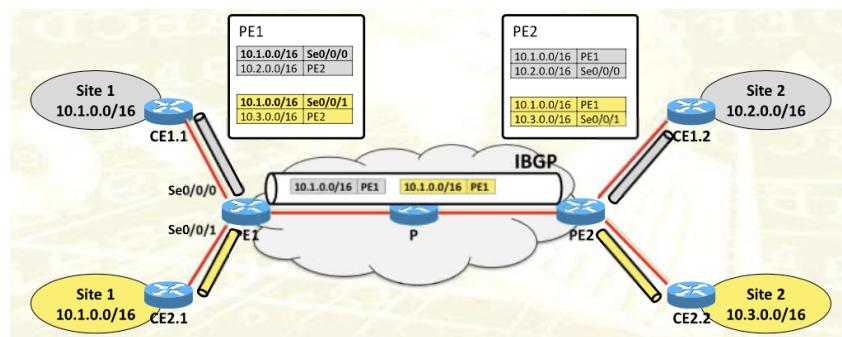


Figure 72: BGP Route Distribution

Tramite un BGP update, PE1 dirà a PE2 quali destinazioni sono raggiungibili tramite di lui, specificando anche il site. Si sta, quindi, informando il PE2 che il PE1 è il next hop per certe destinazioni. Il problema però sorge perché con BGP standard, non si può, nella lista, mettere 2 volte lo stesso indirizzo di destinazione. Non c'è spazio per marcare un indirizzo come "grigio" piuttosto che "giallo".

La soluzione risulta quella di definire nuove famiglie di indirizzi VPN-IP: un nuovo schema di indirizzamento, dove aggiungiamo alla destinazion network un prefisso, chiamato *Route Distinguisher (RD)*, di 8 byte che deve essere unico globalmente.

TYPE (2)	AS number (2)	Locally assigned number (4)
----------	---------------	-----------------------------

Figure 73: Route Distinguisher

Grazie a questo RD, adesso abbiamo route distinti (per colore) e non si hanno problemi con BGP.

Ogni router può usare un RD differente a quello degli altri router per riferirsi a una tabella. Se si usasse RD per ogni VPN, non si avrebbe questo problema, ma si assume che non si stia utilizzando questa soluzione. Quindi, RD non può essere usato come indice nelle VRF, ma viene usato solo per differenziare i messaggi in BGP.

Un altro problema è: si vuole più flessibilità, si potrebbe aver bisogno di comunicare tra un host della VPN gialla e uno della grigia. La soluzione a questo problema è raggiunta tramite *Route Target*: è un altro numero, differente da RD. Viene trasportata in un altro attributo di BGP, il *Community Attribute*. Quando arriva un annuncio a un router, per ogni tabella del router viene controllato il Route Tag: se quel Route Target è consentito per quella tabella, allora l'indirizzo viene aggiunto in quella tabella, altrimenti no.

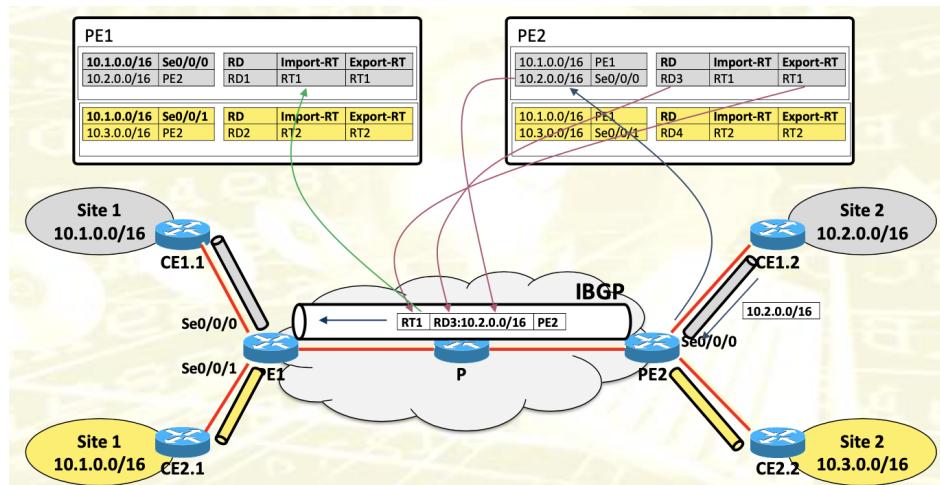


Figure 74: Route Target

Nelle tabelle si hanno due valori in RT, per import ed export.

- **Import-RT:** specifica quale route vogliamo importare in un dato VRF: se il set di Import-RT include almeno uno degli RT nel messaggio BGP, l'informazione contenuta nel messaggio BGP, è usata per aggiornare la VRF
- **Export-RT:** è la lista di RT che devono essere incluse in un messaggio BGP che annuncia le rotte da questa VRF.

Questa modalità di utilizzare Route Target è chiamata *Full-mesh*.

