

Login

Il 29 ottobre 1969 alle 22.30, nello Stanford Research Institute di Menlo Park (California), il programmatore Bill Duvall riceve sul suo calcolatore (uno Scientific Data System 940), la stringa «lo».

A inviarla è stato Charles Kline, suo collega dell'Università della California di Los Angeles.

Il sistema di comunicazione, infatti, dopo aver inoltrato correttamente la «l» e la «o», va in crash e non riesce a spedire «gin».

Questa è stata la prima parola «pronunciata» dalla rete Internet.

La supervisione del progetto fu a cura di **Leonard Kleinrock**, che impiegò il primo rudimentale router denominato **IMP** (*Interface Message Processor*).

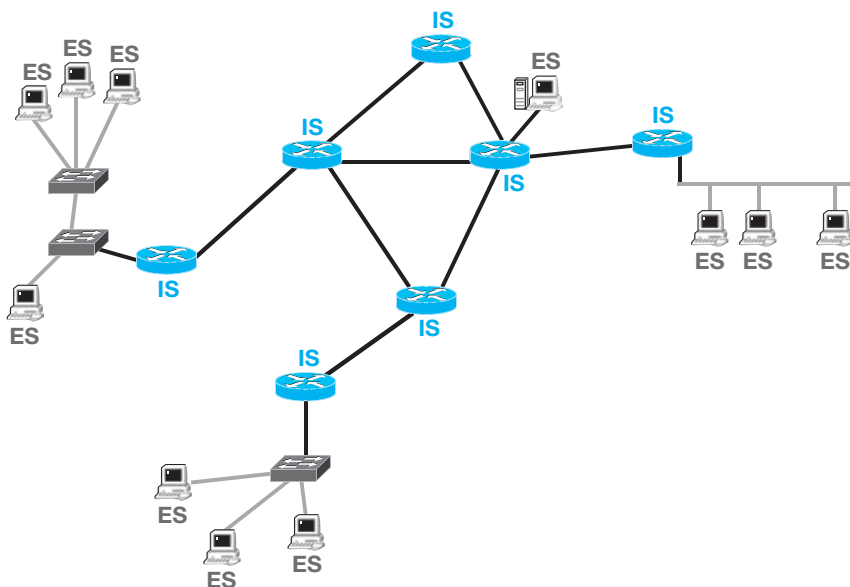
A partire dal 5 dicembre dello stesso anno si collegarono alla rete anche l'Università della California di Santa Barbara e l'Università dello Utah, come stabilito nel progetto originario.

In una rete con topologia a maglia come una WAN, si pone il problema di far raggiungere la destinazione ai pacchetti dei messaggi, destinazione che non è direttamente connessa al mittente come nei modelli LAN.

In altri termini la rete magliata deve fornire un meccanismo per individuare un cammino valido da mittente a destinatario per qualsiasi coppia di stazioni **host** (o **ES**, *End System*) presenti sulla rete.

Questo è il compito del **livello 3 Rete** (*Network Layer*).

Il cammino deve coinvolgere quindi stazioni intermedie, denominate **router** (o **IS**, *Intermediate System*), che rappresentano i nodi intermedi della rete e che posseggono più di una interfaccia fisica per almeno due connessioni su reti differenti.



I nodi intermedi devono quindi possedere una serie di capacità che consentano di instradare i pacchetti da una interfaccia di provenienza a una interfaccia di destinazione.

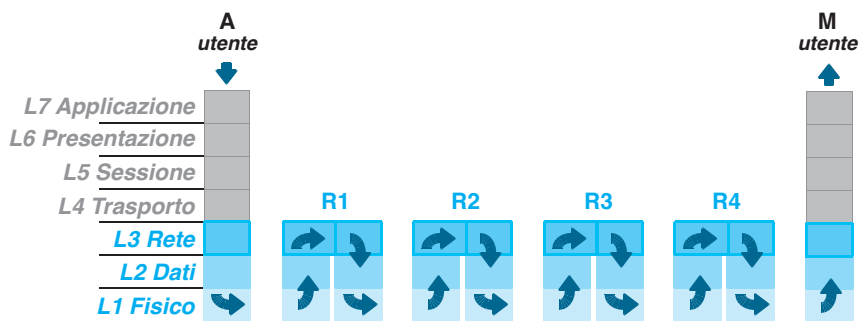
Elementi base dei protocolli di livello 3 Rete (Network) sono quindi:

1. l'apparato **router** e le sue porte di inoltro;
2. gli **indirizzi** contenuti nei pacchetti di livello 3 Rete;
3. la **tabella di instradamento** (*TdI*), cioè i dati per decidere i cammini;
4. l'**algoritmo di instradamento**, le regole per creare la TdI.

Su un nodo intermedio IS deve essere implementato un algoritmo di instradamento (procedure) a sua volta basato su una tabella di instradamento (struttura dati). In sintesi:

Sulla ricezione di un pacchetto, il livello 3 Rete operante sugli IS apre la PDU (il pacchetto) di livello 3 per estrarne gli indirizzi di livello 3, quindi avvia l'algoritmo di instradamento che aggiorna la tabella di instradamento da cui decidere la porta di inoltrare.

Per quanto esposto, la problematica dell'instradamento affrontata al livello 3 di rete viene quasi del tutto affrontata e risolta dai nodi intermedi IS, mentre i nodi ES non implementano in modo sostanziale il livello 3. D'altra parte i nodi IS non hanno spesso ragione di occuparsi delle problematiche dei livelli superiori al terzo, pertanto uno schema a livelli delle stazioni su una rete WAN che descrive una circolazione dall'host **A** all'host **M** risulta essere (vedi anche la figura del paragrafo **Routing CNLS: by network address**):



Si può anche affermare che il livello 3 Network **isola i dettagli dell'hardware di rete** e li nasconde ai livelli superiori: gran parte del livello 3 e tutti i livelli superiori al 3 sono esclusivamente software.

1 Routing

La tabella di instradamento è una struttura dati di semplice concezione, costituita da un certo numero di righe che contengono le **regole** con cui decidere su che porta dell'IS un determinato pacchetto deve essere inoltrato. Questa operazione è detta **routing**.

Ogni riga è costituita da alcuni campi, tra cui *Indirizzo* e *Porta*.

Quando riceve un pacchetto di livello 3, l'IS ne estrae l'indirizzo destinazione, consulta le righe della TdI alla ricerca di tale indirizzo (nella colonna *Indirizzo*) e, individuata la relativa riga, determina la porta ove instradare dal campo *Porta* corrispondente.

Il campo *Indirizzo* può prevedere anche gruppi di indirizzi, in modo tale che non sia necessario avere una riga per ogni singolo nodo della rete.

Inoltre è possibile che esistano più regole che contengono lo stesso indirizzo (ma differente numero di porta), affinché si possano avere alternative di instradamento in caso di porte (o linee) guaste.

Gli altri campi delle righe di una TdI riguardano le cosiddette **metriche**, ovvero una metrica di **costo**, valutata sia in base ai costi economici delle varie linee, sia in base all'efficienza o velocità delle linee; e una metrica di salto (**hops**), che segnala quanti nodi intermedi devono essere attraversati prima della destinazione.

I campi di metrica servono per stabilire l'instradamento in caso di alternative di linea (più regole per la stessa destinazione) e, generalmente, la consuetudine suggerisce di privilegiare prima le linee con minor costo, quindi quelle con minor hops.

Gran parte dell'efficienza e delle caratteristiche degli algoritmi di routing si basa sui modi in cui la TdI viene creata e mantenuta aggiornata dagli IS.

La TdI ideale dovrebbe contenere tante righe quanti sono gli host di una rete e dovrebbe essere disponibile per ogni IS della rete.

Inoltre, a ogni modifica della topologia, ovvero a causa del succedersi di linee interrotte, nuove linee, nuovi IS e IS eliminati, nuovi ES ed ES eliminati, la TdI dovrebbe essere aggiornata immediatamente su tutti gli host.

Tutto ciò non è possibile su reti magliate anche di piccole dimensioni, pertanto gli algoritmi di routing in dotazione agli IS non possono operare in senso deterministico ma devono adeguarsi a criteri sostanziali di funzionamento, tra i quali:

- *Robustezza e versatilità*, per adattarsi senza particolari oneri alle variazioni della topologia.
- *Stabilità*: l'algoritmo deve convergere, cioè deve trovare un cammino in tempi ragionevoli; se non ci sono cambiamenti sulla rete, l'instradamento deve essere immediato usando la TdI.
- *Equità*: l'algoritmo nel calcolare la TdI deve evitare di danneggiare particolari nodi, anche se per questo deve utilizzare qualche cammino non ottimale.

Inoltre si possono stabilire degli obiettivi che costituiscono l'ottimalità di un algoritmo di instradamento, anche se spesso sono in contrasto fra loro:

- Minimizzazione del ritardo medio di ogni pacchetto (dal punto di vista dell'utente di rete).
- Massimizzazione dell'utilizzo della rete (dal punto di vista del gestore di rete).
- Possedere un certo grado di tolleranza ai guasti (*fault tolerance*).

Esistono sostanzialmente due approcci al problema dell'instradamento che, pur essendo simili per quanto riguarda il problema della ricerca del cammino, in seguito adottano tecniche di ritrasmissione completamente differenti; i due approcci riflettono le due categorie di reti che abbiamo già descritto più volte: reti non orientate alla connessione (*connectionless* o **CNLS**) e reti orientate alla connessione (*connection oriented* o **CONS**).



Hops

Dato lo schema nella figura del paragrafo [Routing CNSL: by network address](#), scrivere un programma in linguaggio C che, digitati due numeri indicanti il router di partenza e il router di arrivo, stampi il percorso dell'ipotetico pacchetto.

OUTPUT

```
Digitare router sorgente (1..4): 3
Digitare router destinazione (1..4): 1
Tratta: R3 R2
Tratta: R3 R2 R1 ok
```

Per rappresentare i collegamenti dei router R1, R2 R3 e R4 della figura si può usare una serie di triplette del genere:

R1,R2,2, cioè "sulla porta P2 di R1 è connesso il router R2",

R2,R1,4, cioè "sulla porta P4 di R2 è connesso il router R1", ecc...

In tutto risultano **6** triplette.

Il codice in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01 #include <conio.h>
02 #include <string.h>
03
04 typedef struct {
05     char ra[3];          // Il router ra è connesso...
06     char rb[3];          // ...al router rb ...
07     unsigned char p;    // ...sulla porta p (di ra)
08 } LINK;
09 LINK linkrouter[] =
10     {{ "R1", "R2", 2 }, { "R2", "R1", 4 }, { "R2", "R3", 2 }, { "R3", "R2", 5 }, { "R3", "R4", 3 }, { "R4", "R3", 1 } };
11
12 int main (void)
13 {
14     char chS, chD, szHops[100], *pc;
15     int i, rm, rd, r, ok, n = 0;
16
17     printf("Digitare router sorgente (1..4): ");
18     chS = getche();
19     printf("\nDigitare router destinazione (1..4): ");
20     chD = getche();
21
22     rm = chS - '0';
23     rd = chD - '0';
24     r = -1;
25     szHops[0] = 0;
26     pc = szHops;
27     n = n + sprintf(pc+n, "R%c ", rm + '0');
28
29     while ((rm!=rd) && (n<70))
30     {
31         ok = -1;
32         for (i=0; i<sizeof(linkrouter)/sizeof(LINK); i++)
33         {
34             if (linkrouter[i].ra[1] == rm + '0')
35             {
36                 ok = i;
37                 if (linkrouter[i].rb[1] != r + '0')
38                 {
39                     ok = -2;
40                     r = rm;
41                     rm = linkrouter[i].rb[1] - '0';
42                     break;
43                 }
44             }
45         }
46         if (ok>=0)
```

