ABPS: TED fragmentation fixing

Matteo Martelli

21 settembre 2015

1 Nelle puntate precedenti

In questo documento si fa riferimento alla modifiche apportate al componente software TED (Transmission Error Detector) riguardanti la frammentazione. Per maggiori informazioni su TED e sull'architettura Always Best Packet Switching (ABPS) si rimanda alla documentazione precedente.

Nello specifico TED offre, tramite una struttura cross-layer, un meccanismo in grado di fornire informazioni alle applicazioni riguardo l'avvenuta (o mancata) consegna al primo access point dei datagram UDP trasmessi.

Quando l'applicazione invia un datagram UDP, TED lo marca con un identificativo user-space accessibile dall'applicazione. Successivamente ogni frammento datalink spedito dall'interfaccia di rete viene associato a quell'identificativo e ad una struttura dati contenenti informazioni sullo stato del frammento. Quest'ultime vengono poi integrate con le informazioni riguardanti l'avvenuta o mancata consegna (ACK, NACK) del frammento all'access point e il relativo numero di tentativi di consegna.

Nelle versioni precedenti a questa mancava il corretto supporto per la gestione della frammentazione in IPv6. Vedremo nella prossima sezione quali modifiche sono state apportate al kernel per il corretto funzionamento con la frammentazione in IPv6. Come nella versione precedente lo sviluppo è stato continuato per il kernel 4.0.1.

Infine nella sezione 3 vedremo le modifiche apportate all'applicazione per la gestione delle notifiche di TED dei datagram frammentati oltre a qualche esempio d'utilizzo.

2 Modifiche Kernel

Prima di mostrare le modifiche effettuate nel kernel ricordiamo brevemente come sono formati gli header dei pacchetti IPv6.

2.1 Header IPv6

In particolare siamo interessati all'header del pacchetto IP che nella versione 6 è composta da un parte fissa (fixed) e una parte variabile formata da header opzionali chiamati extension header.

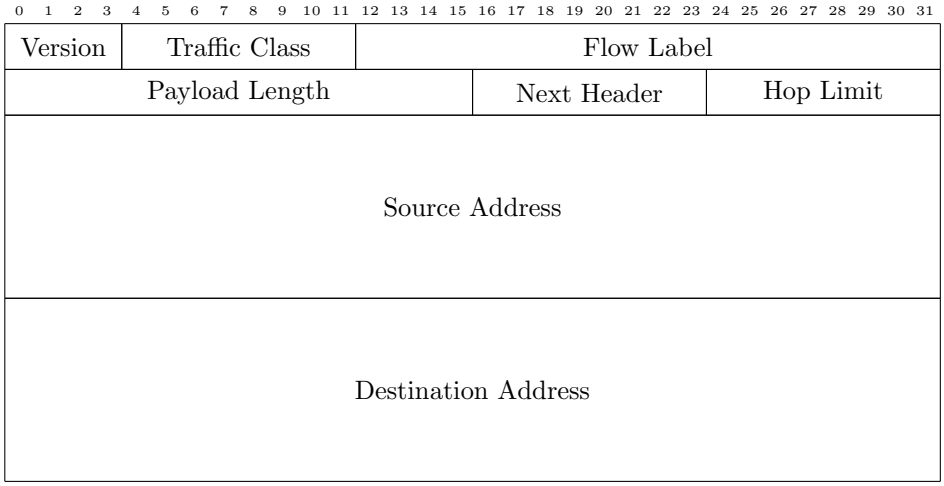

Figura 1: Formato fixed header IPv6

Notiamo che a differenza dei pacchetti IPv4 non sono presenti i campi *Flags* e *Fragment Offset* utilizzati nella versione 4 per ottenere informazioni sulla frammentazione. Per tale scopo, in IPv6, bisogna far riferimento all'extension header *Fragment*, mostrato in figura 2, nel quale sono contenute le informazioni necessarie per riasseblare i pacchetti originali.

