Subject: [PATCH 0/9] Consolidate IP fragment management
Posted by Pavel Emelianov on Fri, 12 Oct 2007 12:55:10 GMT
View Forum Message <> Reply to Message

Patrick recently pointed out, that there are three places that
perform IP fragments management. In ipv4, ipv6 and in ip6
conntracks. Looks like these places can be a bit consolidated.

The proposal is to create a common structure inet_frag_queue to
put common fields like list heads, refcounts etc in, and include
it into the specific fragment queues. Then such objects like
hash tables, lists, locks etc are moved to common place (struct
inet_frags). At the end common code is moved to the
net/ipv4/inet_fragment.c.

The inet_ prefix in file names, data structures and functions, and
the code place (net/ipv4) was proposed by Alexey, but the exact
names were selectd by me, so maybe there can be a better ones.

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

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 Emelyanov <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 corrrupted.
    */
-  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,
 #endif
  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("Allready 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(idev, 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
```

Subject: [PATCH 2/9] Collect frag queues management objects together
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:06:13 GMT
View Forum Message <> Reply to Message

There are some objects that are common in all the places
which are used to keep track of frag queues, they are:

 * hash table
 * LRU list
 * rw lock
 * rnd number for hash function
 * the number of queues
 * the amount of memory occupied by queues
 * secret timer

Move all this stuff into one structure (struct inet_frags)
to make it possible use them uniformly in the future. Like
with the previous patch this mostly consists of hunks like

```
-    write_lock(&ipfrag_lock);
+     write_lock(&ip4_frags.lock);
```

To address the issue with exporting the number of queues and
the amount of memory occupied by queues outside the .c file
they are declared in, I introduce a couple of helpers.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index 74e9cb9..d51f238 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -18,4 +18,19 @@ struct inet_frag_queue {
 #define LAST_IN   1
 };

+#define INETFRAGS_HASHSZ  64
```

```
+
+struct inet_frags {
+ struct list_head lru_list;
+ struct hlist_head hash[INETFRAGS_HASHSZ];
+ rwlock_t  lock;
+ u32   rnd;
+ int   nqueues;
+ atomic_t  mem;
+ struct timer_list secret_timer;
+};
+
+void inet_frags_init(struct inet_frags *);
+void inet_frags_fini(struct inet_frags *);
+
 #endif
diff --git a/include/net/ip.h b/include/net/ip.h
index 3af3ed9..a18dcec 100644
--- a/include/net/ip.h
+++ b/include/net/ip.h
@@ -333,8 +333,8 @@ enum ip_defrag_users
 };

 struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user);
-extern int ip_frag_nqueues;
-extern atomic_t ip_frag_mem;
+int ip_frag_mem(void);
+int ip_frag_nqueues(void);

 /*
  * Functions provided by ip_forward.c
diff --git a/include/net/ipv6.h b/include/net/ipv6.h
index 31b3f1b..77cdab3 100644
--- a/include/net/ipv6.h
+++ b/include/net/ipv6.h
@@ -252,8 +252,8 @@ struct ipv6_txoptions *ipv6_fixup_options(struct ipv6_txoptions
*opt_space,

 extern int ipv6_opt_accepted(struct sock *sk, struct sk_buff *skb);

-extern int ip6_frag_nqueues;
-extern atomic_t ip6_frag_mem;
+int ip6_frag_nqueues(void);
+int ip6_frag_mem(void);

 #define IPV6_FRAG_TIMEOUT (60*HZ)  /* 60 seconds */

diff --git a/net/ipv4/Makefile b/net/ipv4/Makefile
index a02c36d..93fe396 100644
```

```
--- a/net/ipv4/Makefile
+++ b/net/ipv4/Makefile
@@ -10,7 +10,8 @@ obj-y     := route.o inetpeer.o protocol.o \
       tcp_minisocks.o tcp_cong.o \
       datagram.o raw.o udp.o udplite.o \
       arp.o icmp.o devinet.o af_inet.o  igmp.o \
-      sysctl_net_ipv4.o fib_frontend.o fib_semantics.o
+      sysctl_net_ipv4.o fib_frontend.o fib_semantics.o \
+      inet_fragment.o

 obj-$(CONFIG_IP_FIB_HASH) += fib_hash.o
 obj-$(CONFIG_IP_FIB_TRIE) += fib_trie.o
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
new file mode 100644
index 0000000..69623ff
--- /dev/null
+++ b/net/ipv4/inet_fragment.c
@@ -0,0 +1,44 @@
+/*
+ * inet fragments management
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License
+ *  as published by the Free Software Foundation; either version
+ *  2 of the License, or (at your option) any later version.
+ *
+ *   Authors: Pavel Emelyanov <xemul@openvz.org>
+ *    Started as consolidation of ipv4/ip_fragment.c,
+ *    ipv6/reassembly. and ipv6 nf conntrack reassembly
+ */
+
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/module.h>
+#include <linux/timer.h>
+#include <linux/mm.h>
+
+#include <net/inet_frag.h>
+
+void inet_frags_init(struct inet_frags *f)
+{
+ int i;
+
+ for (i = 0; i < INETFRAGS_HASHSZ; i++)
+  INIT_HLIST_HEAD(&f->hash[i]);
+
+ INIT_LIST_HEAD(&f->lru_list);
+ rwlock_init(&f->lock);
```

```
+
+ f->rnd = (u32) ((num_physpages ^ (num_physpages>>7)) ^
+       (jiffies ^ (jiffies >> 6)));
+
+ f->nqueues = 0;
+ atomic_set(&f->mem, 0);
+
+}
+EXPORT_SYMBOL(inet_frags_init);
+
+void inet_frags_fini(struct inet_frags *f)
+{
+}
+EXPORT_SYMBOL(inet_frags_fini);
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 3eb1b6d..5e1667e 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -87,39 +87,39 @@ struct ipq {
  struct inet_peer *peer;
 };

-/* Hash table. */
+static struct inet_frags ip4_frags;

-#define IPQ_HASHSZ 64
+int ip_frag_nqueues(void)
+{
+ return ip4_frags.nqueues;
+}

-/* Per-bucket lock is easy to add now. */
-static struct hlist_head ipq_hash[IPQ_HASHSZ];
-static DEFINE_RWLOCK(ipfrag_lock);
-static u32 ipfrag_hash_rnd;
-static LIST_HEAD(ipq_lru_list);
-int ip_frag_nqueues = 0;
+int ip_frag_mem(void)
+{
+ return atomic_read(&ip4_frags.mem);
+}

 static __inline__ void __ipq_unlink(struct ipq *qp)
 {
  hlist_del(&qp->q.list);
  list_del(&qp->q.lru_list);
- ip_frag_nqueues--;
+ ip4_frags.nqueues--;
```

```
 }

 static __inline__ void ipq_unlink(struct ipq *ipq)
 {
- write_lock(&ipfrag_lock);
+ write_lock(&ip4_frags.lock);
  __ipq_unlink(ipq);
- write_unlock(&ipfrag_lock);
+ write_unlock(&ip4_frags.lock);
 }

 static unsigned int ipqhashfn(__be16 id, __be32 saddr, __be32 daddr, u8 prot)
 {
  return jhash_3words((__force u32)id << 16 | prot,
        (__force u32)saddr, (__force u32)daddr,
-       ipfrag_hash_rnd) & (IPQ_HASHSZ - 1);
+       ip4_frags.rnd) & (INETFRAGS_HASHSZ - 1);
 }

-static struct timer_list ipfrag_secret_timer;
 int sysctl_ipfrag_secret_interval __read_mostly = 10 * 60 * HZ;

 static void ipfrag_secret_rebuild(unsigned long dummy)
@@ -127,13 +127,13 @@ static void ipfrag_secret_rebuild(unsigned long dummy)
  unsigned long now = jiffies;
  int i;

- write_lock(&ipfrag_lock);
- get_random_bytes(&ipfrag_hash_rnd, sizeof(u32));
- for (i = 0; i < IPQ_HASHSZ; i++) {
+ write_lock(&ip4_frags.lock);
+ get_random_bytes(&ip4_frags.rnd, sizeof(u32));
+ for (i = 0; i < INETFRAGS_HASHSZ; i++) {
   struct ipq *q;
   struct hlist_node *p, *n;

-  hlist_for_each_entry_safe(q, p, n, &ipq_hash[i], q.list) {
+  hlist_for_each_entry_safe(q, p, n, &ip4_frags.hash[i], q.list) {
    unsigned int hval = ipqhashfn(q->id, q->saddr,
            q->daddr, q->protocol);

@@ -141,23 +141,21 @@ static void ipfrag_secret_rebuild(unsigned long dummy)
     hlist_del(&q->q.list);

     /* Relink to new hash chain. */
-    hlist_add_head(&q->q.list, &ipq_hash[hval]);
+    hlist_add_head(&q->q.list, &ip4_frags.hash[hval]);
    }
```

```
   }
  }
- write_unlock(&ipfrag_lock);
+ write_unlock(&ip4_frags.lock);

- mod_timer(&ipfrag_secret_timer, now + sysctl_ipfrag_secret_interval);
+ mod_timer(&ip4_frags.secret_timer, now + sysctl_ipfrag_secret_interval);
 }

-atomic_t ip_frag_mem = ATOMIC_INIT(0); /* Memory used for fragments */
-
 /* Memory Tracking Functions. */
 static __inline__ void frag_kfree_skb(struct sk_buff *skb, int *work)
 {
  if (work)
   *work -= skb->truesize;
- atomic_sub(skb->truesize, &ip_frag_mem);
+ atomic_sub(skb->truesize, &ip4_frags.mem);
  kfree_skb(skb);
 }

@@ -165,7 +163,7 @@ static __inline__ void frag_free_queue(struct ipq *qp, int *work)
 {
  if (work)
   *work -= sizeof(struct ipq);
- atomic_sub(sizeof(struct ipq), &ip_frag_mem);
+ atomic_sub(sizeof(struct ipq), &ip4_frags.mem);
  kfree(qp);
 }

@@ -175,7 +173,7 @@ static __inline__ struct ipq *frag_alloc_queue(void)

  if (!qp)
   return NULL;
- atomic_add(sizeof(struct ipq), &ip_frag_mem);
+ atomic_add(sizeof(struct ipq), &ip4_frags.mem);
  return qp;
 }

@@ -236,20 +234,20 @@ static void ip_evictor(void)
  struct list_head *tmp;
  int work;

- work = atomic_read(&ip_frag_mem) - sysctl_ipfrag_low_thresh;
+ work = atomic_read(&ip4_frags.mem) - sysctl_ipfrag_low_thresh;
  if (work <= 0)
   return;
```

```
  while (work > 0) {
-  read_lock(&ipfrag_lock);
-  if (list_empty(&ipq_lru_list)) {
-   read_unlock(&ipfrag_lock);
+  read_lock(&ip4_frags.lock);
+  if (list_empty(&ip4_frags.lru_list)) {
+   read_unlock(&ip4_frags.lock);
    return;
   }
-  tmp = ipq_lru_list.next;
+  tmp = ip4_frags.lru_list.next;
   qp = list_entry(tmp, struct ipq, q.lru_list);
   atomic_inc(&qp->q.refcnt);
-  read_unlock(&ipfrag_lock);
+  read_unlock(&ip4_frags.lock);

   spin_lock(&qp->q.lock);
   if (!(qp->q.last_in&COMPLETE))
@@ -301,7 +299,7 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
 #endif
  unsigned int hash;

- write_lock(&ipfrag_lock);
+ write_lock(&ip4_frags.lock);
  hash = ipqhashfn(qp_in->id, qp_in->saddr, qp_in->daddr,
    qp_in->protocol);
 #ifdef CONFIG_SMP
@@ -309,14 +307,14 @@ 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], q.list) {
+ hlist_for_each_entry(qp, n, &ip4_frags.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->q.refcnt);
-   write_unlock(&ipfrag_lock);
+   write_unlock(&ip4_frags.lock);
    qp_in->q.last_in |= COMPLETE;
    ipq_put(qp_in, NULL);
    return qp;
@@ -329,11 +327,11 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
   atomic_inc(&qp->q.refcnt);

  atomic_inc(&qp->q.refcnt);
```

```
- hlist_add_head(&qp->q.list, &ipq_hash[hash]);
+ hlist_add_head(&qp->q.list, &ip4_frags.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);
+ list_add_tail(&qp->q.lru_list, &ip4_frags.lru_list);
+ ip4_frags.nqueues++;
+ write_unlock(&ip4_frags.lock);
  return qp;
 }

@@ -384,20 +382,20 @@ static inline struct ipq *ip_find(struct iphdr *iph, u32 user)
  struct ipq *qp;
  struct hlist_node *n;

- read_lock(&ipfrag_lock);
+ read_lock(&ip4_frags.lock);
  hash = ipqhashfn(id, saddr, daddr, protocol);
- hlist_for_each_entry(qp, n, &ipq_hash[hash], q.list) {
+ hlist_for_each_entry(qp, n, &ip4_frags.hash[hash], q.list) {
   if (qp->id == id  &&
       qp->saddr == saddr &&
       qp->daddr == daddr &&
       qp->protocol == protocol &&
       qp->user == user) {
    atomic_inc(&qp->q.refcnt);
-   read_unlock(&ipfrag_lock);
+   read_unlock(&ip4_frags.lock);
    return qp;
   }
  }
- read_unlock(&ipfrag_lock);
+ read_unlock(&ip4_frags.lock);

  return ip_frag_create(iph, user);
 }
@@ -583,13 +581,13 @@ static void ip_frag_queue(struct ipq *qp, struct sk_buff *skb)
  skb->dev = NULL;
  qp->q.stamp = skb->tstamp;
  qp->q.meat += skb->len;
- atomic_add(skb->truesize, &ip_frag_mem);
+ atomic_add(skb->truesize, &ip4_frags.mem);
  if (offset == 0)
   qp->q.last_in |= FIRST_IN;

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

