
Subject: [PATCH 1/9] Move common fields from frag_queues in one place
Posted by [Pavel Emelianov](#) on Fri, 12 Oct 2007 13:00:06 GMT
[View Forum Message](#) <> [Reply to Message](#)

Introduce the struct inet_frag_queue in include/net/inet_frag.h
file and place there all the common fields from three structs:

- * struct ipq in ipv4/ip_fragment.c
- * struct nf_ct_frag6_queue in nf_conntrack_reasm.c
- * struct frag_queue in ipv6/reassembly.c

After this, replace these fields on appropriate structures with
this structure instance and fix the users to use correct names
i.e. hunks like

- atomic_dec(&fq->refcnt);
- + atomic_dec(&fq->q.refcnt);

(these occupy most of the patch)

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
new file mode 100644
index 0000000..74e9cb9
--- /dev/null
+++ b/include/net/inet_frag.h
@@ -0,0 +1,21 @@
+#ifndef __NET_FRAG_H__
+#define __NET_FRAG_H__
+
+struct inet_frag_queue {
+ struct hlist_node list;
+ struct list_head lru_list; /* lru list member */
+ spinlock_t lock;
+ atomic_t refcnt;
+ struct timer_list timer; /* when will this queue expire? */
+ struct sk_buff *fragments; /* list of received fragments */
+ ktime_t stamp;
+ int len; /* total length of orig datagram */
+ int meat;
+ __u8 last_in; /* first/last segment arrived? */
+
+#define COMPLETE 4
+#define FIRST_IN 2
+#define LAST_IN 1
```

```

+};
+
+endif
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index fabb86d..3eb1b6d 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -39,6 +39,7 @@
#include <net/icmp.h>
#include <net/checksum.h>
#include <net/inetpeer.h>
+include <net/inet_frag.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/inet.h>
@@ -74,25 +75,13 @@ struct ipfrag_skb_cb

/* Describe an entry in the "incomplete datagrams" queue. */
struct ipq {
- struct hlist_node list;
- struct list_head lru_list; /* lru list member */
+ struct inet_frag_queue q;
+
+ u32 user;
+ __be32 saddr;
+ __be32 daddr;
+ __be16 id;
+ u8 protocol;
- u8 last_in;
-#define COMPLETE 4
-#define FIRST_IN 2
-#define LAST_IN 1
-
- struct sk_buff *fragments; /* linked list of received fragments */
- int len; /* total length of original datagram */
- int meat;
- spinlock_t lock;
- atomic_t refcnt;
- struct timer_list timer; /* when will this queue expire? */
- ktime_t stamp;
- int iif;
- unsigned int rid;
- struct inet_peer *peer;
@@ -111,8 +100,8 @@ int ip_frag_nqueues = 0;

static __inline__ void __ipq_unlink(struct ipq *qp)
{
- hlist_del(&qp->list);

```

```

- list_del(&qp->lru_list);
+ hlist_del(&qp->q.list);
+ list_del(&qp->q.lru_list);
  ip_frag_nqueues--;
}

@@ -144,15 +133,15 @@ static void ipfrag_secret_rebuild(unsigned long dummy)
  struct ipq *q;
  struct hlist_node *p, *n;

- hlist_for_each_entry_safe(q, p, n, &ipq_hash[i], list) {
+ hlist_for_each_entry_safe(q, p, n, &ipq_hash[i], q.list) {
  unsigned int hval = ipqhashfn(q->id, q->saddr,
    q->daddr, q->protocol);

    if (hval != i) {
-   hlist_del(&q->list);
+   hlist_del(&q->q.list);

      /* Relink to new hash chain. */
-   hlist_add_head(&q->list, &ipq_hash[hval]);
+   hlist_add_head(&q->q.list, &ipq_hash[hval]);
    }
  }
}

@@ -198,14 +187,14 @@ static void ip_frag_destroy(struct ipq *qp, int *work)
{
  struct sk_buff *fp;

- BUG_TRAP(qp->last_in&COMPLETE);
- BUG_TRAP(del_timer(&qp->timer) == 0);
+ BUG_TRAP(qp->q.last_in&COMPLETE);
+ BUG_TRAP(del_timer(&qp->q.timer) == 0);

  if (qp->peer)
    inet_putpeer(qp->peer);

  /* Release all fragment data. */
- fp = qp->fragments;
+ fp = qp->q.fragments;
  while (fp) {
    struct sk_buff *xp = fp->next;

@@ -219,7 +208,7 @@ static void ip_frag_destroy(struct ipq *qp, int *work)

static __inline__ void ipq_put(struct ipq *ipq, int *work)
{
- if (atomic_dec_and_test(&ipq->refcnt))

```

```

+ if (atomic_dec_and_test(&ipq->q.refcnt))
    ip_frag_destroy(ipq, work);
}

@@ -228,13 +217,13 @@ static __inline__ void ipq_put(struct ipq *ipq, int *work)
    */
    static void ipq_kill(struct ipq *ipq)
    {
- if (del_timer(&ipq->timer))
- atomic_dec(&ipq->refcnt);
+ if (del_timer(&ipq->q.timer))
+ atomic_dec(&ipq->q.refcnt);

- if (!(ipq->last_in & COMPLETE)) {
+ if (!(ipq->q.last_in & COMPLETE)) {
    ipq_unlink(ipq);
- atomic_dec(&ipq->refcnt);
- ipq->last_in |= COMPLETE;
+ atomic_dec(&ipq->q.refcnt);
+ ipq->q.last_in |= COMPLETE;
    }
}

@@ -258,14 +247,14 @@ static void ip_evictor(void)
    return;
}
    tmp = ipq_lru_list.next;
- qp = list_entry(tmp, struct ipq, lru_list);
- atomic_inc(&qp->refcnt);
+ qp = list_entry(tmp, struct ipq, q.lru_list);
+ atomic_inc(&qp->q.refcnt);
    read_unlock(&ipfrag_lock);