```

46     {
47         r = rm;
48         rm = linkrouter[ok].rb[1] - '0';
49     }
50     if (ok== -1)
51     {
52         printf("\nImpossibile instradare");
53         return 0;
54     }
55
56     n = n + sprintf(pc+n, "R%c ", rm + '0');
57     printf("\nTratta: %s", szHops);
58 }
59
60 (rm==rd) ? printf(" ok") : printf("\nErrore TTL!");
61 return 0;
62 }

```

Le triplette vengono controllate individuando la prima che consente al pacchetto di uscire dal router ma non sulla porta da cui è entrato (**righe 33÷44**).

Se nessuna tripletta soddisfa la condizione, si prende la prima tripletta d'uscita dal router, anche se è quella di provenienza (**righe 45÷49**).

Si nota che:

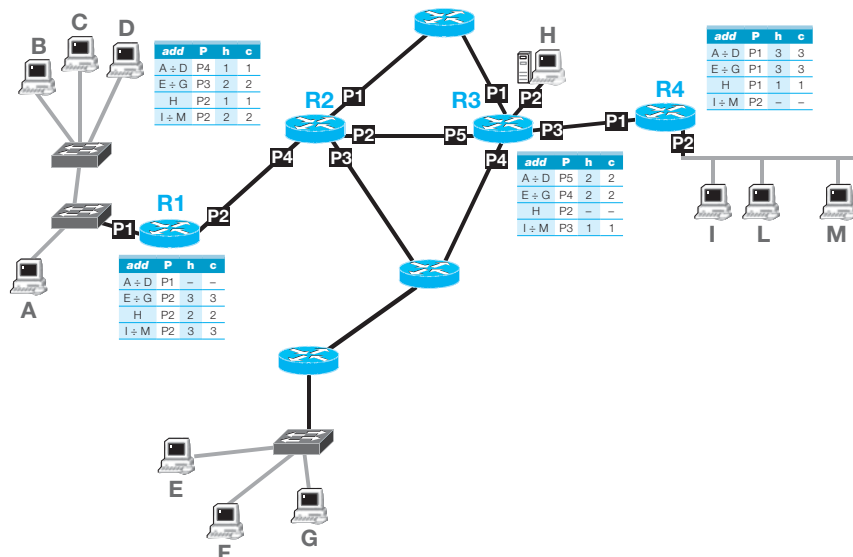
- a** se il mittente non è R1 e la destinazione sta alla sua “destra” (per esempio da R2 a R4), il programma “torna indietro” e poi risale verso la destinazione;
- b** se il mittente non è R4 e il destinatario sta alla sua “sinistra” (per esempio da R3 a R1), il programma agisce razionalmente;
- c** se invece si propone un input illecito tipo da R1 (1) a R9 (9), il programma continua a cercare il percorso indefinitamente (salvo segnalare la situazione con il messaggio “Errore TTL”).

Questa situazione è realistica, come si vedrà più oltre (cfr. **TTL in Pacchetto IP**).

2 Routing CNLS: by network address

Tramite il routing **by network address** il mittente e il destinatario dei pacchetti sono specificati apponendo in ogni pacchetto di L3 Rete i relativi indirizzi. Il pacchetto di L3 Rete rimane immutato per tutto il percorso, mentre la trama L2 Datalink viene modificata per ogni singola tratta attraversata adattandosi allo strato fisico della nuova linea: cambiano quindi gli indirizzi di livello 2. Inoltre l'IS (che nelle reti CNLS viene anche detto semplicemente **router**) deve prendere la decisione di instradamento per ogni pacchetto, consultandone l'indirizzo L3 di destinazione: gli IS usano l'indirizzo del destinatario del pacchetto come chiave per la propria TdI per determinare la porta su cui effettuare il successivo instradamento.

Questa operazione viene ripetuta a ogni transizione per un IS e per ogni pacchetto, perciò implica un rallentamento nel cammino. I pacchetti di un singolo mittente possono prendere direzioni anche differenti e arrivare alla stessa destinazione attraverso cammini diversi. Per due pacchetti destinati allo stesso host, la TdI dell'IS potrebbe essere cambiata in funzione di nuovi calcoli dovuti alla modifica della topologia (per esempio guasti o nuovi nodi), e quindi il cammino dei due pacchetti potrebbe essere diverso.



ESEMPIO

Routing CNLS

Dato lo schema di rete della figura sopra, mostrare, per ogni tratta, l'intestazione di livello 2 e 3 di un pacchetto che parte dall'host A e giunge all'host M.

Il pacchetto in partenza dall'host A circola inizialmente su una LAN e deve raggiungere il router R1. Il suo formato sarà quindi **r1 a A M**, dove la parte chiara indica l'intestazione di livello 2 Datalink (per esempio Ethernet) con gli indirizzi MAC destinazione e mittente (r1 e a), mentre la parte grigia è il pacchetto di livello 3 Network, con l'intestazione che riporta gli indirizzi mittente e destinazione (A e M).

Giunto su R1, il router elimina la busta di livello 2, legge l'indirizzo di livello 3 destinazione, consulta la

TdI riportata in figura e inoltra sulla porta P2 il pacchetto imbustandolo nuovamente con gli indirizzi di livello 2 adeguati: **r2 r1 A M**.

Seguendo lo stesso procedimento, la sequenza dei pacchetti sulle rimanenti tratte risulta:

r3 r2 A M (tratta R2-R3, attraverso la porta P2 di R2);

r4 r3 A M (tratta R3-R4, attraverso la porta P3 di R3);

m r4 A M (sulla LAN di destinazione, attraverso la porta P2 di R4).

Si nota che le TdI riportate non contengono nessuna regola alternativa per gli host, cioè in nessuna colonna *Indirizzo (add)* delle TdI in figura compare un host più di una volta.

Se tutto ciò diminuisce le prestazioni complessive (possibilità di pacchetti duplicati, pacchetti che arrivano in ritardo rispetto all'ordine di trasmissione, ecc.), il procedimento ottiene tuttavia lo scopo di limitare i problemi dovuti a eventuali interruzioni, malfunzionamenti e congestioni rilevabili sui cammini.

Si può affermare che il routing *by network address* supporta un alto tasso di fault tolerance e, in negativo, un alto tasso di congestione di rete.

Questa tecnica è in uso presso architetture di rete non orientate alla connessione (CNLS), dato che privilegiare le alternative di cammino significa non fidarsi dell'affidabilità delle infrastrutture di rete. Per questo motivo i protocolli di L3 di reti CNLS non effettuano il controllo degli errori a questo livello, poiché esso risiederebbe all'interno degli IS, cioè nelle stazioni che possiedono le interfacce di rete inaffidabili, in genere possedute dagli enti gestori delle reti. Il controllo dell'errore viene demandato al livello 4 (Trasporto), cioè alle macchine finali (ES), cioè quelle possedute da chi ha instaurato ed è proprietario della comunicazione.

È il modo classico di routing CNLS adottato dalla rete TCP/IP.



Routing CNLS e next hop

Date le TdI per tre router R1, R2 e R3, acquisire in input gli indirizzi mittente e destinatario di un pacchetto trasmesso dall'host A (direttamente connesso al router R1), quindi mostrare in output il percorso del pacchetto.

(In Appendice la versione per Java)

OUTPUT

```
Digitare indirizzo sorgente (a,b,c): a
Digitare indirizzo destinazione (a,b,c): c
Percorso: a R1 R2 R3 c
```

Per rappresentare i collegamenti dei router R1, R2, R3 e R4 si usa la stessa tecnica del **PROGRAMMA** precedente.

Le tre TdI dei router R1, R2 e R3 possono essere rappresentate con una struttura dati che riflette le componenti standard di una TdI: Indirizzo, porta, hop e costo (il costo non sarà tuttavia analizzato).

Il programma, dà per scontato che il router di partenza sia R1 (**riga 34**).

Tra gli host previsti dal programma (**a, b, c**), si nota che l'unico direttamente connesso a R1 è **a** (infatti gli hops nella riga della TdI di R1 relativa ad **a** vale 0).

Il codice in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01 #include <conio.h>
02 #include <string.h>
03 #include <stdlib.h>
04
05 typedef struct {
06     char add;           // indirizzo
07     unsigned char p;    // porta
08     unsigned char h;    // hop
09     unsigned char c;    // costo
10 } TDI;
11 TDI tdiR1[] = {{ 'a', 1, 0, 0 }, { 'b', 2, 1, 5 }, { 'c', 2, 2, 2 } };
12 TDI tdiR2[] = {{ 'a', 3, 1, 0 }, { 'b', 7, 0, 5 }, { 'c', 2, 1, 2 } };
13 TDI tdiR3[] = {{ 'a', 9, 2, 0 }, { 'b', 8, 1, 5 }, { 'c', 9, 0, 0 } };
14 TDI* tdi;
15
16 typedef struct {
17     char ra[3];          // Il router ra è connesso...
18     char rb[3];          // ...al router rb...
19     unsigned char p;     // ...sulla porta p (di ra)
20 } LINK;
21 LINK linkrouter[] = {{ "R1", "R2", 2 }, { "R2", "R1", 4 }, { "R2", "R3", 2 }, { "R3", "R2", 5 } };
22
23 int main (void)
24 {
25     char chS, chD;
26     char szHops[100], *pc;
27     int i, j, h, r, p, n = 0;
28
29     printf("Digitare indirizzo sorgente (a,b,c): ");
30     chS = getche();
31     printf("\nDigitare indirizzo destinazione (a,b,c): ");
32     chD = getche();
33
34     h = -1; r = 1;
35     szHops[0] = 0;
36     pc = szHops;
37     n = n + sprintf(pc+n, "%c ", chS);
38     while (h)
39     {
40         p = -1; h = -1;
41         n = n + sprintf(pc+n, "R%c ", r + '0');
42
43         if (tdi) free(tdi);
44         switch (r)
45         {
```

```

46 case 1: tdi = malloc(sizeof(tdiR1)); memcpy(tdi,tdiR1,sizeof(tdiR1)); break;
47 case 2: tdi = malloc(sizeof(tdiR2)); memcpy(tdi,tdiR2,sizeof(tdiR2)); break;
48 case 3: tdi = malloc(sizeof(tdiR3)); memcpy(tdi,tdiR3,sizeof(tdiR3)); break;
49 }
50
51 for (i=0; i<sizeof(tdi); i++)
52 {
53     if (chD==tdi[i].add)
54     {
55         h = tdi[i].h; p = tdi[i].p;
56         break;
57     }
58 }
59
60 if (h>0)
61 {
62     for (j=0; j<sizeof(linkrouter)/sizeof(LINK); j++)
63     {
64         if ((linkrouter[j].ra[1]==r+'0') && (linkrouter[j].p==p))
65         {
66             r = linkrouter[j].rb[1]-'0';
67             break;
68         }
69     }
70 }
71 else
72 {
73     if (h==0) n = n + sprintf(pc+n,"%c",chD);
74     else
75     {
76         printf("\nImpossibile instradare");
77         return 0;
78     }
79 }
80 }
81
82 printf("\nPercorso: %s",szHops);
83 return 0;
84 }

```

Per ogni salto (detto anche **next hop**), l'algoritmo di inoltro è realizzato tramite:

1. Caricamento della TdI del router (**righe 44÷49**).
2. L'individuazione della riga di inoltro (*regola*) sulla TdI del router (**righe 51÷58**).
3. L'individuazione del router *next hop* (**righe 62÷69**) o controllo dell'arrivo a destinazione (basato sul controllo del valore hops della riga, dato che hops=0 significa destinazione raggiunta, **righe 73÷79**).

Per prevenire loop ed errori si usa la sentinella $h = -1$.

3 Routing CONS: label swapping

Tramite il **label swapping** il mittente e il destinatario aprono preventivamente una **connessione** decidendo il cammino attraverso gli IS.

La ricerca del cammino tra mittente e destinatario avviene attraverso gli stessi principi operanti sulle reti CNLS, sfruttando tabelle e algoritmi di instradamento attraverso gli indirizzi univoci di livello 3 assegnati alle stazioni.

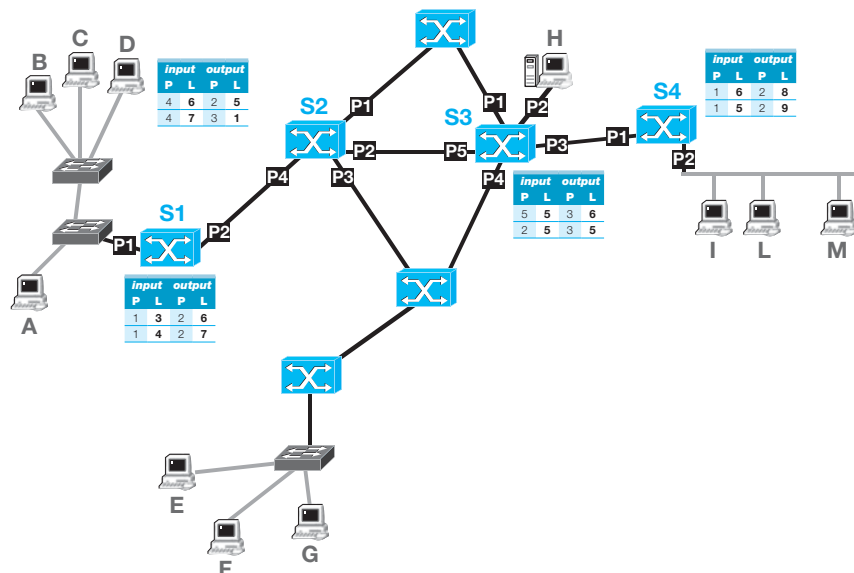
Una volta stabilito il percorso, però, i pacchetti successivi circolano sempre sullo stesso cammino e non contengono più gli indirizzi mittente/destinatario, ma solo un identificativo della connessione, in modo tale che gli

IS (che nelle reti CONS sono anche detti **switch**) li possano instradare nel cammino già calcolato, denominato **circuito virtuale**.

Per ottenere questo effetto è necessario che ogni IS sia a conoscenza del fatto che per determinati pacchetti in entrata, la porta in uscita sia sempre la stessa, quella scelta dall'algoritmo di routing nel momento della connessione. Per questo motivo i protocolli label swapping operanti negli IS agiscono su un campo speciale (**label**) dell'intestazione del pacchetto nel quale inseriscono l'identificativo di riconoscimento del circuito virtuale.

Il procedimento funziona solo se ogni IS, quando riceve i pacchetti, associa la label del pacchetto in entrata da una determinata porta a una nuova label creata localmente e associata alla porta d'uscita stabilita durante la connessione: il pacchetto in entrata subirà, al passaggio su ogni IS, la modifica del campo label (label swapping). Questa associazione viene mantenuta in una speciale tabella dell'IS detta **look-up table**.

L'operazione di label swapping è un'operazione decisamente veloce e che non richiede nessun calcolo se non un accesso indicizzato alla look-up table; addirittura l'operazione può essere eseguita in hardware. La differenza con un accesso a una TdI tradizionale è forte: la coppia (porta ingresso, label sul pacchetto) costituisce già l'indice esatto d'accesso alla look-up table per recuperare la coppia d'uscita (porta d'uscita, nuova label); inoltre la dimensione della tabella è uguale al numero di circuiti virtuali attivi in quel momento su quell'IS, cioè relativamente pochi (mentre una classica TdI deve prevedere, in linea di principio, una riga per ogni possibile destinazione, anche quelle magari non utilizzate da tempo).



Evidentemente questo sistema ha problemi se un circuito virtuale incappa in una tratta difettosa o che cade durante la sessione: il label swapping presume che le connessioni di livello 2 Datalink siano estremamente affidabili e non prevede tolleranza ai guasti di sessione.

Il label swapping è la tecnica standard per le comunicazioni CONS ed è adottata su reti come **ATM** (*Asynchronous Transfer Mode*) e **MPLS** (*Multi Protocol Label Switching*).

Routing CONS

Data la rete della figura nella pagina precedente, mostrare il circuito virtuale di un pacchetto «sessione» emesso dall'host A e diretto all'host M.

Osservando la look-up table del label-switch **S1**, si nota che **A** potrebbe aver creato il circuito virtuale di label **L=3** o di label **L=4**.

Entrambi i circuiti virtuali escono sulla porta **P2** di **S1**, ma una volta giunti a **S2**, il secondo circuito virtuale viene smistato sulla porta **P3** di **S2** (come risulta dalla look-up table di **S2**): questo non è il circuito

virtuale richiesto dal testo, dato che non raggiungerà (presumibilmente) l'host **M**.

Il primo circuito virtuale, invece, viene inoltrato da **S2** sulla porta **P2** con label **L=5** (look-up table di **S2**).

Giunto a **S3**, viene inoltrato sulla porta **P3** con label **L=6**, quindi giunto su **S4** viene inviato a destinazione (**M**) tramite la porta **P2** e con label **L=8**.

In definitiva la sequenza delle label del circuito virtuale che collega l'host **A** all'host **M** è: **3-6-5-6-8**.

Si nota che la figura descrive anche un circuito virtuale completo dall'host **H** a un host sulla rete LAN di **I**, **L**, **M**.

