Le voyage d'un paquet au travers de la pile réseau de linux 2.4

Harald Welte `laforge@gnumonks.org`
(Traduction: Alexandre Dagan `alexandre.dagan@linuxmail.org`)
Revision: 1.0, Date: 2001/07/12 14:53:28

Ce document décrit le trajet d'un paquet réseau à l'intérieur du noyau Linux 2.4.x. Il a changé énormément depuis la version 2.2 avec le choix d'un nouveau système baptisé softirq.

Contents

1 Préface

Je dois m'excuser pour mon ignorance, mais ce document reste très concentré sur le cas par défaut: une architecture x86 et des paquets IP qui sont forwardés.

Je ne suis absolument pas un gourou du noyau et les information fournies par ce document peuvent s'avérer fausses. Alors n'hésitez pas à envoyer vos commentaires ou corrections.

2 Réception du paquet

2.1 L'interruption en réception

Si la carte réseau reçoit une trame Ethernet qui correspond à l'adresse MAC locale ou qui est en broadcast, cela génère une interruption. Le driver réseau pour cette carte récupère capte l'interruption, va chercher les données du paquet via DMA, PIO ou autre pour les mettre en mémoire RAM. Il alloue alors un skb (socket buffer) et appelle une fonction appartenant aux routines de support de la carte indépendamment du protocole: `net/core/dev.c:netif_rx(skb)`.

Si le driver n'a pas encore daté le skb, c'est alors fait. Après cela, le skb est placé dans la file d'attente appropriée pour que le processeur puisse le traiter. Si la file est pleine, le paquet est détruit. Après la mise en file d'attente du skb la réception de l'interruption logicielle est marquée pour exécution via `include/linux/interrupt.h:__cpu_raise_softirq()`.

Le traitement de l'interruption se termine et toutes les interruptions sont ré-autorisées.

2.2 Le network RX softirq

Là nous rencontrons l'un des changements majeurs entre 2.2 et 2.4: la pile réseau entière n'est plus gérée par principe de polling mais par interruptions logicielles (softirq). Ces dernières ont l'avantage de pouvoir s'exécuter sur plusieurs processeurs simultanément. La gestion précédente ne pouvait se faire qu'avec un seul processeur à la fois.

L'enregistrement de la softirq de réception réseau se fait dans `net/core/dev.c:net_init()` en utilisant la fonction `kernel/softirq.c:open_softirq()` fournie par le sous-système de softirq.

Le traitement ultérieur des paquets est réalisé dans la softirq de réception réseau (NEXT_RX_SOFTIRQ) qui est appelée depuis `kernel/softirq.c:do_softirq()`.La fonction `do_softirq()` est elle-même appelée depuis trois endroits dans le noyau:

1. depuis `arch/i386/kernel/irc.c:do_IRQ()`, qui est le traitant des IRQ génériques
2. depuis `arch/i386/kernel/entry.S` dans le cas revient juste d'un appel système
3. à l'intérieur du processus principal de l'échéancier dans `kernel/sched.c:schedule()`

Donc si l'exécution accomplit un de ces points, la fonction `do_softirq()` est appelée, et détecte la marque NET_RX_SOFTIRQ et appelle `net/core/dev.c:net_rx_action()`. Le skb est enlevé de la file d'attente de réception de ce processeur et ensuite traité par le traitant de paquet approprié. Dans le cas d'IPV4, il s'agit de celle associée aux paquets IPV4.

2.3 La fonction de traitement des paquets IPV4

La fonction de traitement des paquets IP est enregistrée via `net/core/dev.c:dev_add_pack()` appelé depuis `net/ipv4/ip_output.c:ip_init()`.

La fonction en charge du traitement du paquet IPV4 est `net/ipv4/ip_input.c:ip_rcv()`. Après quelques vérifications préliminaires (si l'entête du paquet est correcte (IPv4) et si la taille est supérieure à 20 octets) le code vérificateur (checksum) est calculé.

Tous les paquets échouant à l'un des tests précédents est alors détruit (dropped).

Si le paquet passe les tests avec succès, on détermine la taille du paquet IP et on débarrasse le skb du bourrage éventuellement rajouté par la couche transport.

A cet instant, c'est le premier appel aux netfilter hooks.