- spin_lock(&qp->lock);
- if (!(qp->last_in & COMPLETE))
+ spin_lock(&qp->q.lock);
+ if (!(qp->q.last_in & COMPLETE))
    ipq_kill(qp);
- spin_unlock(&qp->lock);
+ spin_unlock(&qp->q.lock);

    ipq_put(qp, &work);
    IP_INC_STATS_BH(IPSTATS_MIB_REASMFAILS);
@@ -279,9 +268,9 @@ static void ip_expire(unsigned long arg)
{
    struct ipq *qp = (struct ipq *) arg;

- spin_lock(&qp->lock);

```

```

+ spin_lock(&qp->q.lock);

- if (qp->last_in & COMPLETE)
+ if (qp->q.last_in & COMPLETE)
    goto out;

    ipq_kill(qp);
@@ -289,8 +278,8 @@ static void ip_expire(unsigned long arg)
    IP_INC_STATS_BH(IPSTATS_MIB_REASMTIMEOUT);
    IP_INC_STATS_BH(IPSTATS_MIB_REASMFAILS);

- if ((qp->last_in&FIRST_IN) && qp->fragments != NULL) {
- struct sk_buff *head = qp->fragments;
+ if ((qp->q.last_in&FIRST_IN) && qp->q.fragments != NULL) {
+ struct sk_buff *head = qp->q.fragments;
    /* Send an ICMP "Fragment Reassembly Timeout" message. */
    if ((head->dev = dev_get_by_index(&init_net, qp->iif)) != NULL) {
        icmp_send(head, ICMP_TIME_EXCEEDED, ICMP_EXC_FRAGTIME, 0);
@@ -298,7 +287,7 @@ static void ip_expire(unsigned long arg)
    }
}
out:
- spin_unlock(&qp->lock);
+ spin_unlock(&qp->q.lock);
    ipq_put(qp, NULL);
}

@@ -320,15 +309,15 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
    * such entry could be created on other cpu, while we
    * promoted read lock to write lock.
    */
- hlist_for_each_entry(qp, n, &ipq_hash[hash], list) {
+ hlist_for_each_entry(qp, n, &ipq_hash[hash], q.list) {
    if (qp->id == qp_in->id &&
        qp->saddr == qp_in->saddr &&
        qp->daddr == qp_in->daddr &&
        qp->protocol == qp_in->protocol &&
        qp->user == qp_in->user) {
- atomic_inc(&qp->refcnt);
+ atomic_inc(&qp->q.refcnt);
        write_unlock(&ipfrag_lock);
- qp_in->last_in |= COMPLETE;
+ qp_in->q.last_in |= COMPLETE;
        ipq_put(qp_in, NULL);
        return qp;
    }
}
@@ -336,13 +325,13 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
#endif

```

```

qp = qp_in;

- if (!mod_timer(&qp->timer, jiffies + sysctl_ipfrag_time))
- atomic_inc(&qp->refcnt);
+ if (!mod_timer(&qp->q.timer, jiffies + sysctl_ipfrag_time))
+ atomic_inc(&qp->q.refcnt);

- atomic_inc(&qp->refcnt);
- hlist_add_head(&qp->list, &ipq_hash[hash]);
- INIT_LIST_HEAD(&qp->lru_list);
- list_add_tail(&qp->lru_list, &ipq_lru_list);
+ atomic_inc(&qp->q.refcnt);
+ hlist_add_head(&qp->q.list, &ipq_hash[hash]);
+ INIT_LIST_HEAD(&qp->q.lru_list);
+ list_add_tail(&qp->q.lru_list, &ipq_lru_list);
  ip_frag_nqueues++;
  write_unlock(&ipfrag_lock);
  return qp;
@@ -357,23 +346,23 @@ static struct ipq *ip_frag_create(struct iphdr *iph, u32 user)
  goto out_nomem;

  qp->protocol = iph->protocol;
- qp->last_in = 0;
+ qp->q.last_in = 0;
  qp->id = iph->id;
  qp->saddr = iph->saddr;
  qp->daddr = iph->daddr;
  qp->user = user;
- qp->len = 0;
- qp->meat = 0;
- qp->fragments = NULL;
+ qp->q.len = 0;
+ qp->q.meat = 0;
+ qp->q.fragments = NULL;
  qp->iif = 0;
  qp->peer = sysctl_ipfrag_max_dist ? inet_getpeer(iph->saddr, 1) : NULL;

  /* Initialize a timer for this entry. */
- init_timer(&qp->timer);
- qp->timer.data = (unsigned long) qp; /* pointer to queue */
- qp->timer.function = ip_expire; /* expire function */
- spin_lock_init(&qp->lock);
- atomic_set(&qp->refcnt, 1);
+ init_timer(&qp->q.timer);
+ qp->q.timer.data = (unsigned long) qp; /* pointer to queue */
+ qp->q.timer.function = ip_expire; /* expire function */
+ spin_lock_init(&qp->q.lock);
+ atomic_set(&qp->q.refcnt, 1);

```

```

return ip_frag_intern(qp);

@@ -397,13 +386,13 @@ static inline struct ipq *ip_find(struct iphdr *iph, u32 user)

    read_lock(&ipfrag_lock);
    hash = ipqhashfn(id, saddr, daddr, protocol);
- hlist_for_each_entry(qp, n, &ipq_hash[hash], list) {
+ hlist_for_each_entry(qp, n, &ipq_hash[hash], q.list) {
    if (qp->id == id &&
        qp->saddr == saddr &&
        qp->daddr == daddr &&
        qp->protocol == protocol &&
        qp->user == user) {
- atomic_inc(&qp->refcnt);
+ atomic_inc(&qp->q.refcnt);
    read_unlock(&ipfrag_lock);
    return qp;
    }
@@ -429,7 +418,7 @@ static inline int ip_frag_too_far(struct ipq *qp)
    end = atomic_inc_return(&peer->rid);
    qp->rid = end;