```
- write_unlock(&ipfrag_lock);
+ write_lock(&ip4_frags.lock);
+ list_move_tail(&qp->q.lru_list, &ip4_frags.lru_list);
+ write_unlock(&ip4_frags.lock);

  return;

@@ -643,12 +641,12 @@ static struct sk_buff *ip_frag_reasm(struct ipq *qp, struct net_device
*dev)
   head->len -= clone->len;
   clone->csum = 0;
   clone->ip_summed = head->ip_summed;
-  atomic_add(clone->truesize, &ip_frag_mem);
+  atomic_add(clone->truesize, &ip4_frags.mem);
  }

  skb_shinfo(head)->frag_list = head->next;
  skb_push(head, head->data - skb_network_header(head));
- atomic_sub(head->truesize, &ip_frag_mem);
+ atomic_sub(head->truesize, &ip4_frags.mem);

  for (fp=head->next; fp; fp = fp->next) {
   head->data_len += fp->len;
@@ -658,7 +656,7 @@ static struct sk_buff *ip_frag_reasm(struct ipq *qp, struct net_device
*dev)
   else if (head->ip_summed == CHECKSUM_COMPLETE)
    head->csum = csum_add(head->csum, fp->csum);
   head->truesize += fp->truesize;
-  atomic_sub(fp->truesize, &ip_frag_mem);
+  atomic_sub(fp->truesize, &ip4_frags.mem);
  }

  head->next = NULL;
@@ -695,7 +693,7 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)
  IP_INC_STATS_BH(IPSTATS_MIB_REASMREQDS);

  /* Start by cleaning up the memory. */
- if (atomic_read(&ip_frag_mem) > sysctl_ipfrag_high_thresh)
+ if (atomic_read(&ip4_frags.mem) > sysctl_ipfrag_high_thresh)
   ip_evictor();

  dev = skb->dev;
@@ -724,13 +722,12 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)

 void __init ipfrag_init(void)
 {
- ipfrag_hash_rnd = (u32) ((num_physpages ^ (num_physpages>>7)) ^
-     (jiffies ^ (jiffies >> 6)));
```

```
+ init_timer(&ip4_frags.secret_timer);
+ ip4_frags.secret_timer.function = ipfrag_secret_rebuild;
+ ip4_frags.secret_timer.expires = jiffies + sysctl_ipfrag_secret_interval;
+ add_timer(&ip4_frags.secret_timer);

- init_timer(&ipfrag_secret_timer);
- ipfrag_secret_timer.function = ipfrag_secret_rebuild;
- ipfrag_secret_timer.expires = jiffies + sysctl_ipfrag_secret_interval;
- add_timer(&ipfrag_secret_timer);
+ inet_frags_init(&ip4_frags);
 }

 EXPORT_SYMBOL(ip_defrag);
diff --git a/net/ipv4/proc.c b/net/ipv4/proc.c
index e5b05b0..fd16cb8 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -70,8 +70,8 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)
  seq_printf(seq, "UDP: inuse %d\n", fold_prot_inuse(&udp_prot));
  seq_printf(seq, "UDPLITE: inuse %d\n", fold_prot_inuse(&udplite_prot));
  seq_printf(seq, "RAW: inuse %d\n", fold_prot_inuse(&raw_prot));
- seq_printf(seq,  "FRAG: inuse %d memory %d\n", ip_frag_nqueues,
-     atomic_read(&ip_frag_mem));
+ seq_printf(seq,  "FRAG: inuse %d memory %d\n",
+   ip_frag_nqueues(), ip_frag_mem());
  return 0;
 }

diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 52e9f6a..eb2ca1b 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -74,28 +74,20 @@ struct nf_ct_frag6_queue
  __u16   nhoffset;
 };

-/* Hash table. */
-
-#define FRAG6Q_HASHSZ 64
-
-static struct hlist_head nf_ct_frag6_hash[FRAG6Q_HASHSZ];
-static DEFINE_RWLOCK(nf_ct_frag6_lock);
-static u32 nf_ct_frag6_hash_rnd;
-static LIST_HEAD(nf_ct_frag6_lru_list);
-int nf_ct_frag6_nqueues = 0;
+static struct inet_frags nf_frags;

 static __inline__ void __fq_unlink(struct nf_ct_frag6_queue *fq)
```

```
 {
  hlist_del(&fq->q.list);
  list_del(&fq->q.lru_list);
- nf_ct_frag6_nqueues--;
+ nf_frags.nqueues--;
 }

 static __inline__ void fq_unlink(struct nf_ct_frag6_queue *fq)
 {
- write_lock(&nf_ct_frag6_lock);
+ write_lock(&nf_frags.lock);
  __fq_unlink(fq);
- write_unlock(&nf_ct_frag6_lock);
+ write_unlock(&nf_frags.lock);
 }

 static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
@@ -109,7 +101,7 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,

  a += JHASH_GOLDEN_RATIO;
  b += JHASH_GOLDEN_RATIO;
- c += nf_ct_frag6_hash_rnd;
+ c += nf_frags.rnd;
  __jhash_mix(a, b, c);

  a += (__force u32)saddr->s6_addr32[3];
@@ -122,10 +114,9 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  c += (__force u32)id;
  __jhash_mix(a, b, c);