4 Algoritmi di instradamento

A prescindere dalla modalità di comunicazione, connessa o datagramma (un altro modo per indicare le comunicazioni disconnesse), è possibile elencare e discutere tutta una serie di algoritmi di instradamento che generalmente sono adottati da numerosi IS, sia in modo CONS sia in modo CNLS.

In definitiva si possono catalogare due classi di algoritmi:

- **non adattativi** (o statici), che usano criteri di instradamento costanti, in genere stabiliti manualmente da un responsabile di rete o basati su algoritmi generali;
- **adattativi** (o dinamici), che usano criteri di instradamento calcolati automaticamente in funzione della topologia della rete, dello stato dei collegamenti e del carico. Monitorizzano in tempo reale, o quasi, lo stato della rete, e si adattano ai diversi scenari che possono presentarsi nella topologia.

In molti casi è necessario ricordare come uno stack di rete utilizzi all'interno del proprio livello 3 Network uno o più algoritmi di instradamento, utilizzati in diversi contesti magari in base alle dislocazioni degli IS all'interno della rete oppure come supporto l'uno dell'altro.

4.1 FDR

FDR (*Fixed Directory Routing*), algoritmo non adattativo in cui ogni nodo IS ha una TdI scritta manualmente con regole *Indirizzo/Porta* costituite da indirizzo di destinazione e linea su cui trasmettere. In qualche caso è possibile associare più linee per un destinatario, da utilizzarsi in caso di guasto sulla prima, per gestire un pò di fault tolerance. Le regole delle TdI con questa caratteristica di fissità sono dette anche **route statiche**.

Il gestore della rete (*network administrator* o *sysadmin*) ha il totale controllo dei flussi di carico e può modificare le TdI allorché si sia in presenza di guasti o per ottimizzare il traffico eventualmente congestionato su qualche nodo.

4.2 Flooding

Adottando il **flooding**, algoritmo non adattativo, i pacchetti vengono ritrasmessi dagli IS sempre su tutte le linee possibili, tranne quella da cui sono arrivati. Metodo proposto in campo militare per l'ovvia considerazione che il pacchetto arriva sempre a destinazione (a prescindere dal prezzo pagato dalla rete), si tratta di un algoritmo garantito convergente.

Su configurazioni a maglia genera numerosi **loop** (circoli viziosi), cioè pacchetti che transitano più volte sugli stessi IS causando notevole congestione; inoltre non ottimizza l'uso delle linee; per questo motivo sono spesso introdotti criteri per evitare i loop e i pacchetti duplicati, come particolari campi nei pacchetti L3 di tipo **age counter** che vengono aggiornati a ogni transito sui nodi intermedi per consentire l'eliminazione di pacchetti recidivi.

Per questi motivi in realtà non viene quasi mai utilizzato come algoritmo di instradamento, ma come metodo di supporto per altri algoritmi.

4.3 Routing centralizzato

Adottando il **routing centralizzato**, algoritmo adattativo, uno speciale IS nella rete detto **RCC** (*Routing Control Center*), si incarica di:

- ricevere informazioni sullo stato della rete dai nodi host connessi a ogni altro router;
- usare tali informazioni per calcolare le TdI per ogni router della rete;
- distribuire le TdI da lui calcolate a tutti gli altri router presenti in rete.

Il metodo centralizzato ottimizza le prestazioni ma è poco robusto, dato che riunisce tutti gli aspetti negativi di una gestione centralizzata: in caso di guasti dell'RCC o di qualche linea a lui connessa, l'intera rete subisce un collasso. Inoltre accumula molto carico sull'RCC che, per questo motivo, deve essere una stazione molto potente sia in prestazioni sia in connessioni.

4.4 Routing isolato

Adottando il **routing isolato**, modo adattativo, ogni IS decide la propria TdI autonomamente e senza consultare altri router. Ogni IS determina la propria TdI esclusivamente in base al traffico che lo attraversa. Questa gestione è molto simile alla gestione del routing sui Bridge 802.1D, in questo caso effettuata tramite un algoritmo detto di **backward learning**, integrato anch'esso con uno speciale algoritmo di Spanning Tree. La gestione del routing in questo caso è eminentemente locale e quindi si adatta per un traffico di questo tipo. I pacchetti "stranieri" sono instradati con difficoltà.

Il backward learning si basa su un concetto semplice: *se da una porta P giunge un pacchetto proveniente dall'host A, allora tramite quella porta P si può raggiungere l'host A*. In questo caso la regola (address=A; porta=P) viene automaticamente aggiunta alla TdI. Anche il backward learning viene usato spesso come supporto per altri algoritmi.

4.5 Routing distribuito

In questo caso gli IS risolvono il problema dell'instradamento – e della costruzione delle relative TdI – cooperando tra loro scambiandosi informazioni di servizio periodicamente, utili per la conoscenza reciproca della topologia della rete e quindi dei cammini possibili.

Si tratta della metodologia attuata più frequentemente soprattutto su reti ampie, e integra le caratteristiche positive dei due sistemi precedenti, cioè l'ottimizzazione delle prestazioni (per esempio routing centralizzato) con la risoluzione efficiente dell'instradamento locale (per esempio routing isolato).

Ogni IS calcola la propria TdI «dialogando» con altri IS e con gli ES, utilizzando una serie di algoritmi e procedure di tipo **neighbor greetings**, anch'essi localizzabili a livello L3 Network.

Esistono due approcci tipici per il routing distribuito:

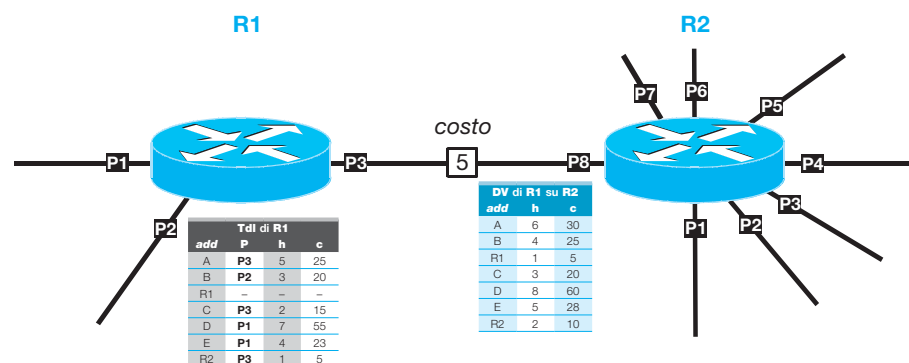
- algoritmi di tipo **distance vector**, relativamente semplici e usati per reti o sottoreti medio-piccole (fino a 1000 nodi);
- algoritmi di tipo **link state packet**, relativamente complessi e usati per reti ampie (migliaia di nodi).

5 Algoritmi distance vector

Algoritmi anche detti di Bellmann-Ford. In questo caso i router adiacenti si scambiano periodicamente dei pacchetti di servizio detti **DV** (*Distance Vector*).

Il DV è un insieme di triple di informazioni di tipo *indirizzo-hops-costo*, derivato dalla TdI e mancante del numero di linea. Le TdI sono costruite in genere con algoritmi di backward learning basati sul traffico locale, mentre l'inoltro di DV serve per aggiornare la topologia dei router dinamicamente: cammini migliori, eliminazione di nodi e relativo autoadattamento, aggiunta di nodi e relativo autoadattamento.

Quando un router modifica la propria TdI in base a qualche modifica della topologia o a qualche ricalcolo ottimizzato, invia solo ai *router adiacenti* un DV.



Ogni router memorizza per ogni sua porta l'ultimo distance vector ricevuto su quella porta. A questo punto un router ricalcola la sua TdI se:

- cade una linea attiva;
- scopre un nuovo IS;

- viene eliminato un IS;
- riceve un DV diverso da quello memorizzato.

Se due DV sulla stessa porta contengono la stessa destinazione, e spesso accade, viene scelta la regola in base al costo e, a ugual costo, in base al minor numero di hops. Pur non riportando la porta, la stessa è ricavabile dal DV semplicemente annotando quella da cui è stato ricevuto.

A questo punto quasi certamente la TdI del router ricevente sarà modificata e quindi sarà costretto a inviare un DV per ogni suo router adiacente.

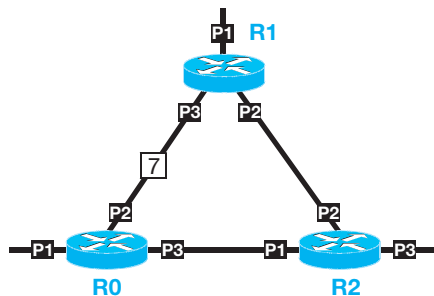
Quando cambiano una o più TdI sui router di una rete, avviene una propagazione di vari DV che ingenera normalmente altre modifiche di TdI e quindi altre propagazioni di DV; ben presto però il traffico di DV diminuirà fino a scomparire, cioè le TdI tenderanno a stabilizzarsi velocemente ottenendo una nuova configurazione di rete. In generale possono presentarsi loop, e in genere l'algoritmo converge in funzione del router più lento (o quello più trafficato o connesso con i link più lenti).

Si tratta di un algoritmo non deterministico e quindi è difficile capirne o predirne il funzionamento. Nessun router contiene la mappa complessiva della rete.

ESEMPIO

Distance vector

Nella figura sono rappresentati tre router di una rete che adottano distance vector.



Le TdI di R0 e R2 valgono:

R0	add	P	h	c	R2	add	P	h	c
A	1	5	10		A	1	6	17	
B	2	3	15		B	2	3	12	
C	3	6	28		C	3	5	21	
R1	2	0	7		R0	1	0	?	
R2	3	0	?		R1	2	0	?	

a) Determinare la TdI di R1 e completare le TdI di R0 e R2.

b) Se giunge a R0 sulla porta P1 il pacchetto di livello 3 Rete: **Z;C;3;15;T-PDU** mostrare come agisce distance vector.

a) Osservando la terza riga delle due TdI assegnate, si desume che il costo del link R0-R2 vale $28-21=7$. Infatti per raggiungere C con costo 28 il router R0 attraversa R2 e R2 a sua volta raggiunge C con costo 21.

Osservando la seconda riga delle TdI si osserva che l'host B viene raggiunto da R0 con costo 15 attraversando R1; analogamente per R2, ma con costo 12. L'host B deve essere raggiunto dalla porta P1 di R1 (altrimenti i costi e i salti non tornano: se B fosse raggiungibile da R0 attraverso R2 con 2 hops, B dovrebbe essere direttamente connesso a R2, ma così non è). Quindi il costo del link sulla porta P1 di R1 vale $15-7=8$.

Ne deriva che il costo del link R2-R1 vale $12-8=4$, desunto dalla seconda riga della TdI di R2.

Ora è possibile completare le tabelle e ipotizzare quella di R1:

R0	add	P	h	c	R2	add	P	h	c	R1	add	P	h	c
A	1	5	10		A	1	6	17		A	3	6	17	
B	2	3	15		B	2	3	12		B	1	1	8	
C	3	6	28		C	3	5	21		C	2	6	24	
R1	2	0	7		R0	1	0	7		R0	3	0	7	
R2	3	0	7		R1	2	0	4		R2	2	0	4	

b) Se nel pacchetto di livello 3 Rete **Z;C;3;15;PDU** giunto su P1 di R0 si interpreta Z come indirizz mittente, C come destinatario e 3 e 15 rispettivamente come hops e costo, allora R0 impara una nuova regola da inserire nella sua TdI: **Z,1,3,15**, inoltra il pacchetto sulla sua porta P3 (regola sulla terza riga della TdI) e avvia un aggiornamento dei DV inviando il **DV=Z,3,15** sia a R2 sia a R1.

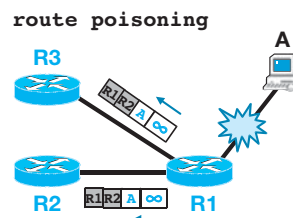
Ben presto l'algoritmo convergerà e le TdI di R1 e R2 avranno aggiunto, rispettivamente, le regole **Z,3,4,22** e **Z,1,4,22**.

5.1 Loop di routing

Gli algoritmi basati su distance vector hanno uno spiacevole effetto collaterale, i cosiddetti **loop di routing**. Si tratta di particolari condizioni per le quali le TdI non sono aggiornate in modo corretto e quindi continuano a instradare cammini non più validi. I router devono quindi adottare alcuni criteri generali per prevenire queste situazioni.

5.2 Route poisoning

Classica situazione per cui un router R1, una volta stabilito che la tratta su una sua porta è guasta (*link down*), deve segnalare la situazione ai router adiacenti. La segnalazione di R1 avviene inviando un'informazione di routing circa la linea guasta, ma con metrica infinita (**route poisoning**). Ricevuta l'informazione, R2 e R3 inibiranno l'eventuale regola di routing contenuta nelle proprie tabelle.



5.3 Split horizon

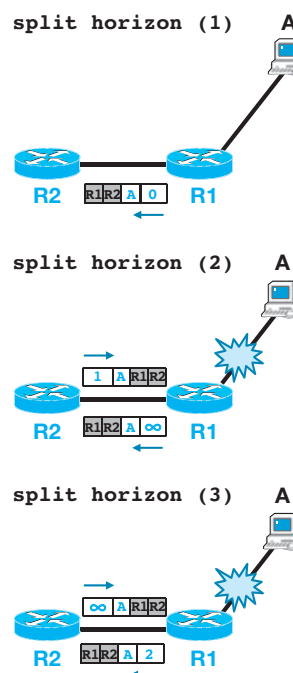
Purtroppo gli aggiornamenti collaborativi delle TdI da parte di router adiacenti hanno un inconveniente: essendo periodici, se avvengono quasi contemporaneamente possono ingenerare loop di routing.

Per esempio, se R1 e R2 sono stabili, R1 ha informato R2 che attraverso di lui si possono raggiungere determinati host. Ora R1 scopre che una sua porta è guasta e proprio quegli host non sono più raggiungibili; invierà l'informazione «poisoned» a R2 con metrica infinita; ma R2, che non ha ancora ricevuto questo messaggio, sta inviando a R1 la regola per cui attraverso di lui si possono raggiungere proprio quegli host.

Ora R2 ha ricevuto l'irraggiungibilità di quegli host, ma R1 invece li ritiene raggiungibili tramite R2, benché con un hop in più. La situazione si ripete periodicamente a ogni scambio periodico di informazioni, generando un **count to infinity** di salti nelle tabelle di R1 e R2.

Per evitare il count to infinity si adotta **split horizon**, una regola abbastanza banale: se R2 riceve una regola da R1 che riguarda una sua linea diretta (hop=0), allora R2 non ritrasmette quella regola a R1.