- rc = qp->fragments && (end - start) > max;
+ rc = qp->q.fragments && (end - start) > max;

    if (rc) {
        IP_INC_STATS_BH(IPSTATS_MIB_REASMFAILS);
@@ -442,22 +431,22 @@ static int ip_frag_reinit(struct ipq *qp)
    {
        struct sk_buff *fp;

- if (!mod_timer(&qp->timer, jiffies + sysctl_ipfrag_time)) {
- atomic_inc(&qp->refcnt);
+ if (!mod_timer(&qp->q.timer, jiffies + sysctl_ipfrag_time)) {
+ atomic_inc(&qp->q.refcnt);
        return -ETIMEDOUT;
    }

- fp = qp->fragments;
+ fp = qp->q.fragments;
    do {
        struct sk_buff *xp = fp->next;
        frag_kfree_skb(fp, NULL);
        fp = xp;
    } while (fp);

- qp->last_in = 0;

```

```

- qp->len = 0;
- qp->meat = 0;
- qp->fragments = NULL;
+ qp->q.last_in = 0;
+ qp->q.len = 0;
+ qp->q.meat = 0;
+ qp->q.fragments = NULL;
  qp->iif = 0;

  return 0;
@@ -470,7 +459,7 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
  int flags, offset;
  int ihl, end;

- if (qp->last_in & COMPLETE)
+ if (qp->q.last_in & COMPLETE)
  goto err;

  if (!(IPCB(skb)->flags & IPSKB_FRAG_COMPLETE) &&
@@ -493,22 +482,22 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
  /* If we already have some bits beyond end
   * or have different end, the segment is corrupted.
   */
- if (end < qp->len ||
-     ((qp->last_in & LAST_IN) && end != qp->len))
+ if (end < qp->q.len ||
+     ((qp->q.last_in & LAST_IN) && end != qp->q.len))
  goto err;
- qp->last_in |= LAST_IN;
- qp->len = end;
+ qp->q.last_in |= LAST_IN;
+ qp->q.len = end;
  } else {
    if (end&7) {
      end &= ~7;
      if (skb->ip_summed != CHECKSUM_UNNECESSARY)
        skb->ip_summed = CHECKSUM_NONE;
    }
- if (end > qp->len) {
+ if (end > qp->q.len) {
  /* Some bits beyond end -> corruption. */
- if (qp->last_in & LAST_IN)
+ if (qp->q.last_in & LAST_IN)
  goto err;
- qp->len = end;
+ qp->q.len = end;
  }
  }
}

```



```

    if (end == offset)
@@ -524,7 +513,7 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
    * this fragment, right?
    */
    prev = NULL;
- for (next = qp->fragments; next != NULL; next = next->next) {
+ for (next = qp->q.fragments; next != NULL; next = next->next) {
    if (FRAG_CB(next)->offset >= offset)
        break; /* bingo! */
    prev = next;
@@ -558,7 +547,7 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
    if (!pskb_pull(next, i))
        goto err;
    FRAG_CB(next)->offset += i;
- qp->meat -= i;
+ qp->q.meat -= i;
    if (next->ip_summed != CHECKSUM_UNNECESSARY)
        next->ip_summed = CHECKSUM_NONE;
    break;
@@ -573,9 +562,9 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
    if (prev)
        prev->next = next;
    else
- qp->fragments = next;
+ qp->q.fragments = next;

- qp->meat -= free_it->len;
+ qp->q.meat -= free_it->len;
    frag_kfree_skb(free_it, NULL);
}
}
@@ -587,19 +576,19 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
    if (prev)
        prev->next = skb;
    else
- qp->fragments = skb;
+ qp->q.fragments = skb;

    if (skb->dev)
        qp->iif = skb->dev->ifindex;
    skb->dev = NULL;
- qp->stamp = skb->tstamp;
- qp->meat += skb->len;
+ qp->q.stamp = skb->tstamp;
+ qp->q.meat += skb->len;
    atomic_add(skb->truesize, &ip_frag_mem);
    if (offset == 0)
- qp->last_in |= FIRST_IN;

```

```

+ qp->q.last_in |= FIRST_IN;

    write_lock(&ipfrag_lock);
- list_move_tail(&qp->lru_list, &ipq_lru_list);
+ list_move_tail(&qp->q.lru_list, &ipq_lru_list);
    write_unlock(&ipfrag_lock);

    return;
@@ -614,7 +603,7 @@ err:
static struct sk_buff *ip_frag_reasm(struct ipq *qp, struct net_device *dev)
{
    struct iphdr *iph;
- struct sk_buff *fp, *head = qp->fragments;
+ struct sk_buff *fp, *head = qp->q.fragments;
    int len;
    int ihlen;

@@ -625,7 +614,7 @@ static struct sk_buff *ip_frag_reasm(struct ipq *qp, struct net_device
*dev)

    /* Allocate a new buffer for the datagram. */
    ihlen = ip_hdrlen(head);
- len = ihlen + qp->len;
+ len = ihlen + qp->q.len;

    if (len > 65535)
        goto out_oversize;
@@ -674,13 +663,13 @@ static struct sk_buff *ip_frag_reasm(struct ipq *qp, struct net_device
*dev)

    head->next = NULL;
    head->dev = dev;
- head->tstamp = qp->stamp;
+ head->tstamp = qp->q.stamp;

    iph = ip_hdr(head);
    iph->frag_off = 0;
    iph->tot_len = htons(len);
    IP_INC_STATS_BH(IPSTATS_MIB_REASMOKS);
- qp->fragments = NULL;
+ qp->q.fragments = NULL;
    return head;

out_nomem:
@@ -715,15 +704,15 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)
    if ((qp = ip_find(ip_hdr(skb), user)) != NULL) {
        struct sk_buff *ret = NULL;

```

```

- spin_lock(&qp->lock);
+ spin_lock(&qp->q.lock);

    ip_frag_queue(qp, skb);