- return c & (FRAG6Q_HASHSZ - 1);
+ return c & (INETFRAGS_HASHSZ - 1);
 }

-static struct timer_list nf_ct_frag6_secret_timer;
 int nf_ct_frag6_secret_interval = 10 * 60 * HZ;

 static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
@@ -133,13 +124,13 @@ static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
  unsigned long now = jiffies;
  int i;

- write_lock(&nf_ct_frag6_lock);
- get_random_bytes(&nf_ct_frag6_hash_rnd, sizeof(u32));
- for (i = 0; i < FRAG6Q_HASHSZ; i++) {
+ write_lock(&nf_frags.lock);
+ get_random_bytes(&nf_frags.rnd, sizeof(u32));
+ for (i = 0; i < INETFRAGS_HASHSZ; i++) {
```

```
   struct nf_ct_frag6_queue *q;
   struct hlist_node *p, *n;

-  hlist_for_each_entry_safe(q, p, n, &nf_ct_frag6_hash[i], q.list) {
+  hlist_for_each_entry_safe(q, p, n, &nf_frags.hash[i], q.list) {
    unsigned int hval = ip6qhashfn(q->id,
             &q->saddr,
             &q->daddr);
@@ -147,23 +138,21 @@ static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
     hlist_del(&q->q.list);
     /* Relink to new hash chain. */
     hlist_add_head(&q->q.list,
-            &nf_ct_frag6_hash[hval]);
+            &nf_frags.hash[hval]);
    }
   }
  }
- write_unlock(&nf_ct_frag6_lock);
+ write_unlock(&nf_frags.lock);

- mod_timer(&nf_ct_frag6_secret_timer, now + nf_ct_frag6_secret_interval);
+ mod_timer(&nf_frags.secret_timer, now + nf_ct_frag6_secret_interval);
 }

-atomic_t nf_ct_frag6_mem = ATOMIC_INIT(0);
-
 /* Memory Tracking Functions. */
 static inline void frag_kfree_skb(struct sk_buff *skb, unsigned int *work)
 {
  if (work)
   *work -= skb->truesize;
- atomic_sub(skb->truesize, &nf_ct_frag6_mem);
+ atomic_sub(skb->truesize, &nf_frags.mem);
  if (NFCT_FRAG6_CB(skb)->orig)
   kfree_skb(NFCT_FRAG6_CB(skb)->orig);

@@ -175,7 +164,7 @@ static inline void frag_free_queue(struct nf_ct_frag6_queue *fq,
 {
  if (work)
   *work -= sizeof(struct nf_ct_frag6_queue);
- atomic_sub(sizeof(struct nf_ct_frag6_queue), &nf_ct_frag6_mem);
+ atomic_sub(sizeof(struct nf_ct_frag6_queue), &nf_frags.mem);
  kfree(fq);
 }

@@ -185,7 +174,7 @@ static inline struct nf_ct_frag6_queue *frag_alloc_queue(void)

  if (!fq)
```

```
   return NULL;
- atomic_add(sizeof(struct nf_ct_frag6_queue), &nf_ct_frag6_mem);
+ atomic_add(sizeof(struct nf_ct_frag6_queue), &nf_frags.mem);
  return fq;
 }

@@ -239,22 +228,22 @@ static void nf_ct_frag6_evictor(void)
  struct list_head *tmp;
  unsigned int work;

- work = atomic_read(&nf_ct_frag6_mem);
+ work = atomic_read(&nf_frags.mem);
  if (work <= nf_ct_frag6_low_thresh)
   return;

  work -= nf_ct_frag6_low_thresh;
  while (work > 0) {
-  read_lock(&nf_ct_frag6_lock);
-  if (list_empty(&nf_ct_frag6_lru_list)) {
-   read_unlock(&nf_ct_frag6_lock);
+  read_lock(&nf_frags.lock);
+  if (list_empty(&nf_frags.lru_list)) {
+   read_unlock(&nf_frags.lock);
    return;
   }
-  tmp = nf_ct_frag6_lru_list.next;
+  tmp = nf_frags.lru_list.next;
   BUG_ON(tmp == NULL);
   fq = list_entry(tmp, struct nf_ct_frag6_queue, q.lru_list);
   atomic_inc(&fq->q.refcnt);
-  read_unlock(&nf_ct_frag6_lock);
+  read_unlock(&nf_frags.lock);

   spin_lock(&fq->q.lock);
   if (!(fq->q.last_in&COMPLETE))
@@ -291,14 +280,14 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
  struct hlist_node *n;
 #endif

- write_lock(&nf_ct_frag6_lock);
+ write_lock(&nf_frags.lock);
 #ifdef CONFIG_SMP
- hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], q.list) {
+ hlist_for_each_entry(fq, n, &nf_frags.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->q.refcnt);
```

```
-   write_unlock(&nf_ct_frag6_lock);
+   write_unlock(&nf_frags.lock);
    fq_in->q.last_in |= COMPLETE;
    fq_put(fq_in, NULL);
    return fq;
@@ -311,11 +300,11 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
   atomic_inc(&fq->q.refcnt);

  atomic_inc(&fq->q.refcnt);
- hlist_add_head(&fq->q.list, &nf_ct_frag6_hash[hash]);
+ hlist_add_head(&fq->q.list, &nf_frags.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);
+ list_add_tail(&fq->q.lru_list, &nf_frags.lru_list);
+ nf_frags.nqueues++;
+ write_unlock(&nf_frags.lock);
  return fq;
 }

@@ -353,17 +342,17 @@ fq_find(__be32 id, struct in6_addr *src, struct in6_addr *dst)
  struct hlist_node *n;
  unsigned int hash = ip6qhashfn(id, src, dst);

- read_lock(&nf_ct_frag6_lock);
- hlist_for_each_entry(fq, n, &nf_ct_frag6_hash[hash], q.list) {
+ read_lock(&nf_frags.lock);
+ hlist_for_each_entry(fq, n, &nf_frags.hash[hash], q.list) {
   if (fq->id == id &&
       ipv6_addr_equal(src, &fq->saddr) &&
       ipv6_addr_equal(dst, &fq->daddr)) {
    atomic_inc(&fq->q.refcnt);
-   read_unlock(&nf_ct_frag6_lock);
+   read_unlock(&nf_frags.lock);
    return fq;
   }
  }
- read_unlock(&nf_ct_frag6_lock);
+ read_unlock(&nf_frags.lock);

  return nf_ct_frag6_create(hash, id, src, dst);
 }
@@ -526,7 +515,7 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,
  skb->dev = NULL;
  fq->q.stamp = skb->tstamp;
  fq->q.meat += skb->len;
```

```
- atomic_add(skb->truesize, &nf_ct_frag6_mem);
+ atomic_add(skb->truesize, &nf_frags.mem);

  /* The first fragment.
   * nhoffset is obtained from the first fragment, of course.
@@ -535,9 +524,9 @@ static int nf_ct_frag6_queue(struct nf_ct_frag6_queue *fq, struct sk_buff
*skb,
   fq->nhoffset = nhoff;
   fq->q.last_in |= FIRST_IN;
  }
- write_lock(&nf_ct_frag6_lock);
- list_move_tail(&fq->q.lru_list, &nf_ct_frag6_lru_list);
- write_unlock(&nf_ct_frag6_lock);
+ write_lock(&nf_frags.lock);
+ list_move_tail(&fq->q.lru_list, &nf_frags.lru_list);
+ write_unlock(&nf_frags.lock);
  return 0;

 err:
@@ -603,7 +592,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)
   clone->ip_summed = head->ip_summed;

   NFCT_FRAG6_CB(clone)->orig = NULL;
-  atomic_add(clone->truesize, &nf_ct_frag6_mem);
+  atomic_add(clone->truesize, &nf_frags.mem);
  }

  /* We have to remove fragment header from datagram and to relocate
@@ -617,7 +606,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)
  skb_shinfo(head)->frag_list = head->next;
  skb_reset_transport_header(head);
  skb_push(head, head->data - skb_network_header(head));
- atomic_sub(head->truesize, &nf_ct_frag6_mem);
+ atomic_sub(head->truesize, &nf_frags.mem);

  for (fp=head->next; fp; fp = fp->next) {
   head->data_len += fp->len;
@@ -627,7 +616,7 @@ nf_ct_frag6_reasm(struct nf_ct_frag6_queue *fq, struct net_device *dev)
   else if (head->ip_summed == CHECKSUM_COMPLETE)
    head->csum = csum_add(head->csum, fp->csum);
   head->truesize += fp->truesize;
-  atomic_sub(fp->truesize, &nf_ct_frag6_mem);
+  atomic_sub(fp->truesize, &nf_frags.mem);
  }

  head->next = NULL;
@@ -777,7 +766,7 @@ struct sk_buff *nf_ct_frag6_gather(struct sk_buff *skb)
   goto ret_orig;
```