Netfilter fournit une interface générique et abstraite au code de routage standard. Il est utilisé pour filtrer les paquets, les découper, la traduction d'adresse (NAT) (Network Address Translation) et la gestion des files d'attente vers l'espace utilisateur. Pour plus de détails, jetez un oeil sur ma présentation "Le sous système netfilter dans Linux 2.4" ou un des guides douteux de Rusty, i.e. the netfilter hacking guide.

Après une traversée réussie du netfilter hook, la fonction `net/ipv4/ipv_input.c:ip_rcv_finish()` est appelée.

A l'intérieur de `ip_rcv_finish()`, la destination du paquet est déterminée par l'appel à la fonction de routage `net/ipv4/route.c:ip_route_input()`. De plus, on recherche si une clef correspondant à ce flot de paquet (Ce paquet est soit le premier d'un flot soit membre d'un flot existant) n'existe pas dans la table de hashage. En outre, si notre paquet IP possède des options IP, elles sont traitées à ce moment. Si aucune clef n'existe, elle est calculée et insérée dans la table. Il est alors fait appel à `net/ipv4/route.c:ip_route_input_slow()`. Dans tous les cas, le trajet de notre paquet se poursuit par l'appel de la méthode `input` du skbuff qui a été positionnée dans la fonction `net/ipv4/route.c:ip_route_input()` à l'une des fonctions suivantes:

net/ipv4/ip_input.c:ip_local_deliver()

La destination du paquet est locale, nous devons nous occuper de la couche 4 du protocole et le transmettre à un processus utilisateur.

net/ipv4/ip_forward.c:ip_forward()

La destination du paquet n'est pas locale, nous devons le transmettre à un autre réseau.

net/ipv4/route.c:ip_error()

Une erreur est survenue, il n'est pas possible de trouver une entrée appropriée pour ce paquet dans la table de routage.

net/ipv4/ipmr.c:ip_mr_output()

C'est un paquet en multicast et nous devons faire un routage multicast.

3 Transmission de paquet à une autre entité physique (device)

Si le routage décide que le paquet doit être transmis, la fonction `net/ipv4/ip_forward.c:ip_forward()` est appelée.

La première tâche de cette fonction est de vérifier la partie de l'entête IP correspondant au TTL (Time To Live: durée de vie du paquet). Si elle est inférieure ou égale à 1, on détruit le paquet et on renvoie un message ICMP indiquant une durée de vie dépassée à l'expéditeur du dit paquet.

On vérifie alors l'entête du skb pour savoir si il reste suffisamment de place en fin de zone mémoire pour y placer l'entête de la couche liaison de l'entité. Le cas échéant, on agrandit le skb en conséquence.

Ensuite le TTL est décrémenté de 1.

Si notre paquet est plus grand que le MTU(Maximum Transport Unit: taille maximale d'un unité de transport dans un réseau) de destination et si le bit de non-fragmentation est à 1 dans l'entête IP, alors on détruit le paquet et on renvoie un message ICMP à l'expéditeur indiquant qu'une fragmentation est nécessaire.

Enfin il est temps de faire appel à un autre netfilter hook: NF_IP_FORWARD.

En considérant que le netfilter hook retourne la valeur NF_ACCEPT, la fonction `net/ipv4/ip_forward:ip_forward_finish()` est la prochaine étape du voyage de notre paquet.

La fonction `ip_forward_finish()` vérifie elle-même si il est nécessaire de rajouter d'éventuelles options dans l'entête IP, et fait appel aux fonctions de `net/ipv4/ip_options.c`. Après cela, elle appelle `include/net/ip.h:ip_send()`.

Ensuite, `ip_send()` est appelée et vérifie si une fragmentation est nécessaire (Si c'est le cas, il y aura appel à la fonction `net/ipv4/ip_fragment.c`), puis on continue dans la fonction `skb->dst->output()`, qui n'est autre que `net/ipv4/ip_forward:ip_output()`

La fonction `ip_output()` traite le NAT (Networking Address Translation: système de traduction d'adresses pour les réseaux privés dans le cas d'un firewal par exemple) puis appelle le netfilter postrouting hook NF_POSTROUTING_HOOK et `ip_finish_output2()` après une traversée réussie du hook.

L'appel à la fonction `ip_finish_output2()` met le matériel en attente de notre skb et appelle `net/ipv4/ip_output.c:ip_output()`.