- if (qp->last_in == (FIRST_IN|LAST_IN) &&
-     qp->meat == qp->len)
+ if (qp->q.last_in == (FIRST_IN|LAST_IN) &&
+     qp->q.meat == qp->q.len)
    ret = ip_frag_reasm(qp, dev);

- spin_unlock(&qp->lock);
+ spin_unlock(&qp->q.lock);
    ipq_put(qp, NULL);
    return ret;
}
diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 25442a8..52e9f6a 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -31,6 +31,7 @@

#include <net/sock.h>
#include <net/snmp.h>
+#include <net/inet_frag.h>

#include <net/ipv6.h>
#include <net/protocol.h>
@@ -63,25 +64,13 @@ struct nf_ct_frag6_skb_cb

struct nf_ct_frag6_queue
{
- struct hlist_node list;
- struct list_head lru_list; /* lru list member */
+ struct inet_frag_queue q;

    __be32 id; /* fragment id */
    struct in6_addr saddr;
    struct in6_addr daddr;

- spinlock_t lock;
- atomic_t refcnt;
- struct timer_list timer; /* expire timer */
- struct sk_buff *fragments;
- int len;
- int meat;
- ktime_t stamp;
- unsigned int csum;

```

```

- __u8 last_in; /* has first/last segment arrived? */
-#define COMPLETE 4
-#define FIRST_IN 2
-#define LAST_IN 1
__u16 nhoffset;
};

@@ -97,8 +86,8 @@ int nf_ct_frag6_nqueues = 0;

static __inline__ void __fq_unlink(struct nf_ct_frag6_queue *fq)
{
- hlist_del(&fq->list);
- list_del(&fq->lru_list);
+ hlist_del(&fq->q.list);
+ list_del(&fq->q.lru_list);
  nf_ct_frag6_nqueues--;
}

@@ -150,14 +139,14 @@ static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
  struct nf_ct_frag6_queue *q;
  struct hlist_node *p, *n;

- hlist_for_each_entry_safe(q, p, n, &nf_ct_frag6_hash[i], list) {
+ hlist_for_each_entry_safe(q, p, n, &nf_ct_frag6_hash[i], q.list) {
  unsigned int hval = ip6qhashfn(q->id,
    &q->saddr,
    &q->daddr);
  if (hval != i) {
- hlist_del(&q->list);
+ hlist_del(&q->q.list);
    /* Relink to new hash chain. */
- hlist_add_head(&q->list,
+ hlist_add_head(&q->q.list,
    &nf_ct_frag6_hash[hval]);
  }
}

@@ -208,11 +197,11 @@ static void nf_ct_frag6_destroy(struct nf_ct_frag6_queue *fq,
{
  struct sk_buff *fp;

- BUG_TRAP(fq->last_in & COMPLETE);
- BUG_TRAP(del_timer(&fq->timer) == 0);
+ BUG_TRAP(fq->q.last_in & COMPLETE);
+ BUG_TRAP(del_timer(&fq->q.timer) == 0);

  /* Release all fragment data. */
- fp = fq->fragments;
+ fp = fq->q.fragments;

```

```

while (fp) {
    struct sk_buff *xp = fp->next;

@@ -225,7 +214,7 @@ static void nf_ct_frag6_destroy(struct nf_ct_frag6_queue *fq,

static __inline__ void fq_put(struct nf_ct_frag6_queue *fq, unsigned int *work)
{
- if (atomic_dec_and_test(&fq->refcnt))
+ if (atomic_dec_and_test(&fq->q.refcnt))
    nf_ct_frag6_destroy(fq, work);
}

@@ -234,13 +223,13 @@ static __inline__ void fq_put(struct nf_ct_frag6_queue *fq, unsigned int
*work)
*/
static __inline__ void fq_kill(struct nf_ct_frag6_queue *fq)
{
- if (del_timer(&fq->timer))
- atomic_dec(&fq->refcnt);
+ if (del_timer(&fq->q.timer))
+ atomic_dec(&fq->q.refcnt);

- if (!(fq->last_in & COMPLETE)) {
+ if (!(fq->q.last_in & COMPLETE)) {
    fq_unlink(fq);
- atomic_dec(&fq->refcnt);
- fq->last_in |= COMPLETE;
+ atomic_dec(&fq->q.refcnt);
+ fq->q.last_in |= COMPLETE;
}
}

@@ -263,14 +252,14 @@ static void nf_ct_frag6_evictor(void)
}
tmp = nf_ct_frag6_lru_list.next;
BUG_ON(tmp == NULL);
- fq = list_entry(tmp, struct nf_ct_frag6_queue, lru_list);
- atomic_inc(&fq->refcnt);
+ fq = list_entry(tmp, struct nf_ct_frag6_queue, q.lru_list);
+ atomic_inc(&fq->q.refcnt);
    read_unlock(&nf_ct_frag6_lock);

- spin_lock(&fq->lock);
- if (!(fq->last_in & COMPLETE))
+ spin_lock(&fq->q.lock);
+ if (!(fq->q.last_in & COMPLETE))
    fq_kill(fq);
- spin_unlock(&fq->lock);

```

```

+ spin_unlock(&fq->q.lock);

    fq_put(fq, &work);
}
@@ -280,15 +269,15 @@ static void nf_ct_frag6_expire(unsigned long data)
{
    struct nf_ct_frag6_queue *fq = (struct nf_ct_frag6_queue *) data;

- spin_lock(&fq->lock);
+ spin_lock(&fq->q.lock);

- if (fq->last_in & COMPLETE)
+ if (fq->q.last_in & COMPLETE)
    goto out;

    fq_kill(fq);

out:
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);
    fq_put(fq, NULL);
}