```
  }

- if (atomic_read(&nf_ct_frag6_mem) > nf_ct_frag6_high_thresh)
+ if (atomic_read(&nf_frags.mem) > nf_ct_frag6_high_thresh)
    nf_ct_frag6_evictor();

  fq = fq_find(fhdr->identification, &hdr->saddr, &hdr->daddr);
@@ -848,20 +837,21 @@ int nf_ct_frag6_kfree_frags(struct sk_buff *skb)

 int nf_ct_frag6_init(void)
 {
- nf_ct_frag6_hash_rnd = (u32) ((num_physpages ^ (num_physpages>>7)) ^
-       (jiffies ^ (jiffies >> 6)));
-
- setup_timer(&nf_ct_frag6_secret_timer, nf_ct_frag6_secret_rebuild, 0);
- nf_ct_frag6_secret_timer.expires = jiffies
+ setup_timer(&nf_frags.secret_timer, nf_ct_frag6_secret_rebuild, 0);
+ nf_frags.secret_timer.expires = jiffies
         + nf_ct_frag6_secret_interval;
- add_timer(&nf_ct_frag6_secret_timer);
+ add_timer(&nf_frags.secret_timer);
+
+ inet_frags_init(&nf_frags);

  return 0;
 }

 void nf_ct_frag6_cleanup(void)
 {
- del_timer(&nf_ct_frag6_secret_timer);
+ inet_frags_fini(&nf_frags);
+
+ del_timer(&nf_frags.secret_timer);
  nf_ct_frag6_low_thresh = 0;
  nf_ct_frag6_evictor();
 }
diff --git a/net/ipv6/proc.c b/net/ipv6/proc.c
index db94501..be526ad 100644
--- a/net/ipv6/proc.c
+++ b/net/ipv6/proc.c
@@ -54,7 +54,7 @@ static int sockstat6_seq_show(struct seq_file *seq, void *v)
  seq_printf(seq, "RAW6: inuse %d\n",
         fold_prot_inuse(&rawv6_prot));
  seq_printf(seq, "FRAG6: inuse %d memory %d\n",
-        ip6_frag_nqueues, atomic_read(&ip6_frag_mem));
+        ip6_frag_nqueues(), ip6_frag_mem());
  return 0;
 }
```

```
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index f48ecc6..7b6315f 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -86,28 +86,30 @@ struct frag_queue
  __u16   nhoffset;
 };

-/* Hash table. */
+static struct inet_frags ip6_frags;

-#define IP6Q_HASHSZ 64
+int ip6_frag_nqueues(void)
+{
+ return ip6_frags.nqueues;
+}

-static struct hlist_head ip6_frag_hash[IP6Q_HASHSZ];
-static DEFINE_RWLOCK(ip6_frag_lock);
-static u32 ip6_frag_hash_rnd;
-static LIST_HEAD(ip6_frag_lru_list);
-int ip6_frag_nqueues = 0;
+int ip6_frag_mem(void)
+{
+ return atomic_read(&ip6_frags.mem);
+}

 static __inline__ void __fq_unlink(struct frag_queue *fq)
 {
  hlist_del(&fq->q.list);
  list_del(&fq->q.lru_list);
- ip6_frag_nqueues--;
+ ip6_frags.nqueues--;
 }

 static __inline__ void fq_unlink(struct frag_queue *fq)
 {
- write_lock(&ip6_frag_lock);
+ write_lock(&ip6_frags.lock);
  __fq_unlink(fq);
- write_unlock(&ip6_frag_lock);
+ write_unlock(&ip6_frags.lock);
 }

 /*
@@ -125,7 +127,7 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
```

```
  a += JHASH_GOLDEN_RATIO;
  b += JHASH_GOLDEN_RATIO;
- c += ip6_frag_hash_rnd;
+ c += ip6_frags.rnd;
  __jhash_mix(a, b, c);

  a += (__force u32)saddr->s6_addr32[3];
@@ -138,10 +140,9 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  c += (__force u32)id;
  __jhash_mix(a, b, c);

- return c & (IP6Q_HASHSZ - 1);
+ return c & (INETFRAGS_HASHSZ - 1);
 }

-static struct timer_list ip6_frag_secret_timer;
 int sysctl_ip6frag_secret_interval __read_mostly = 10 * 60 * HZ;

 static void ip6_frag_secret_rebuild(unsigned long dummy)
@@ -149,13 +150,13 @@ static void ip6_frag_secret_rebuild(unsigned long dummy)
  unsigned long now = jiffies;
  int i;

- write_lock(&ip6_frag_lock);
- get_random_bytes(&ip6_frag_hash_rnd, sizeof(u32));
- for (i = 0; i < IP6Q_HASHSZ; i++) {
+ write_lock(&ip6_frags.lock);
+ get_random_bytes(&ip6_frags.rnd, sizeof(u32));
+ for (i = 0; i < INETFRAGS_HASHSZ; i++) {
   struct frag_queue *q;
   struct hlist_node *p, *n;

-  hlist_for_each_entry_safe(q, p, n, &ip6_frag_hash[i], q.list) {
+  hlist_for_each_entry_safe(q, p, n, &ip6_frags.hash[i], q.list) {
    unsigned int hval = ip6qhashfn(q->id,
             &q->saddr,
             &q->daddr);
@@ -165,24 +166,22 @@ static void ip6_frag_secret_rebuild(unsigned long dummy)
     /* Relink to new hash chain. */
     hlist_add_head(&q->q.list,
-           &ip6_frag_hash[hval]);
+           &ip6_frags.hash[hval]);

   }
  }
 }
- write_unlock(&ip6_frag_lock);
```

```
+ write_unlock(&ip6_frags.lock);

- mod_timer(&ip6_frag_secret_timer, now + sysctl_ip6frag_secret_interval);
+ mod_timer(&ip6_frags.secret_timer, now + sysctl_ip6frag_secret_interval);
 }

-atomic_t ip6_frag_mem = ATOMIC_INIT(0);
-
 /* Memory Tracking Functions. */
 static inline void frag_kfree_skb(struct sk_buff *skb, int *work)
 {
  if (work)
   *work -= skb->truesize;
- atomic_sub(skb->truesize, &ip6_frag_mem);
+ atomic_sub(skb->truesize, &ip6_frags.mem);
  kfree_skb(skb);
 }

@@ -190,7 +189,7 @@ static inline void frag_free_queue(struct frag_queue *fq, int *work)
 {
  if (work)
   *work -= sizeof(struct frag_queue);
- atomic_sub(sizeof(struct frag_queue), &ip6_frag_mem);
+ atomic_sub(sizeof(struct frag_queue), &ip6_frags.mem);
  kfree(fq);
 }

@@ -200,7 +199,7 @@ static inline struct frag_queue *frag_alloc_queue(void)

  if(!fq)
   return NULL;
- atomic_add(sizeof(struct frag_queue), &ip6_frag_mem);
+ atomic_add(sizeof(struct frag_queue), &ip6_frags.mem);
  return fq;
 }

@@ -253,20 +252,20 @@ static void ip6_evictor(struct inet6_dev *idev)
  struct list_head *tmp;
  int work;

- work = atomic_read(&ip6_frag_mem) - sysctl_ip6frag_low_thresh;
+ work = atomic_read(&ip6_frags.mem) - sysctl_ip6frag_low_thresh;
  if (work <= 0)
   return;

  while(work > 0) {
-  read_lock(&ip6_frag_lock);
-  if (list_empty(&ip6_frag_lru_list)) {
```

```
-   read_unlock(&ip6_frag_lock);
+  read_lock(&ip6_frags.lock);
+  if (list_empty(&ip6_frags.lru_list)) {
+   read_unlock(&ip6_frags.lock);
    return;
   }
-  tmp = ip6_frag_lru_list.next;
+  tmp = ip6_frags.lru_list.next;
   fq = list_entry(tmp, struct frag_queue, q.lru_list);
   atomic_inc(&fq->q.refcnt);
-  read_unlock(&ip6_frag_lock);
+  read_unlock(&ip6_frags.lock);

   spin_lock(&fq->q.lock);
   if (!(fq->q.last_in&COMPLETE))
@@ -328,15 +327,15 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)
  struct hlist_node *n;
 #endif

- write_lock(&ip6_frag_lock);
+ write_lock(&ip6_frags.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], q.list) {
+ hlist_for_each_entry(fq, n, &ip6_frags.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->q.refcnt);
-   write_unlock(&ip6_frag_lock);
+   write_unlock(&ip6_frags.lock);
    fq_in->q.last_in |= COMPLETE;
    fq_put(fq_in, NULL);
    return fq;
@@ -349,11 +348,11 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)
   atomic_inc(&fq->q.refcnt);

  atomic_inc(&fq->q.refcnt);
- hlist_add_head(&fq->q.list, &ip6_frag_hash[hash]);
+ hlist_add_head(&fq->q.list, &ip6_frags.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);
+ list_add_tail(&fq->q.lru_list, &ip6_frags.lru_list);
+ ip6_frags.nqueues++;
+ write_unlock(&ip6_frags.lock);
  return fq;
```

```
 }

@@ -392,18 +391,18 @@ fq_find(__be32 id, struct in6_addr *src, struct in6_addr *dst,
  struct hlist_node *n;
  unsigned int hash;

- read_lock(&ip6_frag_lock);
+ read_lock(&ip6_frags.lock);
  hash = ip6qhashfn(id, src, dst);
- hlist_for_each_entry(fq, n, &ip6_frag_hash[hash], q.list) {
+ hlist_for_each_entry(fq, n, &ip6_frags.hash[hash], q.list) {
   if (fq->id == id &&
       ipv6_addr_equal(src, &fq->saddr) &&
       ipv6_addr_equal(dst, &fq->daddr)) {
    atomic_inc(&fq->q.refcnt);
-   read_unlock(&ip6_frag_lock);
+   read_unlock(&ip6_frags.lock);
    return fq;
   }
  }
- read_unlock(&ip6_frag_lock);
+ read_unlock(&ip6_frags.lock);

  return ip6_frag_create(id, src, dst, idev);
 }
@@ -558,7 +557,7 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
  skb->dev = NULL;
  fq->q.stamp = skb->tstamp;
  fq->q.meat += skb->len;
- atomic_add(skb->truesize, &ip6_frag_mem);
+ atomic_add(skb->truesize, &ip6_frags.mem);