Una variante migliore è **split horizon** con **poisoning reverse**: se ricevuta una regola «poisoned» (indicante una linea guasta), si ritrasmette in flooding l'informazione a tutti i router adiacenti, affinché l'informazione sul guasto si diffonda più velocemente.



5.4 Path holddown

La situazione di count to infinity diventa più grave quando si estende a più di due router interconnessi (per esempio $R1 \leftrightarrow R2$, $R2 \leftrightarrow R3$, $R3 \leftrightarrow R1$). Ora lo split horizon non funziona più: R1 in presenza di guasto invia a R2

e R3 l'informazione con metrica infinita; ma contemporaneamente R2 informa R3 che proprio quella linea è ancora attiva (non ha ricevuto ancora l'informazione da R1). Lo split horizon non funziona per R3: il guasto non riguarda una sua linea diretta, quindi invia regolarmente gli aggiornamenti a R2.

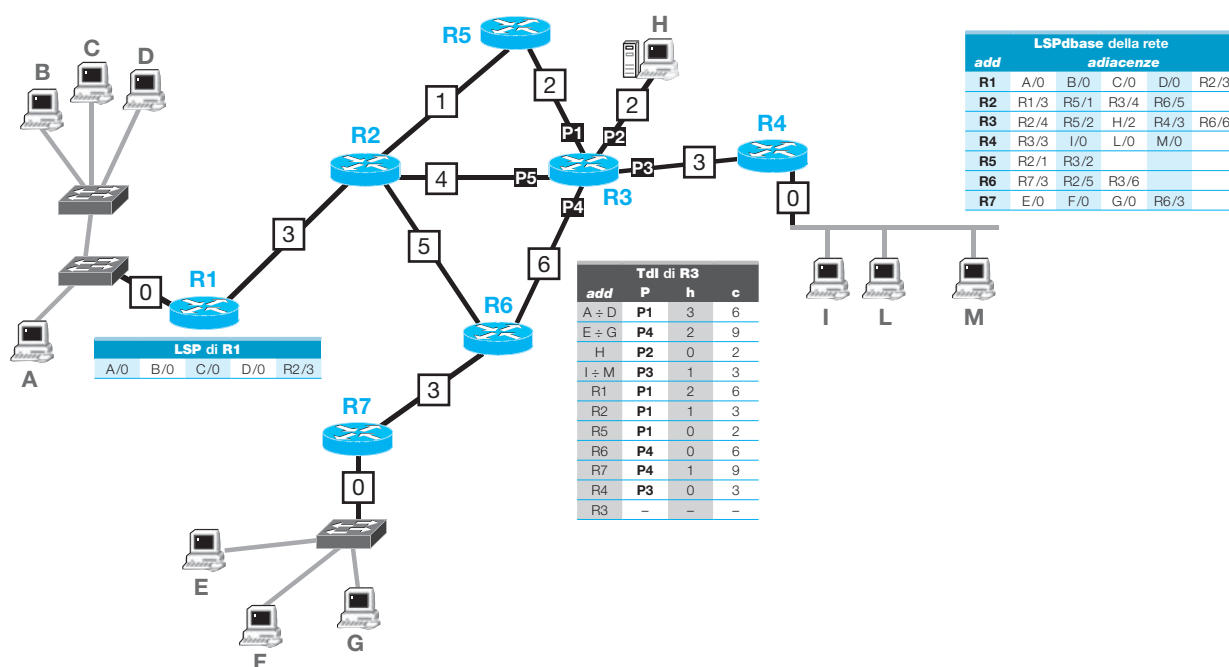
In questi casi si adotta il **path holddown**: una volta ricevuta una regola con metrica infinita (poisoned, cioè linea guasta), bisogna ignorare per un determinato intervallo di tempo altre regole ricevute successivamente circa quella linea.

6 Algoritmi link state packet

Anche in questo caso ogni router «impara» il proprio ambito locale con il backward learning (linee e nodi adiacenti), ma fa in modo che la sua conoscenza si propaghi a tutti i router sulla rete, non solo a quelli adiacenti. Trasmette in flooding queste informazioni tramite un pacchetto di servizio detto **LSP** (*Link State Packet*) in modo che tutti gli altri router, tramite i LSP ricevuti, possano ricostruirsi una mappa globale della rete. In questo modo tutti i router di una rete possiederanno le informazioni dell'intera topologia della rete, a differenza del metodo distance vector.

Quando un router modifica la sua «conoscenza» locale inizia una fase di *update process*. Pertanto genera un LSP che contiene:

- identità di ogni vicino connesso;
- costo del collegamento (*link*) relativo;
- stato del link;
- numero di sequenza;
- tempo di validità;
- age counter.



Il LSP è quindi trasmesso in flooding ottimizzato, per raggiungere tutti i router della rete.

L'insieme dei LSP che giacciono su un router – e che tende a essere identico su ogni altro router della rete – è detto **LSPdbase** (*LSP database*) e rappresenta la matrice delle adiacenze del grafo pesato (in base al costo) della rete.

Tramite il LSPdbase memorizzato su ogni router, dato un indirizzo mittente (quello del router) e un indirizzo destinazione (quello del pacchetto da inoltrare) è sempre possibile calcolare la porta d'uscita tramite l'**algoritmo di Dijkstra** o **SPF** (*Shortest Path First*). Un router quindi applica SPF ogni qualvolta non possiede la regola per un indirizzo destinazione, creando così la propria TdI.

Il metodo link state packet garantisce la convergenza, a differenza del metodo distance vector, e la raggiunge in tempi inferiori purché il router sia sufficientemente potente in termini di memoria e prestazioni di CPU.

Inoltre esso non ingenera i loop di routing, basando il calcolo dell'instradamento su una TdI realizzata in modo deterministico.

Per contro, l'uso del flooding per la diffusione dei LSP determina un considerevole traffico di servizio, con relativa perdita di banda.

Edsger Wybe Dijkstra

È il nome di uno degli studiosi più influenti della storia dell'informatica (1930-2002). Tra i suoi risultati più noti si ricorda il concetto di **semaforo** (fondamentale per la programmazione concorrente), l'algoritmo **Shortest Path** (ricerca del cammino minimo in un grafo pesato) e il biasimo assoluto verso l'istruzione **go to**.

Detti N il numero dei nodi di un grafo ed E i suoi collegamenti, l'algoritmo Shortest Path ha complessità $O(N^2)$, mentre l'algoritmo di Bellman-Ford una complessità pari a $O(N \cdot E)$.

L'algoritmo per determinare se un numero è pari o dispari ha complessità $O(1)$, mentre individuare un elemento in una lista non ordinata di N elementi ha complessità $O(N)$.

ESEMPIO

Link state packet

Data una rete come in figura, calcolare il LSP del router R2 e la TdI di R2.

Il LSP di R2 è la serie di (add/costo) di tutti gli host adiacenti, perciò:

$LSP(R2) = E/0, F/0, H/0, R1/2, R0/4.$

Per calcolare la TdI di R2 serve il LSPdbase della rete, che è la «somma» di tutti i LSP.

Si calcolano quindi gli altri due LSP, relativi a R0 e R1:

$LSP(R0) = A/0, B/0, C/0, R1/1, R2/4;$

$LSP(R1) = D/3, R0/1, R2/2.$

Il LSPdbase risulta:

R0: A/0, B/0, C/0, R1/1, R2/4;

R1: D/3, R0/1, R2/2;

R2: E/0, F/0, H/0, R1/2, R0/4.

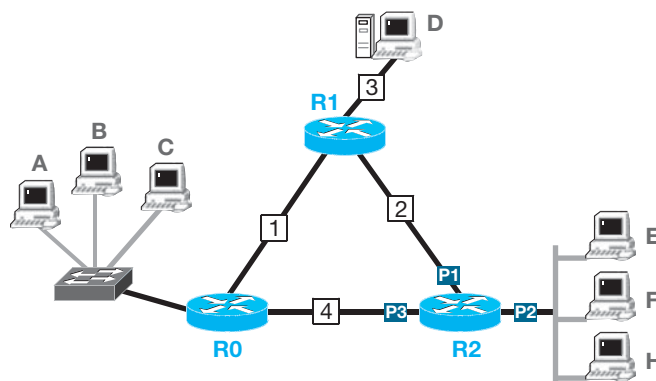
A partire dal LSPdbase, R2 calcola le porte di inoltro di ogni host e router che vi compaiono, calcolando il cammino minimo:

TdI di R2

add	P	h	c
A÷C	1	2	3
D	1	1	5
E÷H	2	0	0
R0	1	1	3
R1	1	0	2

Si nota che potrebbero essere utilizzate alcune regole alternative:

A÷C	3	0	4
D	3	1	8
R0	3	0	4
R1	3	1	5



ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

- 1** Ricordare e commentare gli elementi base dei protocolli di livello 3 Rete.
- 2** Spiegare cosa si intende per regola di routing.
- 3** Commentare il significato di metrica, hop e costo per le regole di routing.
- 4** Elencare i due approcci base del routing e commentarne le caratteristiche.
- 5** Chiarire la nozione di next hop.
- 6** Illustrare la nozione di circuito virtuale.
- 7** Descrivere i modi di routing FDR e flooding.
- 8** Elencare i principali metodi di routing illustrandone le caratteristiche base.
- 9** Introdurre il concetto di algoritmo di routing distribuito.
- 10** Spiegare la tecnica di routing Backward Learning e mostrarne qualche esempio.
- 11** Descrivere l'algoritmo Distance Vector.
- 12** Illustrare la nozione di loop di routing ed elencare le tecniche per prevenirne gli effetti.
- 13** Descrivere l'algoritmo Link State Packet.

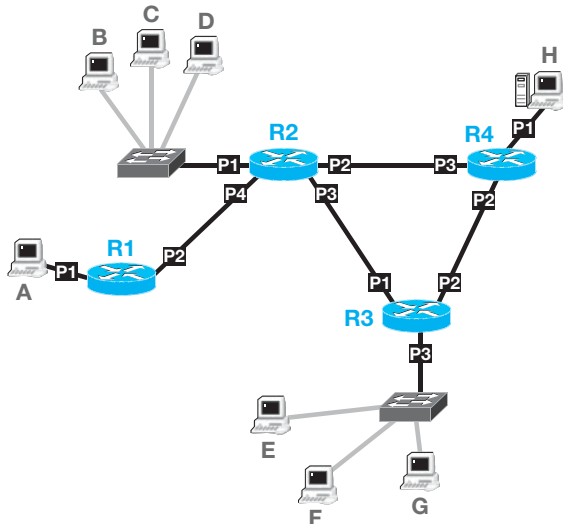
Requisiti avanzati

- 1** Mostrare su un foglio il tragitto «end to end» (da mittente a destinatario) di un pacchetto su WAN in riferimento ai livelli OSI dei nodi attraversati.
- 2** Elencare i criteri generali adottati e gli obiettivi perseguiti da un protocollo di routing.
- 3** Osservando la figura del Routing CNLS, spiegare ogni tratta attraversata da un pacchetto che va da C a H.
- 4** Spiegare i campi di una look-up table.
- 5** Associare i metodi di routing CNLS e CONS ai principali stack di rete.
- 6** Riportare la tassonomia degli algoritmi di instradamento e discuterli.
- 7** Definire la nozione di «route statica» e provare a indicarne un possibile impiego.
- 8** Ricordare quali modi di instradamento di supporto utilizzano Distance Vector e Link State Packet.
- 9** Descrivere il comportamento di Distance Vector alla modifica della topologia.
- 10** Discutere la tecnica di split horizon.
- 11** Discutere la tecnica di path holddown.
- 12** Confrontare i modi di distribuzione della struttura dati distance vector e della struttura dati link state packet.
- 13** Disegnare su un foglio lo schema di una rete magliata con qualche IS e ES, assegnare i costi ai rami e individuare i cammini migliori per qualche coppia di ES.

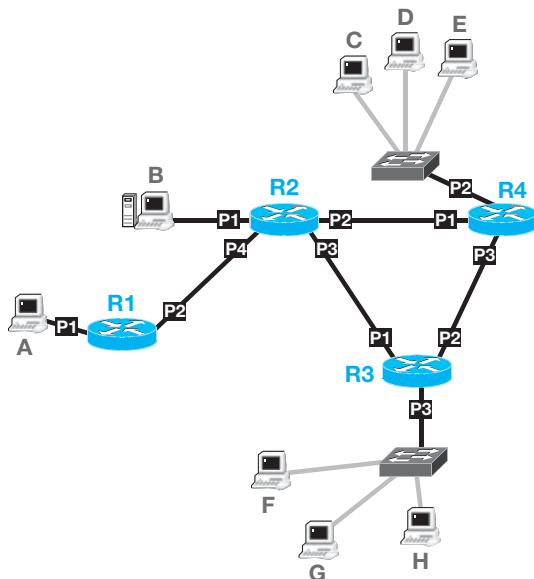
ESERCIZI PER LA VERIFICA SCRITTA

Instradamento

- 1 Dato il seguente schema di rete, determinare le TdI dei router R1, R2, R3 e R4.

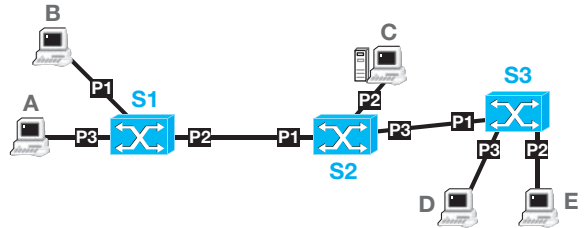


- 2 Dato lo schema di rete precedente, mostrare, per ogni tratta, l'instestazione di livello 2 e 3 di un pacchetto che parte dall'host A e giunge all'host G.
- 3 Dato il seguente schema di rete, determinare le TdI dei router R1, R2, R3 e R4.

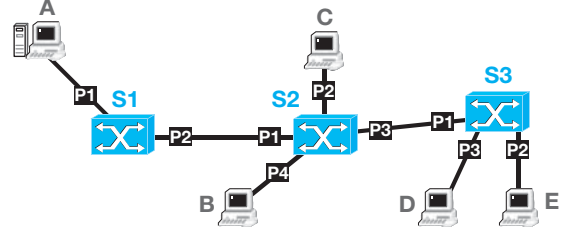


- 4 Dato lo schema di rete precedente, mostrare, per ogni tratta, l'instestazione di livello 2 e 3 di un pacchetto che parte dall'host E e giunge all'host A.