@@ -304,13 +293,13 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,

    write_lock(&nf_ct_frag6_lock);
#ifdef CONFIG_SMP
- hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], list) {
+ hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], q.list) {
    if (fq->id == fq_in->id &&
        ipv6_addr_equal(&fq_in->saddr, &fq->saddr) &&
        ipv6_addr_equal(&fq_in->daddr, &fq->daddr)) {
- atomic_inc(&fq->refcnt);
+ atomic_inc(&fq->q.refcnt);
        write_unlock(&nf_ct_frag6_lock);
- fq_in->last_in |= COMPLETE;
+ fq_in->q.last_in |= COMPLETE;
        fq_put(fq_in, NULL);
        return fq;
    }
}
@@ -318,13 +307,13 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
#ifdef CONFIG_SMP
    fq = fq_in;

- if (!mod_timer(&fq->timer, jiffies + nf_ct_frag6_timeout))
- atomic_inc(&fq->refcnt);
+ if (!mod_timer(&fq->q.timer, jiffies + nf_ct_frag6_timeout))
+ atomic_inc(&fq->q.refcnt);

```

```

- atomic_inc(&fq->refcnt);
- hlist_add_head(&fq->list, &nf_ct_frag6_hash[hash]);
- INIT_LIST_HEAD(&fq->lru_list);
- list_add_tail(&fq->lru_list, &nf_ct_frag6_lru_list);
+ atomic_inc(&fq->q.refcnt);
+ hlist_add_head(&fq->q.list, &nf_ct_frag6_hash[hash]);
+ INIT_LIST_HEAD(&fq->q.lru_list);
+ list_add_tail(&fq->q.lru_list, &nf_ct_frag6_lru_list);
  nf_ct_frag6_nqueues++;
  write_unlock(&nf_ct_frag6_lock);
  return fq;
@@ -347,9 +336,9 @@ nf_ct_frag6_create(unsigned int hash, __be32 id, struct in6_addr *src,
str
  ipv6_addr_copy(&fq->saddr, src);
  ipv6_addr_copy(&fq->daddr, dst);

- setup_timer(&fq->timer, nf_ct_frag6_expire, (unsigned long)fq);
- spin_lock_init(&fq->lock);
- atomic_set(&fq->refcnt, 1);
+ setup_timer(&fq->q.timer, nf_ct_frag6_expire, (unsigned long)fq);
+ spin_lock_init(&fq->q.lock);
+ atomic_set(&fq->q.refcnt, 1);

  return nf_ct_frag6_intern(hash, fq);

@@ -365,11 +354,11 @@ fq_find(__be32 id, struct in6_addr *src, struct in6_addr *dst)
  unsigned int hash = ip6qhashfn(id, src, dst);

  read_lock(&nf_ct_frag6_lock);
- hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], list) {
+ hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], q.list) {
    if (fq->id == id &&
        ipv6_addr_equal(src, &fq->saddr) &&
        ipv6_addr_equal(dst, &fq->daddr)) {
-   atomic_inc(&fq->refcnt);
+   atomic_inc(&fq->q.refcnt);
    read_unlock(&nf_ct_frag6_lock);
    return fq;
  }
@@ -386,7 +375,7 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,
  struct sk_buff *prev, *next;
  int offset, end;

- if (fq->last_in & COMPLETE) {
+ if (fq->q.last_in & COMPLETE) {
  pr_debug("Already completed\n");

```

```

    goto err;
}
@@ -412,13 +401,13 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct
sk_buff *skb,
/* If we already have some bits beyond end
 * or have different end, the segment is corrupted.
 */
- if (end < fq->len ||
-     ((fq->last_in & LAST_IN) && end != fq->len)) {
+ if (end < fq->q.len ||
+     ((fq->q.last_in & LAST_IN) && end != fq->q.len)) {
    pr_debug("already received last fragment\n");
    goto err;
}
- fq->last_in |= LAST_IN;
- fq->len = end;
+ fq->q.last_in |= LAST_IN;
+ fq->q.len = end;
} else {
/* Check if the fragment is rounded to 8 bytes.
 * Required by the RFC.
@@ -430,13 +419,13 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct
sk_buff *skb,
    pr_debug("end of fragment not rounded to 8 bytes.\n");
    return -1;
}
- if (end > fq->len) {
+ if (end > fq->q.len) {
/* Some bits beyond end -> corruption. */
- if (fq->last_in & LAST_IN) {
+ if (fq->q.last_in & LAST_IN) {
    pr_debug("last packet already reached.\n");
    goto err;
}
- fq->len = end;
+ fq->q.len = end;
}
}

@@ -458,7 +447,7 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,
/* this fragment, right?
 */
prev = NULL;
- for (next = fq->fragments; next != NULL; next = next->next) {
+ for (next = fq->q.fragments; next != NULL; next = next->next) {
    if (NFCT_FRAG6_CB(next)->offset >= offset)
        break; /* bingo! */

```



```

    prev = next;
@@ -503,7 +492,7 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,

    /* next fragment */
    NFCT_FRAG6_CB(next)->offset += i;
-   fq->meat -= i;
+   fq->q.meat -= i;
    if (next->ip_summed != CHECKSUM_UNNECESSARY)
        next->ip_summed = CHECKSUM_NONE;
    break;
@@ -518,9 +507,9 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,
    if (prev)
        prev->next = next;
    else
-   fq->fragments = next;
+   fq->q.fragments = next;

-   fq->meat -= free_it->len;
+   fq->q.meat -= free_it->len;
    frag_kfree_skb(free_it, NULL);
}
}
@@ -532,11 +521,11 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct
sk_buff *skb,
    if (prev)
        prev->next = skb;
    else
-   fq->fragments = skb;
+   fq->q.fragments = skb;

    skb->dev = NULL;
-   fq->stamp = skb->tstamp;
-   fq->meat += skb->len;
+   fq->q.stamp = skb->tstamp;
+   fq->q.meat += skb->len;
    atomic_add(skb->truesize, &nf_ct_frag6_mem);

    /* The first fragment.
@@ -544,10 +533,10 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct
sk_buff *skb,
    */
    if (offset == 0) {
        fq->nhoffset = nhoff;
-   fq->last_in |= FIRST_IN;
+   fq->q.last_in |= FIRST_IN;
    }
}

```