  /* The first fragment.
   * nhoffset is obtained from the first fragment, of course.
@@ -567,9 +566,9 @@ static void ip6_frag_queue(struct frag_queue *fq, struct sk_buff *skb,
   fq->nhoffset = nhoff;
   fq->q.last_in |= FIRST_IN;
  }
- write_lock(&ip6_frag_lock);
- list_move_tail(&fq->q.lru_list, &ip6_frag_lru_list);
- write_unlock(&ip6_frag_lock);
+ write_lock(&ip6_frags.lock);
+ list_move_tail(&fq->q.lru_list, &ip6_frags.lru_list);
+ write_unlock(&ip6_frags.lock);
  return;

 err:
@@ -629,7 +628,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,
```

```
   head->len -= clone->len;
   clone->csum = 0;
   clone->ip_summed = head->ip_summed;
-  atomic_add(clone->truesize, &ip6_frag_mem);
+  atomic_add(clone->truesize, &ip6_frags.mem);
  }

  /* We have to remove fragment header from datagram and to relocate
@@ -644,7 +643,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,
  skb_shinfo(head)->frag_list = head->next;
  skb_reset_transport_header(head);
  skb_push(head, head->data - skb_network_header(head));
- atomic_sub(head->truesize, &ip6_frag_mem);
+ atomic_sub(head->truesize, &ip6_frags.mem);

  for (fp=head->next; fp; fp = fp->next) {
   head->data_len += fp->len;
@@ -654,7 +653,7 @@ static int ip6_frag_reasm(struct frag_queue *fq, struct sk_buff **skb_in,
   else if (head->ip_summed == CHECKSUM_COMPLETE)
    head->csum = csum_add(head->csum, fp->csum);
   head->truesize += fp->truesize;
-  atomic_sub(fp->truesize, &ip6_frag_mem);
+  atomic_sub(fp->truesize, &ip6_frags.mem);
  }

  head->next = NULL;
@@ -728,7 +727,7 @@ static int ipv6_frag_rcv(struct sk_buff **skbp)
   return 1;
  }

- if (atomic_read(&ip6_frag_mem) > sysctl_ip6frag_high_thresh)
+ if (atomic_read(&ip6_frags.mem) > sysctl_ip6frag_high_thresh)
   ip6_evictor(ip6_dst_idev(skb->dst));

  if ((fq = fq_find(fhdr->identification, &hdr->saddr, &hdr->daddr,
@@ -764,11 +763,10 @@ void __init ipv6_frag_init(void)
  if (inet6_add_protocol(&frag_protocol, IPPROTO_FRAGMENT) < 0)
   printk(KERN_ERR "ipv6_frag_init: Could not register protocol\n");

- ip6_frag_hash_rnd = (u32) ((num_physpages ^ (num_physpages>>7)) ^
-       (jiffies ^ (jiffies >> 6)));
+ init_timer(&ip6_frags.secret_timer);
+ ip6_frags.secret_timer.function = ip6_frag_secret_rebuild;
+ ip6_frags.secret_timer.expires = jiffies + sysctl_ip6frag_secret_interval;
+ add_timer(&ip6_frags.secret_timer);

- init_timer(&ip6_frag_secret_timer);
- ip6_frag_secret_timer.function = ip6_frag_secret_rebuild;
```