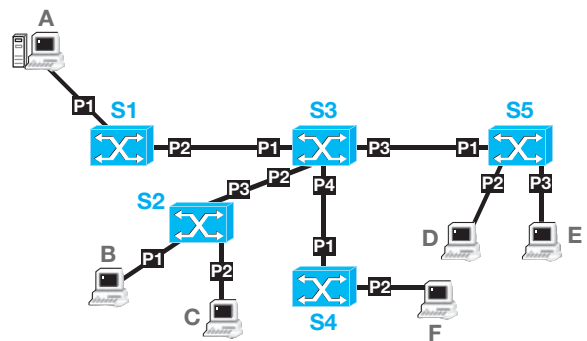
- 5 Dato il seguente schema di rete, determinare le look-up table degli switch in modo che siano definiti i seguenti circuiti virtuali: A-C, A-E, B-C, B-D, C-E.



- 6 Dato il seguente schema di rete, determinare le look-up table degli switch in modo che siano definiti i seguenti circuiti virtuali: A-D, C-E, D-E, A-B, B-A.



- 7 Dato il seguente schema di rete e le look-up table degli switch, indicare quali circuiti virtuali sono descritti.



Switch S2

input		output	
P	L	P	L
1	1	3	2
2	1	3	3
1	2	3	1

Switch S3

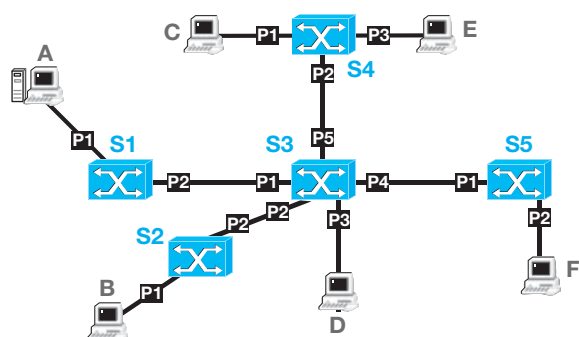
input		output	
P	L	P	L
2	3	3	4
2	1	3	3
2	2	3	2

Switch S5

input		output	
P	L	P	L
1	2	3	4
1	4	2	1
1	3	2	4

- 8 Dato lo schema di rete precedente, aggiornare le look-up table per i seguenti collegamenti: C-E, F-A, D-E, B-F, A-F.

- 9** Dato il seguente schema di rete e le look-up table degli switch, indicare quali circuiti virtuali sono descritti.



Switch S1

input		output	
P	L	P	L
1	1	2	1
2	2	1	2
2	3	1	3

Switch S3

input		output	
P	L	P	L
1	1	5	1
5	2	1	2
3	3	1	3

Switch S4

input		output	
P	L	P	L
2	1	1	1
3	2	2	2

- 10** Dato lo schema di rete precedente, aggiornare le look-up table per i seguenti collegamenti: A-D, E-D, A-F, B-D, C-E.

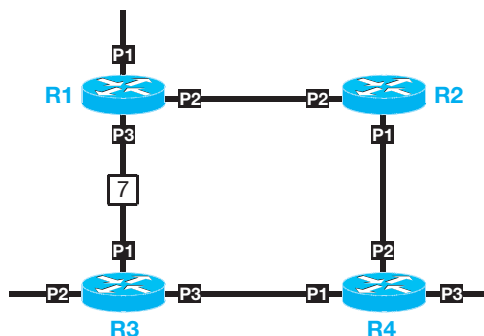
- 11** Dato lo schema di rete dell'Esercizio 1, determinare le Tdl dei router dopo aver instradato il seguente traffico in backward learning (le Tdl sono inizialmente vuote):

A → B, A → H, B → A, H → D, G → A, G → D, A → G.

- 12** Dato lo schema di rete dell'Esercizio 3, determinare le Tdl dei router dopo aver instradato il seguente traffico in backward learning (le Tdl sono inizialmente vuote):

A → B, A → H, B → A, H → D, G → A, G → D, A → G.

- 13** Dato lo schema di rete seguente e le Tdl relative, determinare i costi mancanti delle linee.



Tdl R1

Add	P	h	c
A	3	1	7
B	1	0	0
C	1	0	0

Tdl R2

Add	P	h	c
A	1	2	7
B	2	1	6
C	1	0	0

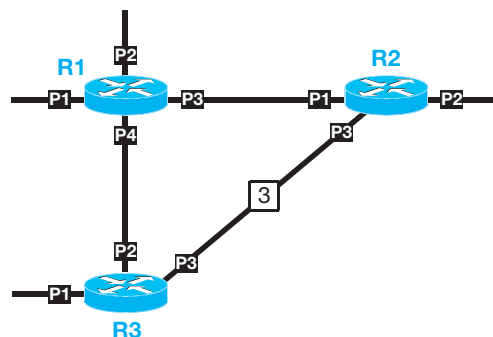
Tdl R4

Add	P	h	c
B	1	2	10
C	2	2	10

- 14** Dato lo schema di rete dell'esempio precedente, riportare le Tdl aggiornate in base al traffico (Mitt, Dest, h, c):

- su P2 di R3 giunge (X,A,0,0);
- su P3 di R4 giunge (Y,B,1,9);
- su P1 di R1 giunge (Z,X,2,3).

- 15** Dato lo schema di rete seguente e le Tdl relative, determinare i costi mancanti delle linee.



Tdl R1

Add	P	h	c
B	4	1	6
C	2	0	0

Tdl R2

Add	P	h	c
A	3	1	3
B	1	2	10

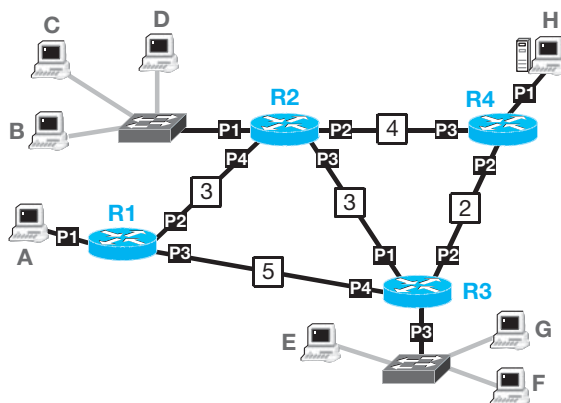
Tdl R3

Add	P	h	c
A	1	0	0
B	1	0	0


- 16** Dato lo schema di rete dell'esempio precedente, riportare le Tdl aggiornate in base al traffico (Mitt, Dest, h, c):

- su P2 di R2 giunge (X,A,0,0);
- su P2 di R1 giunge (Y,B,1,9);
- su P1 di R3 giunge (Z,C,2,3).

- 17** Dato il seguente schema di rete (le linee senza costo hanno costo 0), determinare il LSP di ogni router, le Tdl di ogni router e il LSPdbase della rete.



ESERCIZI PER LA VERIFICA DI LABORATORIO

 Per compilare gli esercizi proposti è stato usato l'ambiente gratuito multiplatforma Windows/Linux **Code::Blocks 10.5** con **gcc 4.4.1** e **Eclipse SDK 4.2.0** con **Jdk 7** per **Java 1.7**.

I testi degli esercizi presentano uno o più **layout** di input/output richiesti; in questo modo sono indicate, a volte, alcune specifiche del programma come l'output, il controllo dell'input o i valori ammissibili.

Instradamento

- 1 Scrivere un programma che implementa l'algoritmo FDR con route statiche scritte in un file di testo (**Add;P;h;c**).

Layout:

OUTPUT
FDR su router con 3 interfacce. TdI:
Add P h c
A 1 3 9
B 2 2 4
C 3 2 10
D 1 1 5
Pacchetto da instradare (Mitt, Dest): Z A
Su porta 1
Pacchetto da instradare (Mitt, Dest): Z X
Non instradabile
Pacchetto da instradare (Mitt, Dest): 0

- 2 Scrivere un programma che implementa l'algoritmo FDR con route statiche scritte in un file di testo (**Add;P;h;c**) e flooding.

Layout:

OUTPUT
FDR su router con 3 interfacce. TdI:
Add P h c
A 1 3 9
B 2 2 4
C 3 2 10
D 1 1 5
Pacchetto da instradare (P, Mitt, Dest): 2 Z A
Su porta 1
Pacchetto da instradare (P, Mitt, Dest): 1 Z X
Su porte 2, 3 (flooding)
Pacchetto da instradare (P, Mitt, Dest): 0

- 3 Scrivere un programma che implementa l'algoritmo backward learning su un router a tre porte.

Layout:

OUTPUT
TdI vuota.
Traffico 1) (P, Mitt, Dest, h, c): 1 A C 2 10

Flooding
Traffico 2) (P, Mitt, Dest, h, c): 2 B C 1 3
Flooding
Traffico 3) (P, Mitt, Dest, h, c): 2 B A 1 3
Su P1
Traffico 4) (P, Mitt, Dest, h, c): 3 C E 3 7
Flooding
Traffico 5) (P, Mitt, Dest, h, c): 1 A C 2 10
Su P3
Traffico 6) (P, Mitt, Dest, h, c): 0

- 4 Scrivere un programma che, date le look-up table di 4 switch, instradi i pacchetti forniti dall'utente nella forma (**Mitt, Label**).

Layout:

OUTPUT
Inserire Mittente e Label: A 3
Circuito virtuale: 3-6-5-6-8
Inserire Mittente e Label: H 5
Circuito virtuale: 5-5-9
Inserire Mittente e Label: 0

*Nota: ispirarsi al programma **Routing CNLS e next hop** presente nel testo. Il layout del programma agisce sullo schema di rete connessa riportata nel disegno del paragrafo **Routing CONS: label swapping**.*

- 5 Scrivere un programma che implementa l'algoritmo FDR con route statiche scritte in un file di testo (**Add;P;h;c**) e flooding ma con il formato di indirizzi IP.

Layout:

OUTPUT
FDR su router con 3 interfacce. TdI:
Add P h c
192.168.0.1 1 3 9
192.168.1.1 2 2 4
192.168.2.1 3 2 10
192.168.3.1 1 1 5
Pacchetto da instradare (P, Mitt, Dest):
2 172.16.13.1 192.168.0.1; su porta 1
Pacchetto da instradare (P, Mitt, Dest):
1 172.16.13.1 192.168.5.1; flooding
Pacchetto da instradare (P, Mitt, Dest):
0

Il protocollo **IP** (*Internet Protocol*), meglio noto come **IPv4** (*IP version 4*), è il protocollo di livello 3 Network più famoso e utilizzato al mondo.

La prima formulazione ufficiale è stata redatta nel 1981 con l’RFC 791.

Assieme a TCP (*Transmission Control Protocol*) costituisce l’ossatura della rete comunemente nota come *TCP/IP* o *Internet Protocol Suite*.

	OSI	TCP/IP
L7 Application		HTTP
L6 Presentation		SMTP NFS
L5 Session		FTP
L4 Transport		TCP UDP
L3 Network		IP routing ICMP ARP
L2 Data link		non specificati
L1 Physical		non specificati

La figura mostra l’architettura dell’Internet Protocol Suite e la paragona con il modello di riferimento ISO/OSI, riportando i protocolli più famosi operanti nei livelli della suite TCP/IP.

Siccome IP è un protocollo **disconnesso**, esso determina la natura complessivamente disconnessa dell’intera rete TCP/IP.

La parte **connessa** di TCP/IP, comunque sempre necessaria in una rete, è offerta dal protocollo di trasporto TCP, che è un protocollo connesso.

In molti casi e soprattutto quando i livelli inferiori sono ritenuti affidabili, TCP/IP può usare come livello di trasporto UDP (*User Datagram Protocol*) al posto di TCP, offrendo in questo caso un esempio di rete quasi totalmente disconnessa dato che UDP è un protocollo disconnesso.

I livelli 5 Sessione e 6 Presentazione non esistono in TCP/IP e sono demandati ai protocolli operanti al livello 7 Applicazione, pertanto non esiste uno standard univoco per gestire le problematiche di livello Sessione e di livello Presentazione.

L’architettura di rete TCP/IP non specifica i livelli 1 e 2 della rete, ma utilizza quelli normalmente disponibili sugli host e conformi agli standard.

Per esempio, nell’ambito di rete locale Ethernet, IP opera sul protocollo LLC o direttamente sulle trame Ethernet; oppure, nell’ambito delle reti geografiche, IP opera sul protocollo PPP, ma anche integrato con tecnologie

connesse come ATM e MPLS (cfr.). Esistono anche realizzazioni per reti fisiche molto strane, spesso diffuse solo all'interno di certe comunità o utilizzate in ambito industriale.

1 Protocollo ARP

Il protocollo **ARP** (*Address Resolution Protocol*, RFC 826) serve per ottenere l'indirizzo di livello 2 (per esempio un MAC address) di una stazione di cui si conosce solo l'indirizzo di livello 3 (per esempio l'indirizzo IP).

Quando un host A (mittente) su una LAN si ritrova a dover spedire una informazione a un host B (destinazione) di cui conosce *solo* l'indirizzo IP, affinché i pacchetti che contengono l'informazione possano raggiungere l'host B sulla LAN, quei pacchetti devono contenere l'indirizzo MAC destinatario dell'host B.

Per recuperare tale indirizzo, l'host A deve servirsi di ARP: invia un pacchetto **ARP Request** in broadcast sulla rete, contenente l'indirizzo IP di B e attende che B, se presente nel dominio di broadcast e regolarmente acceso, risponda con un pacchetto **ARP Reply** di tipo unicast, contenente il suo indirizzo MAC.

Solo ora l'host A può inviare l'informazione all'host B, incapsulandola in una trama Ethernet corredata dall'indirizzo MAC destinazione di B.

Il protocollo ARP deve essere implementato *su tutti gli host di una rete*, sia la parte client (*ARP Request*), sia la parte server (*ARP Reply*).

Il protocollo ARP memorizza tramite il sistema operativo una tabella che contiene le associazioni (*Indirizzo IP, MAC address*) in modo da evitare di inviare un ARP Request se l'indirizzo IP dell'host ricercato è già stato «risolto» e il suo indirizzo MAC si trova nella tabella. Questa tabella è detta **ARP cache** ed è consultabile e gestibile con il programma omonimo **arp** (cfr. *Sistemi e reti, vol. 1, Applicazioni, arp*). Le voci nell'ARP cache vengono periodicamente eliminate e reinserite, per evitare che un'informazione troppo vecchia fornisca associazioni errate.