```

    write_lock(&nf_ct_frag6_lock);
- list_move_tail(&fq->lru_list, &nf_ct_frag6_lru_list);
+ list_move_tail(&fq->q.lru_list, &nf_ct_frag6_lru_list);
    write_unlock(&nf_ct_frag6_lock);
    return 0;

@@ -567,7 +556,7 @@ err:
    static struct sk_buff *
    nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)
    {
- struct sk_buff *fp, *op, *head = fq->fragments;
+ struct sk_buff *fp, *op, *head = fq->q.fragments;
    int    payload_len;

    fq_kill(fq);
@@ -577,7 +566,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)

    /* Unfragmented part is taken from the first segment. */
    payload_len = ((head->data - skb_network_header(head)) -
-      sizeof(struct ipv6hdr) + fq->len -
+      sizeof(struct ipv6hdr) + fq->q.len -
        sizeof(struct frag_hdr));
    if (payload_len > IPV6_MAXPLEN) {
        pr_debug("payload len is too large.\n");
@@ -643,7 +632,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)

    head->next = NULL;
    head->dev = dev;
- head->tstamp = fq->stamp;
+ head->tstamp = fq->q.stamp;
    ipv6_hdr(head)->payload_len = htons(payload_len);

    /* Yes, and fold redundant checksum back. 8) */
@@ -652,7 +641,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)
        skb_network_header_len(head),
        head->csum);

- fq->fragments = NULL;
+ fq->q.fragments = NULL;

    /* all original skbs are linked into the NFCT_FRAG6_CB(head).orig */
    fp = skb_shinfo(head)->frag_list;
@@ -797,21 +786,21 @@ struct sk_buff *nf_ct_frag6_gather(struct sk_buff *skb)
    goto ret_orig;
}

- spin_lock(&fq->lock);
+ spin_lock(&fq->q.lock);

```

```

    if (nf_ct_frag6_queue(fq, clone, fhdr, nhoff) < 0) {
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);
    pr_debug("Can't insert skb to queue\n");
    fq_put(fq, NULL);
    goto ret_orig;
}

- if (fq->last_in == (FIRST_IN|LAST_IN) && fq->meat == fq->len) {
+ if (fq->q.last_in == (FIRST_IN|LAST_IN) && fq->q.meat == fq->q.len) {
    ret_skb = nf_ct_frag6_reasm(fq, dev);
    if (ret_skb == NULL)
        pr_debug("Can't reassemble fragmented packets\n");
}
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);

```

```

    fq_put(fq, NULL);
    return ret_skb;
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 31601c9..f48ecc6 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -53,6 +53,7 @@
#include <net/rawv6.h>
#include <net/ndisc.h>
#include <net/addrconf.h>
+#include <net/inet_frag.h>

```

```

int sysctl_ip6frag_high_thresh __read_mostly = 256*1024;
int sysctl_ip6frag_low_thresh __read_mostly = 192*1024;
@@ -74,26 +75,14 @@ struct ip6frag_skb_cb

```

```

struct frag_queue
{
- struct hlist_node list;
- struct list_head lru_list; /* lru list member */
+ struct inet_frag_queue q;

    __be32 id; /* fragment id */
    struct in6_addr saddr;
    struct in6_addr daddr;

- spinlock_t lock;
- atomic_t refcnt;
- struct timer_list timer; /* expire timer */
- struct sk_buff *fragments;

```

```

- int len;
- int meat;
  int iif;
- ktime_t stamp;
  unsigned int csum;
- __u8 last_in; /* has first/last segment arrived? */
-#define COMPLETE 4
-#define FIRST_IN 2
-#define LAST_IN 1
  __u16 nhoffset;
};

```

```

@@ -109,8 +98,8 @@ int ip6_frag_nqueues = 0;

```

```

static __inline__ void __fq_unlink(struct frag_queue *fq)
{
- hlist_del(&fq->list);
- list_del(&fq->lru_list);
+ hlist_del(&fq->q.list);
+ list_del(&fq->q.lru_list);
  ip6_frag_nqueues--;
}

```

```

@@ -166,16 +155,16 @@ static void ip6_frag_secret_rebuild(unsigned long dummy)
  struct frag_queue *q;
  struct hlist_node *p, *n;

```

```

- hlist_for_each_entry_safe(q, p, n, &ip6_frag_hash[i], list) {
+ hlist_for_each_entry_safe(q, p, n, &ip6_frag_hash[i], q.list) {
  unsigned int hval = ip6qhashfn(q->id,
    &q->saddr,
    &q->daddr);

```

```

  if (hval != i) {
- hlist_del(&q->list);
+ hlist_del(&q->q.list);

  /* Relink to new hash chain. */
- hlist_add_head(&q->list,
+ hlist_add_head(&q->q.list,
    &ip6_frag_hash[hval]);

```

```

  }
@@ -222,11 +211,11 @@ static void ip6_frag_destroy(struct frag_queue *fq, int *work)
{
  struct sk_buff *fp;

- BUG_TRAP(fq->last_in & COMPLETE);

```

```

- BUG_TRAP(del_timer(&fq->timer) == 0);
+ BUG_TRAP(fq->q.last_in & COMPLETE);
+ BUG_TRAP(del_timer(&fq->q.timer) == 0);

/* Release all fragment data. */
- fp = fq->fragments;
+ fp = fq->q.fragments;
while (fp) {
    struct sk_buff *xp = fp->next;

@@ -239,7 +228,7 @@ static void ip6_frag_destroy(struct frag_queue *fq, int *work)

static __inline__ void fq_put(struct frag_queue *fq, int *work)
{
- if (atomic_dec_and_test(&fq->refcnt))
+ if (atomic_dec_and_test(&fq->q.refcnt))
    ip6_frag_destroy(fq, work);
}