```
- ip6_frag_secret_timer.expires = jiffies + sysctl_ip6frag_secret_interval;
- add_timer(&ip6_frag_secret_timer);
+ inet_frags_init(&ip6_frags);
 }
--
1.5.3.4
```

Subject: [PATCH 3/9] Collect common sysctl variables together
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:10:05 GMT
View Forum Message <> Reply to Message

Some sysctl variables are used to tune the frag queues
management and it will be useful to work with them in
a common way in the future, so move them into one
structure, moreover they are the same for all the frag
management codes.

I don't place them in the existing inet_frags object,
introduced in the previous patch for two reasons:

 1. to keep them in the __read_mostly section;
 2. not to export the whole inet_frags objects outside.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index d51f238..ada03ba 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -20,6 +20,13 @@ struct inet_frag_queue {

 #define INETFRAGS_HASHSZ  64

+struct inet_frags_ctl {
+ int high_thresh;
+ int low_thresh;
+ int timeout;
+ int secret_interval;
+};
+
 struct inet_frags {
  struct list_head lru_list;
  struct hlist_head hash[INETFRAGS_HASHSZ];
@@ -28,6 +35,7 @@ struct inet_frags {
  int   nqueues;
```

```
  atomic_t  mem;
  struct timer_list secret_timer;
+ struct inet_frags_ctl *ctl;
 };

 void inet_frags_init(struct inet_frags *);
diff --git a/include/net/ip.h b/include/net/ip.h
index a18dcec..e7b0feb 100644
--- a/include/net/ip.h
+++ b/include/net/ip.h
@@ -177,10 +177,8 @@ extern int sysctl_ip_default_ttl;
 extern int sysctl_ip_nonlocal_bind;

 /* From ip_fragment.c */
-extern int sysctl_ipfrag_high_thresh;
-extern int sysctl_ipfrag_low_thresh;
-extern int sysctl_ipfrag_time;
-extern int sysctl_ipfrag_secret_interval;
+struct inet_frags_ctl;
+extern struct inet_frags_ctl ip4_frags_ctl;
 extern int sysctl_ipfrag_max_dist;

 /* From inetpeer.c */
diff --git a/include/net/ipv6.h b/include/net/ipv6.h
index 77cdab3..b29d76c 100644
--- a/include/net/ipv6.h
+++ b/include/net/ipv6.h
@@ -565,10 +565,8 @@ extern int inet6_hash_connect(struct inet_timewait_death_row
*death_row,
 /*
  * reassembly.c
  */
-extern int sysctl_ip6frag_high_thresh;
-extern int sysctl_ip6frag_low_thresh;
-extern int sysctl_ip6frag_time;
-extern int sysctl_ip6frag_secret_interval;
+struct inet_frags_ctl;
+extern struct inet_frags_ctl ip6_frags_ctl;

 extern const struct proto_ops inet6_stream_ops;
 extern const struct proto_ops inet6_dgram_ops;
diff --git a/include/net/netfilter/ipv6/nf_conntrack_ipv6.h
b/include/net/netfilter/ipv6/nf_conntrack_ipv6.h
index 070d12c..f703533 100644
--- a/include/net/netfilter/ipv6/nf_conntrack_ipv6.h
+++ b/include/net/netfilter/ipv6/nf_conntrack_ipv6.h
@@ -15,8 +15,7 @@ extern void nf_ct_frag6_output(unsigned int hooknum, struct sk_buff *skb,
          struct net_device *out,
```

```
          int (*okfn)(struct sk_buff *));

-extern unsigned int nf_ct_frag6_timeout;
-extern unsigned int nf_ct_frag6_low_thresh;
-extern unsigned int nf_ct_frag6_high_thresh;
+struct inet_frags_ctl;
+extern struct inet_frags_ctl nf_frags_ctl;

 #endif /* _NF_CONNTRACK_IPV6_H*/
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 5e1667e..61035a8 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -50,21 +50,8 @@
  * as well. Or notify me, at least. --ANK
  */

-/* Fragment cache limits. We will commit 256K at one time. Should we
- * cross that limit we will prune down to 192K. This should cope with
- * even the most extreme cases without allowing an attacker to measurably
- * harm machine performance.
- */
-int sysctl_ipfrag_high_thresh __read_mostly = 256*1024;
-int sysctl_ipfrag_low_thresh __read_mostly = 192*1024;
-
 int sysctl_ipfrag_max_dist __read_mostly = 64;

-/* Important NOTE! Fragment queue must be destroyed before MSL expires.
- * RFC791 is wrong proposing to prolongate timer each fragment arrival by TTL.
- */
-int sysctl_ipfrag_time __read_mostly = IP_FRAG_TIME;
-
 struct ipfrag_skb_cb
 {
  struct inet_skb_parm h;
@@ -87,6 +74,25 @@ struct ipq {
  struct inet_peer *peer;
 };

+struct inet_frags_ctl ip4_frags_ctl __read_mostly = {
+ /*
+  * Fragment cache limits. We will commit 256K at one time. Should we
+  * cross that limit we will prune down to 192K. This should cope with
+  * even the most extreme cases without allowing an attacker to
+  * measurably harm machine performance.
+  */
+ .high_thresh  = 256 * 1024,
+ .low_thresh  = 192 * 1024,
```

```
+
+ /*
+  * Important NOTE! Fragment queue must be destroyed before MSL expires.
+  * RFC791 is wrong proposing to prolongate timer each fragment arrival
+  * by TTL.
+  */
+ .timeout  = IP_FRAG_TIME,
+ .secret_interval = 10 * 60 * HZ,
+};
+
 static struct inet_frags ip4_frags;

 int ip_frag_nqueues(void)
@@ -120,8 +126,6 @@ static unsigned int ipqhashfn(__be16 id, __be32 saddr, __be32 daddr,
u8 prot)
        ip4_frags.rnd) & (INETFRAGS_HASHSZ - 1);
 }

-int sysctl_ipfrag_secret_interval __read_mostly = 10 * 60 * HZ;
-
 static void ipfrag_secret_rebuild(unsigned long dummy)
 {
  unsigned long now = jiffies;
@@ -147,7 +151,7 @@ static void ipfrag_secret_rebuild(unsigned long dummy)
  }
  write_unlock(&ip4_frags.lock);

- mod_timer(&ip4_frags.secret_timer, now + sysctl_ipfrag_secret_interval);
+ mod_timer(&ip4_frags.secret_timer, now + ip4_frags_ctl.secret_interval);
 }

 /* Memory Tracking Functions. */
@@ -234,7 +238,7 @@ static void ip_evictor(void)
  struct list_head *tmp;
  int work;

- work = atomic_read(&ip4_frags.mem) - sysctl_ipfrag_low_thresh;
+ work = atomic_read(&ip4_frags.mem) - ip4_frags_ctl.low_thresh;
  if (work <= 0)
   return;

@@ -323,7 +327,7 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
 #endif
  qp = qp_in;

- if (!mod_timer(&qp->q.timer, jiffies + sysctl_ipfrag_time))
+ if (!mod_timer(&qp->q.timer, jiffies + ip4_frags_ctl.timeout))
   atomic_inc(&qp->q.refcnt);
```

```
  atomic_inc(&qp->q.refcnt);
@@ -429,7 +433,7 @@ static int ip_frag_reinit(struct ipq *qp)
 {
  struct sk_buff *fp;

- if (!mod_timer(&qp->q.timer, jiffies + sysctl_ipfrag_time)) {
+ if (!mod_timer(&qp->q.timer, jiffies + ip4_frags_ctl.timeout)) {
    atomic_inc(&qp->q.refcnt);
    return -ETIMEDOUT;
  }
@@ -693,7 +697,7 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)
  IP_INC_STATS_BH(IPSTATS_MIB_REASMREQDS);

  /* Start by cleaning up the memory. */
- if (atomic_read(&ip4_frags.mem) > sysctl_ipfrag_high_thresh)
+ if (atomic_read(&ip4_frags.mem) > ip4_frags_ctl.high_thresh)
    ip_evictor();

  dev = skb->dev;
@@ -724,9 +728,10 @@ void __init ipfrag_init(void)
 {
  init_timer(&ip4_frags.secret_timer);
  ip4_frags.secret_timer.function = ipfrag_secret_rebuild;
- ip4_frags.secret_timer.expires = jiffies + sysctl_ipfrag_secret_interval;
+ ip4_frags.secret_timer.expires = jiffies + ip4_frags_ctl.secret_interval;
  add_timer(&ip4_frags.secret_timer);

+ ip4_frags.ctl = &ip4_frags_ctl;
  inet_frags_init(&ip4_frags);
 }

diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index eb286ab..c98ef16 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -19,6 +19,7 @@
 #include <net/route.h>
 #include <net/tcp.h>
 #include <net/cipso_ipv4.h>
+#include <net/inet_frag.h>

 /* From af_inet.c */
 extern int sysctl_ip_nonlocal_bind;
@@ -357,7 +358,7 @@ ctl_table ipv4_table[] = {
  {
    .ctl_name = NET_IPV4_IPFRAG_HIGH_THRESH,
    .procname = "ipfrag_high_thresh",
```

```
-  .data  = &sysctl_ipfrag_high_thresh,
+  .data  = &ip4_frags_ctl.high_thresh,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec
@@ -365,7 +366,7 @@ ctl_table ipv4_table[] = {
  {
   .ctl_name = NET_IPV4_IPFRAG_LOW_THRESH,
   .procname = "ipfrag_low_thresh",
-  .data  = &sysctl_ipfrag_low_thresh,
+  .data  = &ip4_frags_ctl.low_thresh,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec
@@ -381,7 +382,7 @@ ctl_table ipv4_table[] = {
  {
   .ctl_name = NET_IPV4_IPFRAG_TIME,
   .procname = "ipfrag_time",
-  .data  = &sysctl_ipfrag_time,
+  .data  = &ip4_frags_ctl.timeout,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec_jiffies,
@@ -732,7 +733,7 @@ ctl_table ipv4_table[] = {
  {
   .ctl_name = NET_IPV4_IPFRAG_SECRET_INTERVAL,
   .procname = "ipfrag_secret_interval",
-  .data  = &sysctl_ipfrag_secret_interval,
+  .data  = &ip4_frags_ctl.secret_interval,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec_jiffies,
diff --git a/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
b/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
index 37a3db9..572c0bc 100644
--- a/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
+++ b/net/ipv6/netfilter/nf_conntrack_l3proto_ipv6.c
@@ -18,6 +18,7 @@
 #include <linux/icmp.h>
 #include <linux/sysctl.h>
 #include <net/ipv6.h>
+#include <net/inet_frag.h>

 #include <linux/netfilter_ipv6.h>
 #include <net/netfilter/nf_conntrack.h>
@@ -307,7 +308,7 @@ static ctl_table nf_ct_ipv6_sysctl_table[] = {
  {
   .ctl_name = NET_NF_CONNTRACK_FRAG6_TIMEOUT,
```

```
   .procname = "nf_conntrack_frag6_timeout",
-  .data  = &nf_ct_frag6_timeout,
+  .data  = &nf_frags_ctl.timeout,
   .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec_jiffies,
@@ -315,7 +316,7 @@ static ctl_table nf_ct_ipv6_sysctl_table[] = {
  {
   .ctl_name = NET_NF_CONNTRACK_FRAG6_LOW_THRESH,
   .procname = "nf_conntrack_frag6_low_thresh",
-  .data  = &nf_ct_frag6_low_thresh,
+  .data  = &nf_frags_ctl.low_thresh,
   .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec,
@@ -323,7 +324,7 @@ static ctl_table nf_ct_ipv6_sysctl_table[] = {
  {
   .ctl_name = NET_NF_CONNTRACK_FRAG6_HIGH_THRESH,
   .procname = "nf_conntrack_frag6_high_thresh",
-  .data  = &nf_ct_frag6_high_thresh,
+  .data  = &nf_frags_ctl.high_thresh,
   .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec,
diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index eb2ca1b..966a888 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -49,10 +49,6 @@
 #define NF_CT_FRAG6_LOW_THRESH 196608  /* == 192*1024 */
 #define NF_CT_FRAG6_TIMEOUT IPV6_FRAG_TIMEOUT

-unsigned int nf_ct_frag6_high_thresh __read_mostly = 256*1024;
-unsigned int nf_ct_frag6_low_thresh __read_mostly = 192*1024;
-unsigned long nf_ct_frag6_timeout __read_mostly = IPV6_FRAG_TIMEOUT;
-
 struct nf_ct_frag6_skb_cb
 {
  struct inet6_skb_parm h;
@@ -74,6 +70,13 @@ struct nf_ct_frag6_queue
  __u16   nhoffset;
 };

+struct inet_frags_ctl nf_frags_ctl __read_mostly = {
+ .high_thresh  = 256 * 1024,
+ .low_thresh  = 192 * 1024,
+ .timeout  = IPV6_FRAG_TIMEOUT,
+ .secret_interval = 10 * 60 * HZ,
```

```
+};
+
 static struct inet_frags nf_frags;

 static __inline__ void __fq_unlink(struct nf_ct_frag6_queue *fq)
@@ -117,8 +120,6 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  return c & (INETFRAGS_HASHSZ - 1);
 }

-int nf_ct_frag6_secret_interval = 10 * 60 * HZ;
-
 static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
 {
  unsigned long now = jiffies;
@@ -144,7 +145,7 @@ static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
  }
  write_unlock(&nf_frags.lock);

- mod_timer(&nf_frags.secret_timer, now + nf_ct_frag6_secret_interval);
+ mod_timer(&nf_frags.secret_timer, now + nf_frags_ctl.secret_interval);
 }

 /* Memory Tracking Functions. */
@@ -229,10 +230,10 @@ static void nf_ct_frag6_evictor(void)
  unsigned int work;

  work = atomic_read(&nf_frags.mem);
- if (work <= nf_ct_frag6_low_thresh)
+ if (work <= nf_frags_ctl.low_thresh)
   return;

- work -= nf_ct_frag6_low_thresh;
+ work -= nf_frags_ctl.low_thresh;
  while (work > 0) {
   read_lock(&nf_frags.lock);
   if (list_empty(&nf_frags.lru_list)) {
@@ -296,7 +297,7 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
 #endif
  fq = fq_in;

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

  atomic_inc(&fq->q.refcnt);
@@ -766,7 +767,7 @@ struct sk_buff *nf_ct_frag6_gather(struct sk_buff *skb)
   goto ret_orig;
  }
```

```
- if (atomic_read(&nf_frags.mem) > nf_ct_frag6_high_thresh)
+ if (atomic_read(&nf_frags.mem) > nf_frags_ctl.high_thresh)
   nf_ct_frag6_evictor();

  fq = fq_find(fhdr->identification, &hdr->saddr, &hdr->daddr);
@@ -838,10 +839,10 @@ int nf_ct_frag6_kfree_frags(struct sk_buff *skb)
 int nf_ct_frag6_init(void)
 {
  setup_timer(&nf_frags.secret_timer, nf_ct_frag6_secret_rebuild, 0);
- nf_frags.secret_timer.expires = jiffies
-         + nf_ct_frag6_secret_interval;
+ nf_frags.secret_timer.expires = jiffies + nf_frags_ctl.secret_interval;
  add_timer(&nf_frags.secret_timer);

+ nf_frags.ctl = &nf_frags_ctl;
  inet_frags_init(&nf_frags);

  return 0;
@@ -852,6 +853,6 @@ void nf_ct_frag6_cleanup(void)
  inet_frags_fini(&nf_frags);

  del_timer(&nf_frags.secret_timer);
- nf_ct_frag6_low_thresh = 0;
+ nf_frags_ctl.low_thresh = 0;
  nf_ct_frag6_evictor();
 }
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 7b6315f..f0e22be 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -55,11 +55,6 @@
 #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;
-
-int sysctl_ip6frag_time __read_mostly = IPV6_FRAG_TIMEOUT;
-
 struct ip6frag_skb_cb
 {
  struct inet6_skb_parm h;
@@ -86,6 +81,13 @@ struct frag_queue
  __u16   nhoffset;
 };

+struct inet_frags_ctl ip6_frags_ctl __read_mostly = {
```

```
+ .high_thresh   = 256 * 1024,
+ .low_thresh  = 192 * 1024,
+ .timeout  = IPV6_FRAG_TIMEOUT,
+ .secret_interval = 10 * 60 * HZ,
+};
+
 static struct inet_frags ip6_frags;

 int ip6_frag_nqueues(void)
@@ -143,8 +145,6 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  return c & (INETFRAGS_HASHSZ - 1);
 }

-int sysctl_ip6frag_secret_interval __read_mostly = 10 * 60 * HZ;
-
 static void ip6_frag_secret_rebuild(unsigned long dummy)
 {
  unsigned long now = jiffies;
@@ -173,7 +173,7 @@ static void ip6_frag_secret_rebuild(unsigned long dummy)
  }
  write_unlock(&ip6_frags.lock);

- mod_timer(&ip6_frags.secret_timer, now + sysctl_ip6frag_secret_interval);
+ mod_timer(&ip6_frags.secret_timer, now + ip6_frags_ctl.secret_interval);
 }

 /* Memory Tracking Functions. */
@@ -252,7 +252,7 @@ static void ip6_evictor(struct inet6_dev *idev)
  struct list_head *tmp;
  int work;

- work = atomic_read(&ip6_frags.mem) - sysctl_ip6frag_low_thresh;
+ work = atomic_read(&ip6_frags.mem) - ip6_frags_ctl.low_thresh;
  if (work <= 0)
   return;

@@ -344,7 +344,7 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)
 #endif
  fq = fq_in;

- if (!mod_timer(&fq->q.timer, jiffies + sysctl_ip6frag_time))
+ if (!mod_timer(&fq->q.timer, jiffies + ip6_frags_ctl.timeout))
   atomic_inc(&fq->q.refcnt);

  atomic_inc(&fq->q.refcnt);
@@ -727,7 +727,7 @@ static int ipv6_frag_rcv(struct sk_buff **skbp)
   return 1;
  }
```

```
- if (atomic_read(&ip6_frags.mem) > sysctl_ip6frag_high_thresh)
+ if (atomic_read(&ip6_frags.mem) > ip6_frags_ctl.high_thresh)
   ip6_evictor(ip6_dst_idev(skb->dst));

  if ((fq = fq_find(fhdr->identification, &hdr->saddr, &hdr->daddr,
@@ -765,8 +765,9 @@ void __init ipv6_frag_init(void)

  init_timer(&ip6_frags.secret_timer);
  ip6_frags.secret_timer.function = ip6_frag_secret_rebuild;
- ip6_frags.secret_timer.expires = jiffies + sysctl_ip6frag_secret_interval;
+ ip6_frags.secret_timer.expires = jiffies + ip6_frags_ctl.secret_interval;
  add_timer(&ip6_frags.secret_timer);

+ ip6_frags.ctl = &ip6_frags_ctl;
  inet_frags_init(&ip6_frags);
 }
diff --git a/net/ipv6/sysctl_net_ipv6.c b/net/ipv6/sysctl_net_ipv6.c
index 3fb4427..97c425f 100644
--- a/net/ipv6/sysctl_net_ipv6.c
+++ b/net/ipv6/sysctl_net_ipv6.c
@@ -12,6 +12,7 @@
 #include <net/ndisc.h>
 #include <net/ipv6.h>
 #include <net/addrconf.h>
+#include <net/inet_frag.h>

 #ifdef CONFIG_SYSCTL

@@ -41,7 +42,7 @@ static ctl_table ipv6_table[] = {
  {
   .ctl_name = NET_IPV6_IP6FRAG_HIGH_THRESH,
   .procname = "ip6frag_high_thresh",
-  .data  = &sysctl_ip6frag_high_thresh,
+  .data  = &ip6_frags_ctl.high_thresh,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec
@@ -49,7 +50,7 @@ static ctl_table ipv6_table[] = {
  {
   .ctl_name = NET_IPV6_IP6FRAG_LOW_THRESH,
   .procname = "ip6frag_low_thresh",
-  .data  = &sysctl_ip6frag_low_thresh,
+  .data  = &ip6_frags_ctl.low_thresh,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec
@@ -57,7 +58,7 @@ static ctl_table ipv6_table[] = {
```

```
 {
  .ctl_name = NET_IPV6_IP6FRAG_TIME,
  .procname = "ip6frag_time",
-  .data  = &sysctl_ip6frag_time,
+  .data  = &ip6_frags_ctl.timeout,
  .maxlen  = sizeof(int),
  .mode  = 0644,
  .proc_handler = &proc_dointvec_jiffies,
@@ -66,7 +67,7 @@ static ctl_table ipv6_table[] = {
 {
  .ctl_name = NET_IPV6_IP6FRAG_SECRET_INTERVAL,
  .procname = "ip6frag_secret_interval",
-  .data  = &sysctl_ip6frag_secret_interval,
+  .data  = &ip6_frags_ctl.secret_interval,
  .maxlen  = sizeof(int),
  .mode  = 0644,
  .proc_handler = &proc_dointvec_jiffies,
--
1.5.3.4
```

