

nnetfilter listification

Florian Westphal

4096R/AD5FF600 fw@strlen.de

80A9 20C5 B203 E069 F586

AE9F 7091 A8D9 AD5F F600

Red Hat

July 2019


Problem statement

- ▶ rx processing passes one skb to higher layers at a time
- ▶ several places (gro, backlog dequeue) could build lists containing more than one skb
- ▶ parts of stack have list-processing equivalents, up to ipv4 input
- ▶ currently does list deconstruction/construction where needed
- ▶ e.g. for netfilter, nf/tc ingress, rps, and so on
- ▶ is there anything useful netfilter could do with this?

current state from nf point of view

```
static inline void
NF_HOOK_LIST(u8 pf, u8 hook, struct net *net, struct sock *sk,
             struct list_head *head, struct net_device *in, struct net_device *out,
             int (*okfn)(struct net *, struct sock *, struct sk_buff *))
{
    struct sk_buff *skb, *next;
    LIST_HEAD_INIT(sublist);

    list_for_each_entry_safe(skb, next, head, list) {
        list_del(&skb->list);
        if (nf_hook(pf, hook, net, sk, skb, in, out, okfn) == 1)
            list_add_tail(&skb->list, &sublist);
    }
    list_splice(&sublist, head);
}
```

In short, iterate over list, pass each skb through netfilter core, collect results. 

more desirable state

```
static inline void
NF_HOOK_PREROUTING_LIST(u8 pf,
    struct list_head *head,
    int (*okfn)(struct net *, struct sock *, struct sk_buff *))
{
    nf_hook_prerouting(pf, head, okfn);
}
```

- ▶ sk and outdev are always NULL
- ▶ net, indev can differ for each skb
- ▶ list gets passed into netfilter core

But how does that help?

initial step is easy ...

```
nf_hook_slow_list(list_head *head, nf_hook_state *state,
                  const struct nf_hook_entries *e)
{
    [...]
    list_for_each_entry_safe(skb, next, head, list) {
        list_del(&skb->list);
        state->indev = skb->dev;
        state->net = dev_net(skb->dev);
        ret = nf_hook_slow(skb, state, e);
        if (ret == 1)
            list_add_tail(&skb->list, &sublist);
    }
    list_splice(&sublist, head);
}
```

pushes list deconstruction into nf core, so only minimal saves (we fetch list of hooks to run once per list, not per skb).

... but then it becomes hard

core problem:

```
typedef unsigned int nf_hookfn(void *priv,  
                                struct sk_buff *skb,  
                                const struct nf_hook_state *state);
```

- ▶ this is the function that all netfilter hooks implement
- ▶ can't be changed unless all are converted at once
- ▶ huge code churn
- ▶ very repetitive code pattern: `list_for_each_entry_safe` everywhere
- ▶ even worse: we lose return value – all hooks need to handle drop/queue/stolen

Can we convert gradually?

Which hooks are most useful/promising?

Gradual conversion

- ▶ could extend core to support both 'priv, skb, state' and 'priv, list, state' arguments for hook functions
- ▶ annotation tells core if list is needed or not

```
nf_hook_slow_list(list)
  for_each_hookfn(entries) {
    if (is_listified(e->hookfn)) {
      nf_call_listfn(e->hookfn, priv, list, state);
    } else {
for_each_skb_in_list(list) {
  list_del(&skb->list);
  ret = nf_call_hookfn(e->hookfn, priv, skb, state);

      if (ret == NF_ACCEPT) {
        list_add_tail(&skb->list, &sublist);
        [...] /* Handle other verdicts */
      }
}
```

Problems so far

- ▶ causes frequent list iterations in netfilter core
- ▶ can't be avoided: converted hook needs to iterate too
- ▶ saves indirect calls
- ▶ converted hook could try to be a bit smarter: we know when skbs are done
- ▶ this would be great for flow table infrastructure (which hooks at ingress).

ingress hook

- ▶ used by the fast-forward fastpath (software fallback for flow offloading)
- ▶ with list instead of skb we would know when batch is done
- ▶ could extend/patch stack to leverage `xmit_more` to NIC, i.e. delay TXTD update until the last packet
- ▶ major problem: network core doesn't pass a list to the function that calls `nf_ingress`

ingress hook (2)

```
__netif_receive_skb_list_core(struct list_head *head, ...
{
    struct packet_type *pt_curr = NULL; /* Current (common) ptype of sublist */
    struct net_device *od_curr = NULL; /* Current (common) orig_dev of sublist */
    LIST_HEAD_INIT(sublist);
    struct sk_buff *skb, *next;

    list_for_each_entry_safe(skb, next, head, list) {
        struct packet_type *pt_prev = NULL;

        __netif_receive_skb_core(skb, pfmemalloc, &pt_prev); // nf_ingress()
        if (!pt_prev) continue;
        ...
        __netif_receive_skb_list_ptype(&sublist, pt_curr, od_curr);
    }
}
```

Summary

- ▶ listification for netfilter seems doable, but questionable for several reasons
 - ▶ code churn
 - ▶ one list loop per (converted) hook
 - ▶ "save indirect call" is not worth the pain
- ▶ seems better to investigate alternative for the fast forwarding path (flow table)
- ▶ needs work outside of netfilter (bulk xmit) first