@@ -248,13 +237,13 @@ static __inline__ void fq_put(struct frag_queue *fq, int *work)
*/
static __inline__ void fq_kill(struct frag_queue *fq)
{
- if (del_timer(&fq->timer))
- atomic_dec(&fq->refcnt);
+ if (del_timer(&fq->q.timer))
+ atomic_dec(&fq->q.refcnt);

- if (!(fq->last_in & COMPLETE)) {
+ if (!(fq->q.last_in & COMPLETE)) {
    fq_unlink(fq);
- atomic_dec(&fq->refcnt);
- fq->last_in |= COMPLETE;
+ atomic_dec(&fq->q.refcnt);
+ fq->q.last_in |= COMPLETE;
}
}

@@ -275,14 +264,14 @@ static void ip6_evictor(struct inet6_dev *idev)
    return;
}
tmp = ip6_frag_lru_list.next;
- fq = list_entry(tmp, struct frag_queue, lru_list);
- atomic_inc(&fq->refcnt);
+ fq = list_entry(tmp, struct frag_queue, q.lru_list);
+ atomic_inc(&fq->q.refcnt);
    read_unlock(&ip6_frag_lock);

```

```

- spin_lock(&fq->lock);
- if (!(fq->last_in & COMPLETE))
+ spin_lock(&fq->q.lock);
+ if (!(fq->q.last_in & COMPLETE))
    fq_kill(fq);
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);

    fq_put(fq, &work);
    IP6_INC_STATS_BH(iddev, IPSTATS_MIB_REASMFAILS);
@@ -294,9 +283,9 @@ static void ip6_frag_expire(unsigned long data)
    struct frag_queue *fq = (struct frag_queue *) data;
    struct net_device *dev = NULL;

- spin_lock(&fq->lock);
+ spin_lock(&fq->q.lock);

- if (fq->last_in & COMPLETE)
+ if (fq->q.last_in & COMPLETE)
    goto out;

    fq_kill(fq);
@@ -311,7 +300,7 @@ static void ip6_frag_expire(unsigned long data)
    rcu_read_unlock();

    /* Don't send error if the first segment did not arrive. */
- if (!(fq->last_in & FIRST_IN) || !fq->fragments)
+ if (!(fq->q.last_in & FIRST_IN) || !fq->q.fragments)
    goto out;

    /*
@@ -319,12 +308,12 @@ static void ip6_frag_expire(unsigned long data)
    segment was received. And do not use fq->dev
    pointer directly, device might already disappeared.
    */
- fq->fragments->dev = dev;
- icmpv6_send(fq->fragments, ICMPV6_TIME_EXCEED, ICMPV6_EXC_FRAGTIME, 0, dev);
+ fq->q.fragments->dev = dev;
+ icmpv6_send(fq->q.fragments, ICMPV6_TIME_EXCEED, ICMPV6_EXC_FRAGTIME, 0, dev);
out:
    if (dev)
        dev_put(dev);
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);
    fq_put(fq, NULL);
}

@@ -342,13 +331,13 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)

```

```

write_lock(&ip6_frag_lock);
hash = ip6qhashfn(fq_in->id, &fq_in->saddr, &fq_in->daddr);
#ifdef CONFIG_SMP
- hlist_for_each_entry(fq, n, &ip6_frag_hash[hash], list) {
+ hlist_for_each_entry(fq, n, &ip6_frag_hash[hash], q.list) {
    if (fq->id == fq_in->id &&
        ipv6_addr_equal(&fq_in->saddr, &fq->saddr) &&
        ipv6_addr_equal(&fq_in->daddr, &fq->daddr)) {
- atomic_inc(&fq->refcnt);
+ atomic_inc(&fq->q.refcnt);
    write_unlock(&ip6_frag_lock);
- fq_in->last_in |= COMPLETE;
+ fq_in->q.last_in |= COMPLETE;
    fq_put(fq_in, NULL);
    return fq;
}
@@ -356,13 +345,13 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)
#endif
fq = fq_in;

- if (!mod_timer(&fq->timer, jiffies + sysctl_ip6frag_time))
- atomic_inc(&fq->refcnt);
+ if (!mod_timer(&fq->q.timer, jiffies + sysctl_ip6frag_time))
+ atomic_inc(&fq->q.refcnt);

- atomic_inc(&fq->refcnt);
- hlist_add_head(&fq->list, &ip6_frag_hash[hash]);
- INIT_LIST_HEAD(&fq->lru_list);
- list_add_tail(&fq->lru_list, &ip6_frag_lru_list);
+ atomic_inc(&fq->q.refcnt);
+ hlist_add_head(&fq->q.list, &ip6_frag_hash[hash]);
+ INIT_LIST_HEAD(&fq->q.lru_list);
+ list_add_tail(&fq->q.lru_list, &ip6_frag_lru_list);
ip6_frag_nqueues++;
write_unlock(&ip6_frag_lock);
return fq;
@@ -382,11 +371,11 @@ ip6_frag_create(__be32 id, struct in6_addr *src, struct in6_addr *dst,
    ipv6_addr_copy(&fq->saddr, src);
    ipv6_addr_copy(&fq->daddr, dst);

- init_timer(&fq->timer);
- fq->timer.function = ip6_frag_expire;
- fq->timer.data = (long) fq;
- spin_lock_init(&fq->lock);
- atomic_set(&fq->refcnt, 1);
+ init_timer(&fq->q.timer);
+ fq->q.timer.function = ip6_frag_expire;
+ fq->q.timer.data = (long) fq;

```

```

+ spin_lock_init(&fq->q.lock);
+ atomic_set(&fq->q.refcnt, 1);

return ip6_frag_intern(fq);