Subject: [PATCH 4/9] Consolidate the xxx_frag_kill
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:12:43 GMT
View Forum Message <> Reply to Message

Since now all the xxx_frag_kill functions now work
with the generic inet_frag_queue data type, this can
be moved into a common place.

The xxx_unlink() code is moved as well.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index ada03ba..9902363 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -41,4 +41,6 @@ struct inet_frags {
 void inet_frags_init(struct inet_frags *);
 void inet_frags_fini(struct inet_frags *);

+void inet_frag_kill(struct inet_frag_queue *q, struct inet_frags *f);
+
 #endif
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
index 69623ff..534eaa8 100644
```

```
--- a/net/ipv4/inet_fragment.c
+++ b/net/ipv4/inet_fragment.c
@@ -42,3 +42,26 @@ void inet_frags_fini(struct inet_frags *f)
 {
 }
 EXPORT_SYMBOL(inet_frags_fini);
+
+static inline void fq_unlink(struct inet_frag_queue *fq, struct inet_frags *f)
+{
+ write_lock(&f->lock);
+ hlist_del(&fq->list);
+ list_del(&fq->lru_list);
+ f->nqueues--;
+ write_unlock(&f->lock);
+}
+
+void inet_frag_kill(struct inet_frag_queue *fq, struct inet_frags *f)
+{
+ if (del_timer(&fq->timer))
+  atomic_dec(&fq->refcnt);
+
+ if (!(fq->last_in & COMPLETE)) {
+  fq_unlink(fq, f);
+  atomic_dec(&fq->refcnt);
+  fq->last_in |= COMPLETE;
+ }
+}
+
+EXPORT_SYMBOL(inet_frag_kill);
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 61035a8..5b376c4 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -105,20 +105,6 @@ int ip_frag_mem(void)
  return atomic_read(&ip4_frags.mem);
 }

-static __inline__ void __ipq_unlink(struct ipq *qp)
-{
- hlist_del(&qp->q.list);
- list_del(&qp->q.lru_list);
- ip4_frags.nqueues--;
-}
-
-static __inline__ void ipq_unlink(struct ipq *ipq)
-{
- write_lock(&ip4_frags.lock);
- __ipq_unlink(ipq);
```

```
- write_unlock(&ip4_frags.lock);
-}
-
 static unsigned int ipqhashfn(__be16 id, __be32 saddr, __be32 daddr, u8 prot)
 {
  return jhash_3words((__force u32)id << 16 | prot,
@@ -219,14 +205,7 @@ static __inline__ void ipq_put(struct ipq *ipq, int *work)
  */
 static void ipq_kill(struct ipq *ipq)
 {
- if (del_timer(&ipq->q.timer))
-  atomic_dec(&ipq->q.refcnt);
-
- if (!(ipq->q.last_in & COMPLETE)) {
-  ipq_unlink(ipq);
-  atomic_dec(&ipq->q.refcnt);
-  ipq->q.last_in |= COMPLETE;
- }
+ inet_frag_kill(&ipq->q, &ip4_frags);
 }

 /* Memory limiting on fragments.  Evictor trashes the oldest
diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 966a888..2ebe515 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -79,20 +79,6 @@ struct inet_frags_ctl nf_frags_ctl __read_mostly = {

 static struct inet_frags nf_frags;