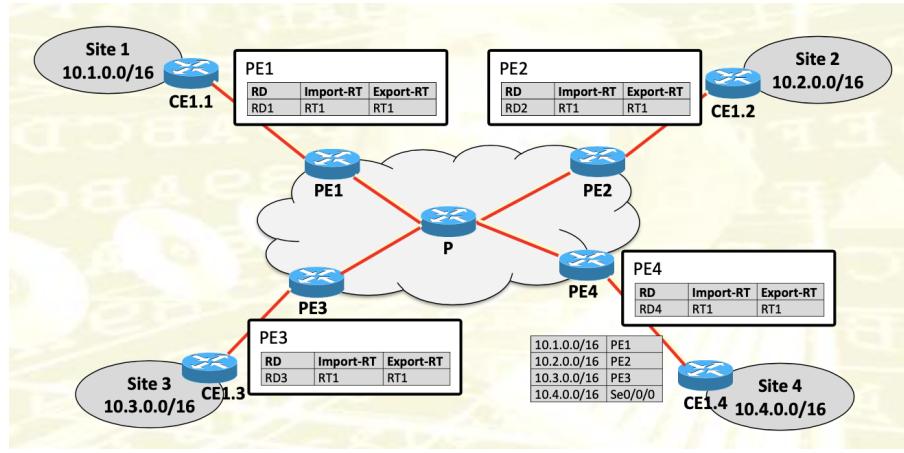


Figure 75: Full-mesh VPN

Adesso, bisogna vedere come inoltrare i pacchetti. Dato che nel mezzo a dei PE, potrebbero esserci delle reti IP, quindi con destination-based forwarding, risulta quindi impossibile proseguire. È necessario fare tunneling tra PE (MPLS tunnel). I router interni scambiano solamente i pacchetti tra i PE, non sanno nulla dei VPN-IP e delle VRF, quindi torna di nuovo il problema dell'overlapping degli IP. Giunto ad un PE, l'informazione "giallo o grigio", non può essere trovata nel pacchetto IP (dato che è presente il tunnel). L'idea è quindi quella di avere più tunnel, uno per VPN. Quindi, se un pacchetto è appartenente alla VPN "gialla", attraverserà l'LSP "giallo".

Si può, però, fare di meglio attraverso il label stacking. Si avrà un tunnel tra PE1 e PE2 e si avrà lo stack di multipli LSP su questo tunnel, uno per ogni VPN.

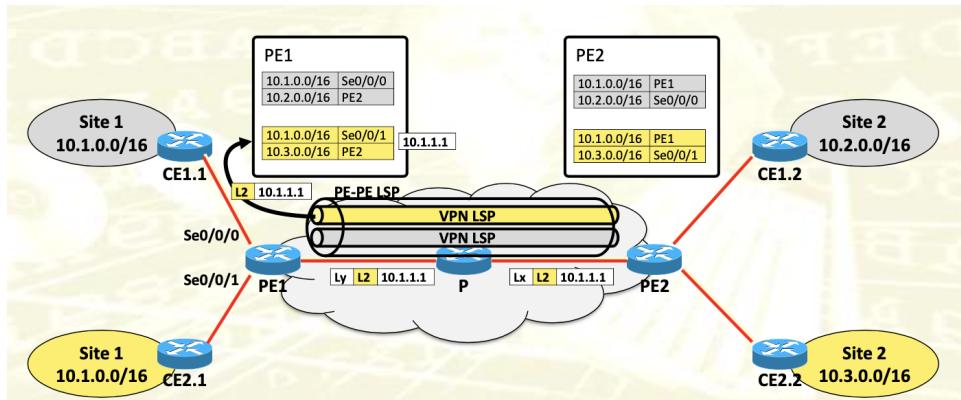


Figure 76: MPLS label stacking su VPN

Lx ed Ly individuano il tunnel tra PE1 e PE2, L2 individua la VPN giusta. Come sapere a chi corrisponde L2?

Questa informazione la propaga BGP. Il messaggio BGP contiene RD, RT (per il route filtering), label che indica quale routing table usare. Per aggiungere un nuovo site, con questo approccio, è necessario ritoccare la configurazione del PE a cui si sta connettendo: gli deve comunicare gli RT di import ed export. Queste info vengono poi passate fra i router col protocollo di routing.

Come si può linkare una label ad una specifica tabella? Mediante MP-iBGP, il router PE2 indica nel messaggio di MP-iBGP update, quale label deve essere utilizzato come VPN label.

**Benefici di BGP/MPLS IP VPN** Per quanto riguarda il customer:

- Sbarazzarsi e scaricare la gestione del routing al provider
- Avere servizi di regolamento degli accessi (firewall ecc..)

Per quanto riguarda il provider:

- Serve multipli clienti VPN con una infrastruttura comune
- La gestione di VPN è nascosta nel core
- Scala aggiungendo PE quando necessario

Il tunneling di MPLS gioca un ruolo chiave per consentire tutto questo.