@@ -405,11 +394,11 @@ fq_find(__be32 id, struct in6_addr *src, struct in6_addr *dst,

    read_lock(&ip6_frag_lock);
    hash = ip6qhashfn(id, src, dst);
- hlist_for_each_entry(fq, n, &ip6_frag_hash[hash], list) {
+ hlist_for_each_entry(fq, n, &ip6_frag_hash[hash], q.list) {
    if (fq->id == id &&
        ipv6_addr_equal(src, &fq->saddr) &&
        ipv6_addr_equal(dst, &fq->daddr)) {
- atomic_inc(&fq->refcnt);
+ atomic_inc(&fq->q.refcnt);
    read_unlock(&ip6_frag_lock);
    return fq;
}

@@ -426,7 +415,7 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    struct sk_buff *prev, *next;
    int offset, end;

- if (fq->last_in & COMPLETE)
+ if (fq->q.last_in & COMPLETE)
    goto err;

    offset = ntohs(fhdr->frag_off) & ~0x7;
@@ -454,11 +443,11 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    /* If we already have some bits beyond end
     * or have different end, the segment is corrupted.
     */
- if (end < fq->len ||
-     ((fq->last_in & LAST_IN) && end != fq->len))
+ if (end < fq->q.len ||
+     ((fq->q.last_in & LAST_IN) && end != fq->q.len))
    goto err;
- fq->last_in |= LAST_IN;
- fq->len = end;
+ fq->q.last_in |= LAST_IN;
+ fq->q.len = end;
} else {
    /* Check if the fragment is rounded to 8 bytes.
     * Required by the RFC.
@@ -473,11 +462,11 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    offsetof(struct ipv6hdr, payload_len));
    return;
}

```



```

- if (end > fq->len) {
+ if (end > fq->q.len) {
    /* Some bits beyond end -> corruption. */
- if (fq->last_in & LAST_IN)
+ if (fq->q.last_in & LAST_IN)
    goto err;
- fq->len = end;
+ fq->q.len = end;
}
}

```

```

@@ -496,7 +485,7 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    * this fragment, right?
    */

```

```

prev = NULL;
- for(next = fq->fragments; next != NULL; next = next->next) {
+ for(next = fq->q.fragments; next != NULL; next = next->next) {
    if (FRAG6_CB(next)->offset >= offset)
        break; /* bingo! */
    prev = next;

```

```

@@ -533,7 +522,7 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    if (!pskb_pull(next, i))
        goto err;
    FRAG6_CB(next)->offset += i; /* next fragment */
- fq->meat -= i;
+ fq->q.meat -= i;
    if (next->ip_summed != CHECKSUM_UNNECESSARY)
        next->ip_summed = CHECKSUM_NONE;
    break;

```

```

@@ -548,9 +537,9 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    if (prev)
        prev->next = next;
    else
- fq->fragments = next;
+ fq->q.fragments = next;

```

```

- fq->meat -= free_it->len;
+ fq->q.meat -= free_it->len;
    frag_kfree_skb(free_it, NULL);
}
}

```

```

@@ -562,13 +551,13 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    if (prev)
        prev->next = skb;
    else
- fq->fragments = skb;
+ fq->q.fragments = skb;

```

```

if (skb->dev)
    fq->iif = skb->dev->ifindex;
skb->dev = NULL;
- fq->stamp = skb->tstamp;
- fq->meat += skb->len;
+ fq->q.stamp = skb->tstamp;
+ fq->q.meat += skb->len;
    atomic_add(skb->truesize, &ip6_frag_mem);

/* The first fragment.
@@ -576,10 +565,10 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
    */
    if (offset == 0) {
        fq->nhoffset = nhoff;
- fq->last_in |= FIRST_IN;
+ fq->q.last_in |= FIRST_IN;
    }
    write_lock(&ip6_frag_lock);
- list_move_tail(&fq->lru_list, &ip6_frag_lru_list);
+ list_move_tail(&fq->q.lru_list, &ip6_frag_lru_list);
    write_unlock(&ip6_frag_lock);
    return;

@@ -600,7 +589,7 @@ err:
static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,
    struct net_device *dev)
{
- struct sk_buff *fp, *head = fq->fragments;
+ struct sk_buff *fp, *head = fq->q.fragments;
    int    payload_len;
    unsigned int nhoff;

@@ -611,7 +600,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,

    /* Unfragmented part is taken from the first segment. */
    payload_len = ((head->data - skb_network_header(head)) -
-        sizeof(struct ipv6hdr) + fq->len -
+        sizeof(struct ipv6hdr) + fq->q.len -
        sizeof(struct frag_hdr));
    if (payload_len > IPV6_MAXPLEN)
        goto out_oversize;
@@ -670,7 +659,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,

    head->next = NULL;
    head->dev = dev;
- head->tstamp = fq->stamp;
+ head->tstamp = fq->q.stamp;
    ipv6_hdr(head)->payload_len = htons(payload_len);

```

```
IP6CB(head)->nhoff = nhoff;
```

```
@@ -685,7 +674,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,
    rcu_read_lock();
    IP6_INC_STATS_BH(__in6_dev_get(dev), IPSTATS_MIB_REASMOKS);
    rcu_read_unlock();
- fq->fragments = NULL;
+ fq->q.fragments = NULL;
    return 1;
```

out_oversize:

```
@@ -746,15 +735,15 @@ static int ipv6_frag_rcv(struct sk_buff **skbp)
    ip6_dst_idev(skb->dst)) != NULL) {
    int ret = -1;
```

```
- spin_lock(&fq->lock);
+ spin_lock(&fq->q.lock);
```

```
    ip6_frag_queue(fq, skb, fhdr, IP6CB(skb)->nhoff);
```

```
- if (fq->last_in == (FIRST_IN|LAST_IN) &&
-     fq->meat == fq->len)
+ if (fq->q.last_in == (FIRST_IN|LAST_IN) &&
+     fq->q.meat == fq->q.len)
    ret = ip6_frag_reasm(fq, skbp, dev);
```

```
- spin_unlock(&fq->lock);
+ spin_unlock(&fq->q.lock);
    fq_put(fq, NULL);
    return ret;
}
```

--

1.5.3.4