Come si è cercato di rappresentare nella figura precedente, il protocollo ARP non è rigorosamente un protocollo di livello 3 Network (come indica la letteratura), piuttosto un protocollo di livello “2,5”, notazione che indica una diretta dipendenza dal livello 2 Data link su cui ARP si ritrova a operare.

1.1 Pacchetto ARP

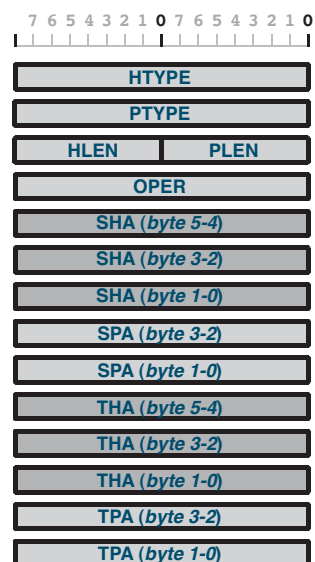
Il pacchetto ARP ha dimensione fissa 28 ottetti. Il significato dei campi è il seguente (tra parentesi la dimensione in bit):

- **HTYPE** (16, *Hardware Type*): codifica del tipo di livello fisico (Ethernet=1).
- **PTYPE** (16, *Protocol Type*): codifica del protocollo richiedente. Come Ethertype (per esempio IPv4=0800h).

Proxy ARP

Siccome ARP agisce in broadcast, la sua azione non si estende tra le subnet di una rete privata: gli host di una sottorete diversa da quella del mittente *ARP Request* non risponderanno, dato che i router bloccano il traffico MAC broadcast.

In questi casi si può attivare sui router di una rete privata il protocollo **proxy ARP** (RFC 1027): ora la richiesta ARP verrà inoltrata dal router alla sottorete opportuna, e l'host «distante» risponderà al router con il proprio indirizzo MAC. Il router con proxy ARP quindi renderà la risposta al richiedente originale.



RARP

Reverse Address Resolution Protocol, RFC 903, è il nome del protocollo «inverso» ad ARP: indicato un indirizzo di livello 2 (per esempio il MAC address) di un host, restituisce l'indirizzo di livello 3 (per esempio l'indirizzo IP) di quell'host.

Come per ARP, la *RARP Request* è inviata in broadcast, ma a differenza di ARP la parte server del protocollo (cioè chi risponde con *RARP Reply*) è realizzata solo su alcune macchine, in modo da centralizzare la distribuzione degli indirizzi IP. Il formato del pacchetto è identico, anche se l'uso dei campi è ovviamente diverso. OPER vale 3 per *RARP Request* e 4 per *RARP Reply*.

RARP ha un problema: i pacchetti broadcast vengono 'fermati' dai router, perciò il servizio di distribuzione degli indirizzi IP non è ottimizzato.

Per questo motivo RARP è reso obsoleto da protocolli di livello superiore come **BOOTP** e **DHCP** (cfr. *Sistemi e reti*, vol. 3, **DHCP**).

- **HLEN** (8, *Hardware Length*): dimensione in byte dell'indirizzo fisico richiesto.
- **PLEN** (8, *Protocol Length*): dimensione in byte dell'indirizzo «richiedente».
- **OPER** (16, *Operation*): codifica per ARP Request (1) e Reply (2).
- **SHA** (16, *Sender Hardware Address*): se *Request*, indirizzo MAC del richiedente; se *Reply*, indirizzo MAC **richiesto**.
- **SPA** (16, *Sender Protocol Address*): se *Request*, indirizzo IP del richiedente; se *Reply*, indirizzo IP richiesto.
- **THA** (16, *Target Hardware Address*): se *Request*, non usato (tutti a 0); se *Reply*, indirizzo MAC del richiedente.
- **TPA** (16, *Target Protocol Address*): se *Request*, indirizzo IP «**di cui scoprire l'equivalente fisico**»; se *Reply*, indirizzo IP del richiedente.

ESEMPIO

Frame ARP

Data la seguente sequenza esadecimale di byte che rappresenta un pacchetto ARP, decodificarne l'intestazione.

```
00 01 08 00 06 04 00 02 00 13 46 0b 22 ba c0 a8 00 01 00 16 ce  
6e 8b 24 c0 a8 00 72
```

HTYPE (byte 0-1 = 0001h): codifica 1, cioè hardware di tipo Ethernet.

PTYPE (byte 2-3 = 0800h): richiedente con codifica Ethertype, cioè IP.

HLEN (byte 4 = 06h): lunghezza indirizzo richiesto 6 byte (MAC Address).

PLEN (byte 5 = 04h): lunghezza indirizzo del richiedente 4 byte (IP address).

OPER (byte 6-7 = 00h 02h): pacchetto ARP di tipo **ARP Reply**.

SHA (byte 8-13 = 00h 13h 46h 0bh 22h bah): **MAC richiesto**.

SPA (byte 14-17 = c0h a8h 00h 01h): IP host richiesto, 192.168.0.1.

THA (byte 18-23 = 00h 16h ceh 6eh 8bh 24h): MAC del richiedente.

TPA (byte 24-27 = c0h a8h 00h 72h): IP del richiedente (192.168.0.114).

Quando un host A vuole l'indirizzo MAC di un host B di cui conosce solo l'indirizzo IP, prima di tutto consulta la sua ARP cache per verificare se esiste l'indirizzo IP di B. Infatti se esiste l'indirizzo IP di B in ARP cache, allora si può prelevare immediatamente il suo indirizzo MAC e non è necessario fare una richiesta in rete.

Se invece l'indirizzo IP di B non compare in ARP cache, allora va fatta una richiesta ARP in rete. Si prepara quindi un pacchetto ARP (*ARP Request*) con i seguenti campi impostati:

HTYPE = 1, hardware di tipo Ethernet.

PTYPE = 0800h, richiedente di tipo IP (Ethertype = 0800h).

HLEN = 6, dimensione indirizzo richiesto, i 6 byte di un indirizzo MAC.

PLEN = 4, dimensione indirizzo del richiedente, i 4 byte di un indirizzo IP.

OPER = 1, pacchetto ARP di tipo *ARP Request*.

SHA = **MAC(A)**, l'indirizzo MAC del richiedente.

SPA = **IP(A)**, l'indirizzo IP del richiedente.

THA = 00h 00h 00h 00h 00h 00h (tutti a zero, non usato in *ARP Request*).

TPA = **IP(B)**, l'indirizzo IP dell'host di cui si vuole l'indirizzo MAC.

Quindi il pacchetto viene incapsulato in un frame **Ethernet broadcast** con *Ethertype* = 0806h (la codifica di ARP) e trasmesso su Ethernet.

Il pacchetto *ARP Reply* corrispondente sarà di tipo unicast, e conterrà, nel campo SHA, il MAC dell'host B ricercato (**MAC(B)**).

Infine A, ricevendo risposta, inserirà l'associazione (**IP(B), MAC(B)**) nell'ARP cache.



Leggere un frame da file di testo

Dato un file di testo (per esempio frameeth.txt) contenente una serie di coppie di caratteri esadecimali rappresentanti altrettanti byte, scrivere un programma in linguaggio C per ricavare l'equivalente array di byte in memoria.

(In Appendice la versione per Java)

Array frame:

```
00 0d 2b 11 f1 01 00 50 fc 23 4b 9c 00 81 f0 f0 da 3a 0e 00 ff ef 16 00 00 00 00
00 00 00 c6 10 a1 31 00 50 fc 23 4b 9c 05 dc aa aa 03 00 00 00 08 00 45 00 00 3d
5e d1 00 ff fa 0e 65 21 00 11 43 51 fd 99 08 00 45 00 00 3d 5e d1 00 00 80 11 b8
16 c2 02
```

OUTPUT

Per analizzare convenientemente i pacchetti circolanti su una rete è utile rappresentare la loro sequenza di byte in esadecimale, magari in un file di testo.

Quindi è molto comodo poter disporre di quella sequenza in un vettore di byte in modo da poter accedere agli elementi con un indice.

Un esempio di programma in linguaggio C potrebbe essere:

```
00 #include <stdio.h>
01 #include <string.h>
02 #include <stdlib.h>
03
04 unsigned char* getBytesFromTextFile(char* szFileName, int* n);
05
06 int main (void)
07 {
08     unsigned char* pb;
09     int i,n;
10
11     pb = getBytesFromTextFile("frameeth.txt",&n);
12     printf("Array frame:\n");
13     for (i=0; i<n; i++)
14     {
15         printf("%02x ",pb[i]);
16     }
17     return 0;
18 }
19
20 unsigned char* getBytesFromTextFile(char* szFileName, int* n)
21 {
22     #define MAXCHARxRIGA (1000)
23     #define MAXRIGHE (1000)
24
25     FILE *fp;
26     char buffer[MAXCHARxRIGA]; char szBuffer[MAXRIGHE*MAXCHARxRIGA];
27     unsigned char* pbyte; char ch;
28     unsigned char byt=0;
29     int l=-1, blnAncora=1, sts=0, i=0, j=0;
30
31     *n = -1; szBuffer[0]=0;
32     fp = fopen (szFileName, "r");
33     while (blnAncora)
34     {
35         blnAncora=1;
36         if (feof(fp) || fgets(buffer, MAXCHARxRIGA, fp)==NULL) blnAncora=0;
```



```

37     else
38     {
39         buffer[strlen(buffer)-1] = '\0';
40         strcat(szBuffer,buffer);
41     }
42 }
43 fclose(fp); //printf("FILE:\n%s\n\bytes letti\n",szBuffer);
44
45 l = strlen(szBuffer);
46 while (i<l)
47 {
48     ch = szBuffer[i];
49     if (ch>='A' && ch<='Z') ch=ch+32;
50     switch (sts)
51     {
52         case 0:
53             if ((ch>='0' && ch<='9') || (ch>='a' && ch<='f'))
54             {
55                 if (ch >= '0' && ch <= '9') byt = 16*( ch - '0');
56                 else byt = 16*(ch - 'a' + 10);
57                 sts = 1;
58             }
59             break;
60
61         case 1:
62             if ((ch>='0' && ch<='9') || (ch>='a' && ch<='f'))
63             {
64                 if (ch >= '0' && ch <= '9') byt = byt + ch - '0';
65                 else byt = byt + ch - 'a' + 10;
66                 if (j==0) pbyte = malloc(1); else pbyte=realloc(pbyte,j+1);
67                 *(pbyte+j) = byt;
68                 j++;
69                 sts = 0;
70             }
71             break;
72     }
73     i++;
74 }
75 *n = j;
76 return pbyte;
77 }

```

La funzione **getBytesFromTextFile()** ha il compito di leggere la sequenza Ascii delle coppie di caratteri esadecimali su un file di testo e restituire il vettore che poi viene stampato nel main().

La funzione accetta il nome del file fisico su disco e restituisce il vettore sottoforma di puntatore e la sua dimensione.

La sequenza su disco può essere scritta su una o più righe, tipicamente con uno spazio che separa le coppie di caratteri esadecimali.

I caratteri esadecimali possono essere scritti indifferentemente in minuscolo o maiuscolo (**riga 49**). Il breve algoritmo di conversione (**righe 48÷73**) usa un semplice automa a stati finiti con due stati codificati sulla variabile **sts**.

2 Pacchetto IP

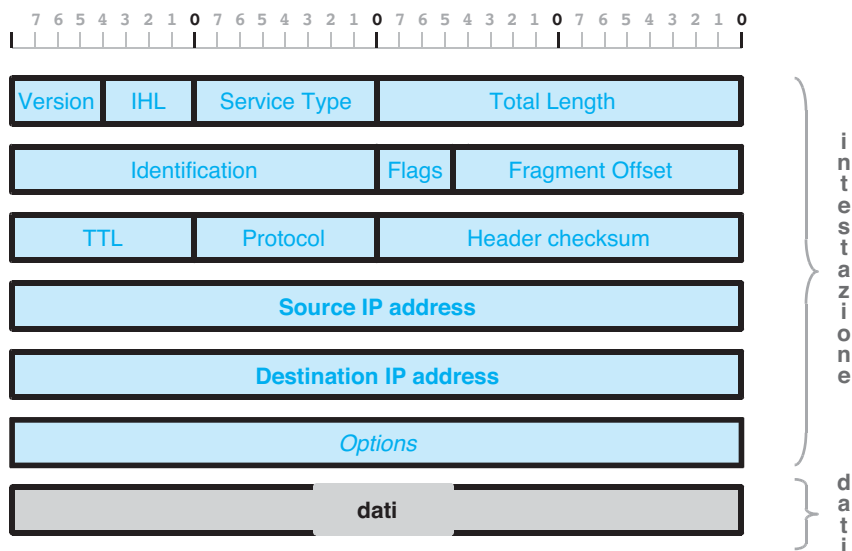
Il compito fondamentale di IP è quindi di instradare i messaggi sulla rete tramite tabelle di instradamento e algoritmi di routing, adottando un preciso schema di indirizzamento per gli indirizzi di livello 3.

Inoltre ha anche funzioni di frammentazione e riassemblaggio dei messaggi, e di rilevazione (ma *non* correzione) degli errori.

Un singolo pacchetto IP, detto anche **datagramma IP**, contiene tipicamente una PDU di livello superiore (per esempio TCP), che può essere ampia fino a 64 kB. Anche se IP è in grado di gestire pacchetti di tale di-

menzione, di fatto quasi sempre li frammenta in pacchetti più piccoli, di solito 1500 byte.

Il formato del pacchetto IP è mostrato in figura:



Sia i campi, sia i byte nei campi, sia i bit all'interno dei byte sono inseriti in modo big endian: il primo byte a essere spedito (e ricevuto) è *Version/IHL*, all'interno del quale, per esempio, il valore 45h significa *Version=4* e *IHL=5*.

L'ampiezza minima dell'header IP vale 20 byte, ma ha una lunghezza variabile a causa del campo facoltativo *Options*. In ogni caso non supera i 60 byte. L'header IP è seguito dal campo **dati** (o payload) che contiene la PDU del protocollo di livello superiore (per esempio TCP).