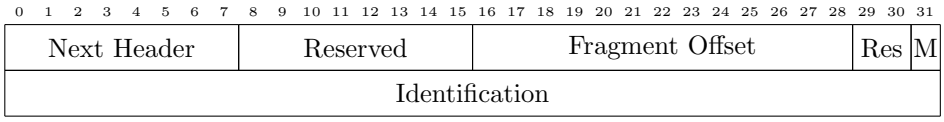

Figura 2: Formato fragment extension header IPv6. I campi *Res* e *Reserved* indicano spazio riservato inizializzato a 0. Il campo *M* si riferisce al campo *More Fragment*

Gli extension header sono situati tra il fixed header e gli header dei protocolli di livello superiore. Tramite il campo *Next Header* gli header formano una catena. In particolare il campo *Next Header* del fixed header indica il tipo del primo extension header e il *Next Header* dell'ultimo extension header (o del fixed header ove non ci fosse nessun extension header) indica il tipo dell'header del protocollo di livello superiore (ad esempio TCP o UDP).

2.2 Frammentazione IPv6 in TED

La modifica principale risiede nella funzione `ipv6_get_udp_info` la quale si occupa di recuperare le informazioni sul frammento IPv6 del relativo frame 802.11 al momento del suo invio.

Nella nuova versione si scorre innanzitutto la catena degli header fino ad incontrare l'header fragment:

```
/* Variables initialization */
*fragment_offset = *more_fragment = hdrs_len = error = 0;
nexthdr = ipv6_hdr(skb)->nexthdr;
target = NEXTHDR_FRAGMENT;

do {
        struct ipv6_opt_hdr _hdr, *hp;
        unsigned int hdrlen;
        found = (nexthdr == target);
```

```
        if ((!ipv6_ext_hdr(nexthdr)) || nexthdr == NEXTHDR_NONE) {
                break;
        }

        hp = skb_header_pointer(skb, offset, sizeof(_hdr), &_hdr);
        if (hp == NULL) {
                error = -EBADMSG;
                break;
        }

        if (nexthdr == NEXTHDR_FRAGMENT) {
                hdrlen = 8;
        } else if (nexthdr == NEXTHDR_AUTH) {
                hdrlen = (hp->hdrlen + 2) << 2;
        } else
                hdrlen = ipv6_optlen(hp);

        if (!found) {
                nexthdr = hp->nexthdr;
                offset += hdrlen;
        }

        hdrs_len += hdrlen;
} while (!found);
```

In realtà questa operazione è un controllo di sanità in quanto secondo le specifiche dell'RFC 2460 l'header fragment dovrebbe essere il diretto successore del fixed header. Ad ogni modo dopo aver individuato l'header fragment si può accedere alla bitmap `frag_off` della relativa struttura dati. Tramite la bitmap si possono ottenere quindi il campo *Fragment Offset* e il campo *More Fragment* del frammento:

```
/* fh is the pointer to the fragment header struct and it is retreived
according to the offset from the begging of the packet */
fh = skb_header_pointer(skb, offset, sizeof(_frag), &_frag);

if (fh) {
        *fragment_offset = ntohs(fh->frag_off) & ~0x7;
        *more_fragment = ((fh->frag_off & htons(IP6_MF)) > 0);
}
```

Infine l'ultimo campo che ci interessa ottenere è la lunghezza del frammento. La parte frammentabile è tutto quello che segue l'header fragment, quindi eventuali altri extension header IPv6, gli header dei protocolli dei livelli superiori e il frammento dati del messaggio. Bisogna quindi togliere dalla dimensione del frammento indicata dal campo *Payload* del fixed header, la dimensione dell'header fragment. Questo perchè il campo *Payload* esclude già la dimensione del fixed header e perchè l'header fragment è il diretto successore del fixed header; inoltre tutto quello che segue va considerato parte del frammento[1]:

```
*fragment_data_length = ntohs(payload_iphdr->payload_len) - hdrs_len;
```

A seguito di queste modiche il kernel con TED abilitato riesce correttamente ad ottenere le informazioni sulla frammentazione anche per IPv6.

[1] Sempre per controllo di sanità, viene sotratto al payload tutto quello che parte dalla fine del fixed header fino alla fine dell'header fragment.

3 Modifiche Applicazione

3.1 Gestione Asincrona delle Notifiche

3.2 Gestione della Frammentazione

3.3 Refactoring e Organizzazione

3.4 Utilizzo