Il significato dei campi dell'header del pacchetto IP è il seguente (tra parentesi la dimensione in bit):

- **Version** (4): è il numero di versione del protocollo IP che ha generato il pacchetto. Per IPv4 vale 4.
- **IHL** (4, *Internet Header Length*): è la lunghezza dell'header IP, variabile in funzione del campo **Options**, espressa come numero di parole da 32 bit (4 byte). Il minimo *IHL* vale 5 ($5 \cdot 4 = 20$ byte se il campo *Options* è lungo 0 byte), il massimo vale 15 ($15 \cdot 4 = 60$ byte, se il campo *Options* è lungo 40 byte). Serve per «raggiungere» la parte *dati* e calcolarne la dimensione assieme al campo *Total Length*.
- **Service Type** (8): campo a bit che specifica come un protocollo di livello superiore vuole che il pacchetto sia trattato; dopo varie ridefinizioni oggi i trattamenti sono codificati in RFC 3168, una codifica delle quali riguarda lo streaming in tempo reale della voce VoIP (Voice over IP).
- **Total Length** (16): è la lunghezza del pacchetto IP (header + dati) espresso in byte; il pacchetto IP più ampio occupa quindi $2^{16} = 65536$ byte (64 kB), che può definirsi l'MTU di IP. Assieme a *IHL* serve per calcolare la dimensione dalla parte *dati* del datagramma IP ($= \text{Total Length} - \text{IHL} \cdot 4$).
- **Identification** (16): contiene un numero intero che identifica il pacchetto frammentato da IP; è usato per permettere il riassettaggio di

RFC 3514: il bit del male

Network Working Group
S. Bellovin

Request for Comments: 3514
AT&T Labs Research
Category: Informational
1 April 2003

The Security Flag in the IPv4 Header

Abstract

Firewalls, packet filters, intrusion detection systems, and the like often have difficulty distinguishing between packets that have malicious intent and those that are merely unusual. We define a security flag in the IPv4 header as a means of distinguishing the two cases.

1. Introduction

Firewalls, packet filters, intrusion detection systems, and the like often have difficulty distinguishing between packets that have malicious intent and those that are merely unusual. The problem is that making such determinations is hard. To solve this problem, we define a security flag, known as the “evil” bit, in the IPv4 [RFC791] header.

Benign packets have **this bit set to 0**; those that are used for an attack will have **the bit set to 1**. [...]

Osservare la data di questa RFC.

IP checksum

L'*Header checksum* di un pacchetto IPv4 comprende le prime 10 parole da 16 bit dell'header IP, escludendo la parola riservata all'*Header checksum*: in tutto 9 word (a 16 bit).

Un modo efficace di effettuare il calcolo consiste nell'eseguire la somma semplice delle 9 word, quindi se il risultato eccede i 16 bit considerare la parte troncata: a questa si somma il valore eccedente (carry).

Infine si inverte il valore ottenuto con un NOT.

Il ricevente calcola agevolmente l'integrità dell'header IP eseguendo il calcolo di checksum su *tutte* le 10 parole dell'header IP (compreso l'*Header checksum*): il risultato del calcolo dà 0 se l'header è integro; risulta diverso da 0 in caso di errore.

un pacchetto frammentato, dato che tutti i frammenti di uno stesso pacchetto originale hanno questo campo impostato con lo stesso valore.

- **Flags** (3): campo di bit che specifica se un pacchetto può essere frammentato o meno (bit1, DF, *don't fragment*) e se si tratta dell'ultimo frammento di un pacchetto o del primo e/o unico frammento (bit2, MF, *more fragment*). Siccome il destinatario non sa il numero totale dei frammenti di un pacchetto e i frammenti possono anche arrivare fuori ordine, il ricevente riconosce l'ultimo frammento solo quando analizza questo bit.

Se un datagramma IP che presenta DF=1 è più ampio dell'MTU della rete fisica su cui deve essere inoltrato (per esempio da un router), il pacchetto viene scartato e il router notifica al mittente l'errore tramite il protocollo ICMP (*Internet Control Message Protocol*).

- **Fragment Offset** (13): è il numero di sequenza di un frammento (in multipli di 8 byte). Dato che il campo occupa 13 bit, può numerare al massimo $2^{13}=8192$ pacchettini (da 0 a 8191) da 8 byte l'uno, che è la dimensione minima di un pacchetto frammentato (in questo particolare caso i pacchettini devono essere numerati al massimo fino a 8188).

Chi deve riassemblare il pacchetto originale fa scattare un timer alla ricezione del primo frammento: se il timer scade prima di aver ricevuto tutti i frammenti va segnalato al mittente con un messaggio ICMP.

Si deve tenere presente che se il frammento di un pacchetto originale si presenta su un router che deve inoltrarlo su una rete il cui MTU è inferiore all'ampiezza della dimensione del frammento, il frammento viene a sua volta frammentato, e così via.

- **TTL** (4, *Time to live*): è un campo che viene decrementato a ogni passaggio (hop) del pacchetto su un router; quando il contatore arriva a zero il pacchetto viene scartato e il router notifica lo scarto al mittente con il protocollo ICMP. È uno dei pochi campi dell'header che viene modificato durante il cammino e ciò causa anche il ricalcolo del campo *Header Checksum*.
- **Protocol** (8): codifica che identifica il protocollo di routing o di livello superiore contenuto nel campo *dati* del pacchetto. La codifica valida a livello mondiale è gestita dallo IANA (*Internet Assigned Numbers Authority*).
- **Header checksum** (16): è un campo utilizzato per controllare che l'header IP sia corretto. Si nota che la somma di controllo riguarda solo l'intestazione del pacchetto, dato che IP demanda il controllo del pacchetto allo strato superiore (tipicamente TCP). Il calcolo è effettuato con il modo *Internet checksum* (cfr. **Sistemi e reti, vol. 1, Controllo dell'errore**). È uno dei pochi campi dell'header che può essere modificato durante il cammino, per esempio a causa di un decremento di TTL.
- **Source address** (32) e **Destination address** (32): sono gli **indirizzi IP** di mittente e destinatario, entrambi su 4 byte (32 bit);
- **Options**: è un campo facoltativo (presente solo se *HIL*>5), usato per fornire varie informazioni, tipo sicurezza e routing.

La lista dei protocolli più importanti trasportati da un datagramma IP nel campo *Protocol*:

Valore	Sigla	Protocollo
01h	ICMP	Internet Control Message Protocol, RFC 792
02h	IGMP	Internet Group Management Protocol, RFC 1112
06h	TCP	Transmission Control Protocol, RFC 793
08h	EGP	Exterior Gateway Protocol, RFC 888
09h	IGP	Interior Gateway Protocol (variabile in base a quello adottato)
11h	UDP	User Datagram Protocol, RFC 768
58h	EIGRP	Enhanced Interior Gateway Routing Protocol (Cisco)
59h	OSPF	Open Shortest Path First, RFC 1583
7Ch	IS-IS	Intermediate System To Intermediate System, RFC 1142
89h	MPLS	Multiprotocol Label Switching, RFC 4023

ESEMPIO

Header IP

Data la seguente sequenza esadecimale di byte che rappresenta un pacchetto IP, decodificarne l'intestazione.

45 00 00 28 0f 47 40 00 80 06 91 ed 91 fe a0 ed 41 d0 e4 df 0d 2c 00 50 38 af ff f3 11 4c 6c 54 50 10 25 bc 6c bd 00 00

Versione (byte 0 = 45h): 4, ipv4.

HIL(byte 0 = 45h): 5, no campo Options, lunghezza header=5 · 4=20 byte.

Total length (byte 2-3 = 0028h): lunghezza del pacchetto = 40 byte (28h).

Identification (byte 4-5 = 0f47h): 0f47h.

Flags (byte 6 = 40h): 40h = **01000000b**, cioè Flags=02h, DF=1.

Il pacchetto non deve essere frammentato.

Fragment offset (byte 6-7 = 4000h): 4000h = **0100000000000000b**, cioè Offset = 0.

TTL (byte 8 = 80h): conteggio impostato a 128 (80h).

Protocol (byte 9 = 06h): in base alla codifica IANA, 06h = TCP.

Il datagramma IP trasporta un pacchetto TCP.

Header checksum (byte 10-11 = 91edh): l'IP Internet checksum vale 37357 (91edh).

Source IP address (byte 12-15 = 91fea0edh): 145.254.160.237 (91h.feh.a0h.edh).

Destination IP address (byte 16-19 = 41d0e4dfh): 65.208.228.223 (41h.d0h.e4h.dfh).

Dati (byte 20-39): PDU di TCP.



PROGRAMMA

Header IP

Data una sequenza di byte rappresentante un pacchetto IP, decodificare il contenuto del suo header con un programma in linguaggio C.

OUTPUT

```
Ip frame:
45 00 00 28 0f 47 40 00 80 06 91 ed 91 fe a0 ed 41 d0 e4 df 0d 2c 00 50 38 af ff
f3 11 4c 6c 54 50 10 25 bc 6c bd 00 00
Version: 4
HIL: 5
Total Length: 40
Identification: 3911
Flags: 2; MF=0; DF=1
Fragment offset: 0
TTL: 128
Protocol: 06h
Header checksum: 37357
Source Ip: 145.254.160.237
Destination Ip: 65.208.228.223
```

La sequenza di byte del pacchetto IP è memorizzata in un file di testo (ip.txt), pertanto viene usata la funzione `getBytesFromFile()` implementata nel **PROGRAMMA** precedente (e qui inclusa).

Un esempio di programma in linguaggio C potrebbe essere:

```
00 #include <stdio.h>
01 #include <string.h>
02 #include <stdlib.h>
03 #include "getBytesFromFile.c"
04
05 int main (void)
06 {
07     unsigned char* pb;
08     int i,n;
09     unsigned char byte;
10     unsigned int word;
11
12     pb = getBytesFromFile("ip.txt",&n);
13     printf("Ip frame:\n");
14     for (i=0; i<n; i++)
15     {
16         printf("%02x ",pb[i]);
17     }
18
19     byte = pb[0] >> 4;
20     printf("\nVersion: %d",byte);
21
22     byte = pb[0] & 0x0f;
23     printf("\nHIL: %d",byte);
24
25     word = pb[2]*256 + pb[3];
26     printf("\nTotal Length: %d",word);
27
28     word = pb[4]*256 + pb[5];
29     printf("\nIdentification: %d",word);
30
31     byte = pb[6] >> 5;
32     printf("\nFlags: %d; MF=%d; DF=%d",byte, byte>>2, byte>>1);
33
34     word = pb[6]*256 + pb[7];
35     printf("\nFragment offset: %d",word & 0x1FFF);
36
37     byte = pb[8];
38     printf("\nTTL: %d",byte);
39
40     byte = pb[9];
41     printf("\nProtocol: %02x",byte);
42
43     word = pb[10]*256 + pb[11];
44     printf("\nHeader checksum: %d",word);
45
46     printf("\nSource Ip: %d.%d.%d.%d",pb[12],pb[13],pb[14],pb[15]);
47     printf("\nDestination Ip: %d.%d.%d.%d",pb[16],pb[17],pb[18],pb[19]);
48
49     return 0;
50 }
```

Frammentazione IP

Un router riceve un datagramma IP ampio 5000 byte (*Total Length* = 5000) e lo deve inoltrare su una rete Ethernet con MTU pari a 1500 byte. Elencare i frammenti creati con i valori dei campi significativi per la frammentazione.

Si ipotizza IHL=5 (pacchetto senza Options, cioè con header ampio 20 byte).

La dimensione massima del pacchetto IP circolante sulla rete Ethernet vale 1500, di cui $1500 - 20 = 1480$ utilizzabili per il campo *dati* (payload).

Siccome 1480 è un multiplo di 8, la dimensione del payload del frammento è accettabile: $1480/8 = 185$ (altrimenti si sarebbe dovuto scegliere un'ampiezza inferiore multipla di 8).

I 5000 byte originali devono essere ripartiti in frammenti ampi 1480 byte, per un totale di $5000/1480=3,37$, quindi **3** frammenti da 1480 byte più il frammento finale ampio $5000 \% 1480 = 560$ byte (il resto della divisione precedente).

Il router quindi assegna, su base locale, il valore **23** al campo *Identification*, un valore ancora non utilizzato per i propri pacchetti.

In definitiva i quattro frammenti, con i campi significativi, risultano:

Frammento	Total Length	Identification	DF	MF	Fragment Offset
0	20 + 1480	23	0	1	0
1	20 + 1480	23	0	1	185
2	20 + 1480	23	0	1	370
3	20 + 560	23	0	0	555

3 Protocollo ICMP

Il protocollo **ICMP** (*Internet Control Message Protocol*, RFC 792) è dichiarato di livello 3 Network di TCP/IP, anche se è incapsulato all'interno di datagrammi IP, a differenza di ARP ma come TCP, con un valore di *IP Protocol* = 1 (vedi tabella precedente).

IP usa ICMP in varie occasioni, spesso su iniziativa dei router, che segnalano agli host alcune situazioni anomale o condizioni d'errore durante l'espletamento delle proprie funzioni. Infatti IP è un protocollo disconnesso, ovvero senza controllo dell'errore e senza consegna garantita dei pacchetti (protocollo di tipo *best effort*, ovvero del tipo "faccio il meglio che posso").

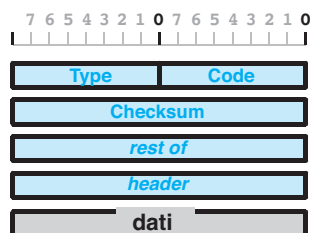
ICMP aiuta IP ad affrontare le situazioni d'errore (che non è progettato per risolvere) attraverso delle segnalazioni, tra cui si ricordano quelle direttamente utilizzate per indicare le condizioni anomale già analizzate descrivendo il protocollo IP:

- Quando un pacchetto IP non può essere frammentato ma deve «passare» attraverso una rete con MTU inferiore alla sua ampiezza, il pacchetto viene scartato e il router invia al mittente un messaggio ICMP *Destination Unreachable-Fragmentation Needed And DF Set*.
- Quando il timer per riassemblaggio scade senza aver ricevuto tutti i frammenti, il pacchetto originale viene scartato e il router invia al mittente un messaggio ICMP *Time Exceeded-Fragment reassembly time exceeded*.

- Quando un router, abbassando il TTL di un pacchetto IP in transito, arriva a zero lo scarta, ma notifica al mittente un messaggio ICMP *Time Exceeded-Time To Live Exceeded in Transit*.

Inoltre ICMP segnala problemi abbastanza tipici, come per esempio quando un router non riesce a individuare un cammino per una rete o un host di destinazione: il mittente riceve un messaggio ICMP *Destination Unreachable-Destination network unreachable* o *Destination host unreachable*. I messaggi ICMP spesso non vengono esibiti dai programmi applicativi che usano IP, come per esempio i browser per la navigazione web: le applicazioni, infatti, mascherano questi messaggi d'errore con proprie versioni del messaggio o ponendo rimedio alla segnalazione con soluzioni alternative.

Le segnalazioni ICMP avvengono tramite l'invio di un pacchetto abbastanza semplice (tra parentesi la dimensione in bit):



- **Type** (8): codifica ICMP del tipo di messaggio da consegnare.
- **Code** (8): sottotipo per il tipo di messaggio; precisa l'indicazione.
- **Checksum** (16): somma di controllo del pacchetto.
- **rest of header** (32): parte dell'header disponibile per altre informazioni.
- **dati**: payload di ICMP; tipicamente contiene tutto l'header IP del pacchetto che è coinvolto nella segnalazione, più otto byte del campo dati di quel pacchetto. Il campo è di ampiezza variabile perché l'intestazione IP è variabile.

Le codifiche più importanti per *Type/Code* sono riportate in tabella:

Type	Descrizione	Code	Messaggio
0	Echo Reply	0	Echo reply (usato dal programma ping)
3	Destination Unreachable	0	Destination network unreachable
		1	Destination host unreachable
		2	Destination protocol unreachable
		3	Destination port unreachable
		4	Fragmentation required, and DF flag set
		5	Source route failed
		6	Destination network unknown
		7	Destination host unknown
		8	Source host isolated
		9	Network administratively prohibited
		10	Host administratively prohibited
		11	Network unreachable for TOS
		12	Host unreachable for TOS
		13	Communication administratively prohibited
		14	Host Precedence Violation
		15	Precedence cutoff in effect
4	Source Quench	0	Source quench (congestion control)
5	Redirect Message	0	Redirect Datagram for the Network
		1	Redirect Datagram for the Host
		2	Redirect Datagram for the TOS & network
		3	Redirect Datagram for the TOS & host
6	Alternate Host Address	0	
8	Echo Request	0	Echo request (usato dal programma ping)
9	Router Advertisement	0	Router Advertisement

10	Router Solicitation	0	Router discovery/selection/solicitation
11	Time Exceeded	0	TTL expired in transit
		1	Fragment reassembly time exceeded
12	Parameter Problem: Bad IP header	0	Pointer indicates the error
		1	Missing a required option
		2	Bad length
13	Timestamp	0	Timestamp
14	Timestamp Reply	0	Timestamp reply
15	Information Request	0	Information Request
16	Information Reply	0	Information Reply
17	Address Mask Request	0	Address Mask Request
18	Address Mask Reply	0	Address Mask Reply
30	Traceroute	0	Information Request

ESEMPIO

Datagramma ICMP

Data la sequenza di byte esadecimali seguente, decodificare il datagramma IP contenuto:

```
45 00 00 3c 8e 9b 00 00 80 01 2a 62 c0 a8
00 72 c0 a8 00 01 08 00 49 5c 03 00 01 00
61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e
6f 70 71 72 73 74 75 76 77 61 62 63 64 65
66 67 68 69
```

Il primo byte (byte 0 = 45h) indica la coppia *Version/HIL* dell'header di IP, cioè 45h significa 4=IPv4 e

HL=5, cioè nessuna *Options* e quindi header IP lungo $5 \cdot 4 = 20$ byte.

Si nota che il campo *Protocol* vale 1 (byte 9 = 1), che la codifica IANA riserva a **ICMP**.

Il 21esimo byte (byte 20 = 8) è perciò la parte dati del pacchetto IP, ovvero il primo byte del pacchetto ICMP.

Esso coincide con il campo *Type*=8, cioè si tratta di un ICMP *Echo Request* con *Code* = 0.

Si nota che il mittente IP ha indirizzo 192.168.0.114 e il destinatario IP ha indirizzo 192.168.0.1, rispettivamente i byte 12÷15 e 16÷19.

3.1 ICMP routing

Oltre a segnalare condizioni d'errore ICMP consente di realizzare un sottile strato di routing per gli host che, normalmente, realizzano il livello 3 Network in modo abbastanza elementare rispetto allo strato di Rete implementato sui router.

Per esempio, un router può ricevere un pacchetto IP da un host e, consultando la propria TdI, può rendersi conto che l'host non ha deciso correttamente l'instradamento: un altro router è più indicato per ricevere quel pacchetto.

In questo caso il router notifica all'host «distratto» il nuovo router che deve considerare come ottimale con un messaggio ICMP *Redirect Message* contenente l'indirizzo IP del router su cui effettuare la redirectione nel campo *rest of header*. In questo modo la rudimentale TdI dell'host può aggiornarsi con una nuova regola temporanea.

Inoltre, tra gli ultimi messaggi a essere stati introdotti nel protocollo ICMP, si trovano *Address Mask Request* e *Address Mask Reply*, per permettere a un host di richiedere e quindi impostare automaticamente uno dei parametri base per il routing degli host, ovvero la *subnet mask* usata nella rete di appartenenza (cfr. [Subnetting](#)).



Pacchetto ICMP

Data una sequenza di byte rappresentante un pacchetto IP, scrivere un programma in linguaggio C che verifica se si tratta di un pacchetto ICMP.

(In Appendice la versione per Java)

OUTPUT

```
Ip frame:
45 00 00 3c 8e 9b 00 00 80 01 2a 62 c0 a8 00 72 c0 a8 00 01 08 00 49 5c 03 00 01 00 61
62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68
HIL: 5, perciò i dati IP iniziano dall'indice: 20
IP Protocol: 01h
    Pacchetto ICMP (01)
    Type: 08h Code: 00h (Echo Request)
```

Per verificare se il datagramma IP trasporta ICMP bisogna verificare se il campo *Protocol* dell'header IP contiene il valore **1** (vedi Tabella pag. 111 per il campo *Protocol* di IP).

Quindi bisogna «saltare» l'header IP calcolandone la dimensione ($HIL*4$) e decodificare i primi due byte del campo dati di IP, che coincidono con la coppia *Type/Code* di ICMP.

Un esempio di programma in linguaggio C potrebbe essere:

```
00 #include <stdio.h>
01 #include <string.h>
02 #include <stdlib.h>
03 #include "getBytesFromTextFile.c"
04
05 #define IPPROTOCOL_ICMP (0x01)
06 #define ICMP_TYPE_ECHOREQUEST (8)
07 #define ICMP_CODE_ECHOREQUEST (0)
08
09 int main (void)
10 {
11     unsigned char* pb;
12     int i,n;
13     unsigned char byte;
14     unsigned int idxIPData;
15
16     pb = getBytesFromTextFile("icmp.txt",&n);
17     printf("Ip frame:\n");
18     for (i=0; i<n; i++)
19     {
20         printf("%02x ",pb[i]);
21     }
22
23     byte = pb[0] & 0x0f;
24     idxIPData = byte*4;
25     printf("\nHIL: %d, perciò i dati IP iniziano dall'indice: %d",byte, idxIPData);
26
27     byte = pb[9];
28     printf("\nIP Protocol: %02xh",byte);
29     if (byte==IPPROTOCOL_ICMP)
30     {
31         unsigned char Type, Code;
32
33         printf("\n\tPacchetto ICMP (%02x)",byte);
34         Type = pb[idxIPData]; Code = pb[idxIPData+1];
35         printf("\n\tType: %02xh Code: %02xh",Type,Code);
36         if (Type == ICMP_TYPE_ECHOREQUEST && Code==ICMP_CODE_ECHOREQUEST)
37         {
38             printf(" (Echo Request)");
39         }
40     }
41     else printf("\n\tNon è un pkt ICMP (%02xh)",byte);
42
43     return 0;
44 }
```

4 Programma ping

Uno degli usi più proficui del protocollo ICMP viene fatto dal programma universalmente noto come **ping**, disponibile su piattaforme Windows, Linux e anche su sistemi operativi dedicati come IOS di Cisco Inc. (cfr. *Sistemi e reti, vol. 1, Applicazioni, ping*).

Fornendo l'indirizzo IP di un host al programma *ping*, esso opera spedendo messaggi ICMP *Echo Request* al destinatario e aspettando i relativi messaggi di risposta ICMP *Echo Reply* che ogni host su una rete è costretto a inviare (come prescritto dalla RFC 1122).

Durante il processo, *ping* misura l'**RTT** (*Round Trip Time*) dei pacchetti ICMP e registra eventuali pacchetti persi. Al termine mostra l'output dei risultati fornendo un'indicazione relativamente veloce sullo stato di raggiungibilità dell'host. Per la sua semplicità ed estrema utilità il programma *ping* è una utility amministrativa di rete indispensabile per verificare se un determinato host è acceso e/o raggiungibile.

RTT

L'**RTT**, detto anche **RTD**, è il tempo impiegato da un pacchetto su una rete per viaggiare da un mittente a un destinatario e tornare al mittente.

Il termine si usa in svariati contesti, come per esempio per il programma *ping*.

Il nome «ping» è stato deciso dall'autore del programma Mike Muss (1958-2000) nel 1983, in analogia con il suono emesso dai sonar. In seguito **ping** fu definito come acronimo per *Packet InterNet Groper*.

ESEMPIO

Ping e MTU

Determinare l'MTU della rete Ethernet usando il comando ping, sapendo che su sistemi operativi di tipo Windows ping genera datagrammi IP di dimensioni fisse pari a 32 byte, mentre per sistemi operativi di tipo Linux pari a 64 byte.

Affrontiamo il problema su un sistema operativo di tipo Windows (per Linux la soluzione è identica, benché con una variante nell'uso di *ping*).

Siccome un pacchetto IP generato da *ping* per Windows (IP incapsula ICMP in datagrammi IP) è piccolo (32 byte) e sicuramente inferiore all'MTU di Ethernet (infatti i messaggi *ping* normalmente non generano errori), si impone a *ping* di generare un pacchetto di dimensione desiderata, per esempio 2000 byte.

Consultando l'help del comando (**ping /?**), l'opzione da specificare è **-l 2000**.

Inoltre si può imporre a *ping* di impostare i suoi pacchetti IP come non frammentabili, cioè con **DF=1**. L'opzione da specificare, in questo caso, è **-f**.

In questo modo un pacchetto che eccede l'MTU della rete e non è frammentabile genera un errore:

```
C:\Windows\system32>ping 192.168.0.1 -l 2000 -f

Esecuzione di Ping 192.168.0.1 con 2000 byte di dati:
È necessario frammentare il pacchetto ma DF è attivo.
È necessario frammentare il pacchetto ma DF è attivo.
È necessario frammentare il pacchetto ma DF è attivo.
È necessario frammentare il pacchetto ma DF è attivo.

Statistiche Ping per 192.168.0.1:
    Pacchetti: Trasmessi = 4, Ricevuti = 0,
    Persi = 4 (100% persi),

C:\Windows\system32>
```

Provando varie dimensioni inferiori con il comando proposto, si giunge a determinare che la prima dimensione accettata (non dà errori usando *ping*) vale **1472**:

```
C:\Windows\system32>ping 192.168.0.1 -l 1472 -f

Esecuzione di Ping 192.168.0.1 con 1472 byte di dati:
Risposta da 192.168.0.1: byte=1472 durata=1ms TTL=128
Risposta da 192.168.0.1: byte=1472 durata=1ms TTL=128
Risposta da 192.168.0.1: byte=1472 durata=1ms TTL=128
Risposta da 192.168.0.1: byte=1472 durata=1ms TTL=128

Statistiche Ping per 192.168.0.1:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 1ms, Massimo = 1ms, Medio = 1ms

C:\Windows\system32>
```

In effetti si deve considerare che *ping* interpreta il parametro **-l** come dimensione del pacchetto IP esclusa l'intestazione IP (20 byte) e il pacchetto ICMP incapsulato (8 byte).

Quindi l'MTU di Ethernet vale $1472 + 20 + 8 = \mathbf{1500}$ byte.

Il comando *ping* corrispondente per Linux è: **ping -M do -s 2000 -c 4 192.168.0.1**.

```
linux:~# ping -M do -s 2000 -c 4 192.168.0.1

PING 192.168.0.1 (192.168.0.1) 2000(2028) bytes of data.
From 192.168.0.34 icmp_seq=1 Frag needed and DF set (mtu = 1500)
From 192.168.0.34 icmp_seq=1 Frag needed and DF set (mtu = 1500)
From 192.168.0.34 icmp_seq=1 Frag needed and DF set (mtu = 1500)
From 192.168.0.34 icmp_seq=1 Frag needed and DF set (mtu = 1500)
--- 192.168.0.1 ping statistics ---
0 packets transmitted, 0 received, +4 errors

linux:~# ping -M do -s 1472 -c 4 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 1472(1500) bytes of data.
1480 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=1.56 ms
1480 bytes from 192.168.0.1: icmp_req=2 ttl=64 time=1.43 ms
1480 bytes from 192.168.0.1: icmp_req=3 ttl=64 time=1.38 ms
1480 bytes from 192.168.0.1: icmp_req=4 ttl=64 time=1.40 ms

--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.388/1.446/1.565/0.075 ms
linux:~#
```

In Appendice si può trovare una lista di comandi *ping* di utilità generale.

4.1 Programma traceroute

Una seconda proficua utilizzazione del protocollo ICMP riguarda il programma **traceroute** (*tracert* per i sistemi Windows, cfr. [Sistemi e reti, vol. 1, Applicazioni, traceroute](#)).

Come nel caso di *ping*, *traceroute*/*tracert* rappresenta una utility amministrativa molto diffusa, offrendo risposta quasi immediata alla richiesta dell'elenco dei router attraversati da un pacchetto per raggiungere una destinazione data.

Il principio di funzionamento di *traceroute* è di una semplicità imbarazzante: se si invia un pacchetto IP qualsiasi a un host remoto (quello di cui si vuol tracciare la strada) con il campo *TTL*=1, il primo router sulla strada sarà costretto a rispondere al mittente con un pacchetto d'errore ICMP *Time Exceeded*.

Avrà, infatti, abbassato il *TTL* del pacchetto e, resosi conto di aver raggiunto 0, dovrà scartarlo e notificare lo scarto al mittente con il messaggio ICMP *Time Exceeded*.

Il messaggio d'errore giungerà al mittente che ora può annotarsi l'indirizzo IP del primo router attraversato.

Ora basta proseguire il procedimento aumentando il *TTL* di una unità a ogni tentativo, fino alla ricezione di un pacchetto ICMP *Echo Reply* (se come pacchetto di prova si era usato un ICMP *Echo Request*). Alla ricezione dell'ICMP *Echo Reply*, *traceroute* sa che la lista dei router attraversati è completata e può stampare il resoconto.