-static __inline__ void __fq_unlink(struct nf_ct_frag6_queue *fq)
-{
- hlist_del(&fq->q.list);
- list_del(&fq->q.lru_list);
- nf_frags.nqueues--;
-}
-
-static __inline__ void fq_unlink(struct nf_ct_frag6_queue *fq)
-{
- write_lock(&nf_frags.lock);
- __fq_unlink(fq);
- write_unlock(&nf_frags.lock);
-}
-
 static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
          struct in6_addr *daddr)
 {
@@ -213,14 +199,7 @@ 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->q.timer))
-  atomic_dec(&fq->q.refcnt);
-
- if (!(fq->q.last_in & COMPLETE)) {
-  fq_unlink(fq);
-  atomic_dec(&fq->q.refcnt);
-  fq->q.last_in |= COMPLETE;
- }
+ inet_frag_kill(&fq->q, &nf_frags);
 }

 static void nf_ct_frag6_evictor(void)
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index f0e22be..57e32f4 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -100,20 +100,6 @@ int ip6_frag_mem(void)
  return atomic_read(&ip6_frags.mem);
 }

-static __inline__ void __fq_unlink(struct frag_queue *fq)
-{
- hlist_del(&fq->q.list);
- list_del(&fq->q.lru_list);
- ip6_frags.nqueues--;
-}
-
-static __inline__ void fq_unlink(struct frag_queue *fq)
-{
- write_lock(&ip6_frags.lock);
- __fq_unlink(fq);
- write_unlock(&ip6_frags.lock);
-}
-
 /*
  * callers should be careful not to use the hash value outside the ipfrag_lock
  * as doing so could race with ipfrag_hash_rnd being recalculated.
@@ -236,14 +222,7 @@ static __inline__ void fq_put(struct frag_queue *fq, int *work)
  */
 static __inline__ void fq_kill(struct frag_queue *fq)
 {
- if (del_timer(&fq->q.timer))
-  atomic_dec(&fq->q.refcnt);
-
-
```

```
- if (!(fq->q.last_in & COMPLETE)) {
-  fq_unlink(fq);
-  atomic_dec(&fq->q.refcnt);
-  fq->q.last_in |= COMPLETE;
- }
+ inet_frag_kill(&fq->q, &ip6_frags);
 }

 static void ip6_evictor(struct inet6_dev *idev)
--
1.5.3.4
```

Subject: [PATCH 5/9] Consolidate xxx_the secret_rebuild
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:16:25 GMT
View Forum Message <> Reply to Message

This code works with the generic data types as well, so
move this into inet_fragment.c

This move makes it possible to hide the secret_timer
management and the secret_rebuild routine completely in
the inet_fragment.c

Introduce the ->hashfn() callback in inet_frags() to get
the hashfun for a given inet_frag_queue() object.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index 9902363..e374412 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -36,6 +36,8 @@ struct inet_frags {
  atomic_t  mem;
  struct timer_list secret_timer;
  struct inet_frags_ctl *ctl;
+
+ unsigned int  (*hashfn)(struct inet_frag_queue *);
 };

 void inet_frags_init(struct inet_frags *);
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
index 534eaa8..ec10e05 100644
--- a/net/ipv4/inet_fragment.c
+++ b/net/ipv4/inet_fragment.c
```

```
@@ -16,9 +16,38 @@
 #include <linux/module.h>
 #include <linux/timer.h>
 #include <linux/mm.h>
+#include <linux/random.h>

 #include <net/inet_frag.h>

+static void inet_frag_secret_rebuild(unsigned long dummy)
+{
+ struct inet_frags *f = (struct inet_frags *)dummy;
+ unsigned long now = jiffies;
+ int i;
+
+ write_lock(&f->lock);
+ get_random_bytes(&f->rnd, sizeof(u32));
+ for (i = 0; i < INETFRAGS_HASHSZ; i++) {
+  struct inet_frag_queue *q;
+  struct hlist_node *p, *n;
+
+  hlist_for_each_entry_safe(q, p, n, &f->hash[i], list) {
+   unsigned int hval = f->hashfn(q);
+
+   if (hval != i) {
+    hlist_del(&q->list);
+
+    /* Relink to new hash chain. */
+    hlist_add_head(&q->list, &f->hash[hval]);
+   }
+  }
+ }
+ write_unlock(&f->lock);
+
+ mod_timer(&f->secret_timer, now + f->ctl->secret_interval);
+}
+
 void inet_frags_init(struct inet_frags *f)
 {
  int i;
@@ -35,11 +64,17 @@ void inet_frags_init(struct inet_frags *f)
  f->nqueues = 0;
  atomic_set(&f->mem, 0);

+ init_timer(&f->secret_timer);
+ f->secret_timer.function = inet_frag_secret_rebuild;
+ f->secret_timer.data = (unsigned long)f;
+ f->secret_timer.expires = jiffies + f->ctl->secret_interval;
+ add_timer(&f->secret_timer);
```

```
 }
 EXPORT_SYMBOL(inet_frags_init);

 void inet_frags_fini(struct inet_frags *f)
 {
+ del_timer(&f->secret_timer);
 }
 EXPORT_SYMBOL(inet_frags_fini);

diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 5b376c4..7aee137 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -112,32 +112,12 @@ static unsigned int ipqhashfn(__be16 id, __be32 saddr, __be32 daddr,
u8 prot)
        ip4_frags.rnd) & (INETFRAGS_HASHSZ - 1);
 }

-static void ipfrag_secret_rebuild(unsigned long dummy)
+static unsigned int ip4_hashfn(struct inet_frag_queue *q)
 {
- unsigned long now = jiffies;
- int i;
+ struct ipq *ipq;

- write_lock(&ip4_frags.lock);
- get_random_bytes(&ip4_frags.rnd, sizeof(u32));
- for (i = 0; i < INETFRAGS_HASHSZ; i++) {
-  struct ipq *q;
-  struct hlist_node *p, *n;
-
-  hlist_for_each_entry_safe(q, p, n, &ip4_frags.hash[i], q.list) {
-   unsigned int hval = ipqhashfn(q->id, q->saddr,
-           q->daddr, q->protocol);
-
-   if (hval != i) {
-    hlist_del(&q->q.list);
-
-    /* Relink to new hash chain. */
-    hlist_add_head(&q->q.list, &ip4_frags.hash[hval]);
-   }
-  }
- }
- write_unlock(&ip4_frags.lock);
-
- mod_timer(&ip4_frags.secret_timer, now + ip4_frags_ctl.secret_interval);
+ ipq = container_of(q, struct ipq, q);
+ return ipqhashfn(ipq->id, ipq->saddr, ipq->daddr, ipq->protocol);
```

```
 }

 /* Memory Tracking Functions. */
@@ -705,12 +685,8 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)

 void __init ipfrag_init(void)
 {
- init_timer(&ip4_frags.secret_timer);
- ip4_frags.secret_timer.function = ipfrag_secret_rebuild;
- ip4_frags.secret_timer.expires = jiffies + ip4_frags_ctl.secret_interval;
- add_timer(&ip4_frags.secret_timer);
-
  ip4_frags.ctl = &ip4_frags_ctl;
+ ip4_frags.hashfn = ip4_hashfn;
  inet_frags_init(&ip4_frags);
 }

diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 2ebe515..a3aef38 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -106,32 +106,12 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  return c & (INETFRAGS_HASHSZ - 1);
 }

-static void nf_ct_frag6_secret_rebuild(unsigned long dummy)
+static unsigned int nf_hashfn(struct inet_frag_queue *q)
 {
- unsigned long now = jiffies;
- int i;
+ struct nf_ct_frag6_queue *nq;

- write_lock(&nf_frags.lock);
- get_random_bytes(&nf_frags.rnd, sizeof(u32));
- for (i = 0; i < INETFRAGS_HASHSZ; i++) {
-  struct nf_ct_frag6_queue *q;
-  struct hlist_node *p, *n;
-
-  hlist_for_each_entry_safe(q, p, n, &nf_frags.hash[i], q.list) {
-   unsigned int hval = ip6qhashfn(q->id,
-             &q->saddr,
-             &q->daddr);
-   if (hval != i) {
-    hlist_del(&q->q.list);
-    /* Relink to new hash chain. */
-    hlist_add_head(&q->q.list,
-            &nf_frags.hash[hval]);
-   }
```

```
-  }
- }
- write_unlock(&nf_frags.lock);
-
- mod_timer(&nf_frags.secret_timer, now + nf_frags_ctl.secret_interval);
+ nq = container_of(q, struct nf_ct_frag6_queue, q);
+ return ip6qhashfn(nq->id, &nq->saddr, &nq->daddr);
 }

 /* Memory Tracking Functions. */
@@ -817,11 +797,8 @@ int nf_ct_frag6_kfree_frags(struct sk_buff *skb)

 int nf_ct_frag6_init(void)
 {
- setup_timer(&nf_frags.secret_timer, nf_ct_frag6_secret_rebuild, 0);
- nf_frags.secret_timer.expires = jiffies + nf_frags_ctl.secret_interval;
- add_timer(&nf_frags.secret_timer);
-
  nf_frags.ctl = &nf_frags_ctl;
+ nf_frags.hashfn = nf_hashfn;
  inet_frags_init(&nf_frags);

  return 0;
@@ -831,7 +808,6 @@ void nf_ct_frag6_cleanup(void)
 {
  inet_frags_fini(&nf_frags);

- del_timer(&nf_frags.secret_timer);
  nf_frags_ctl.low_thresh = 0;
  nf_ct_frag6_evictor();
 }
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 57e32f4..7a357fd 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -131,35 +131,12 @@ static unsigned int ip6qhashfn(__be32 id, struct in6_addr *saddr,
  return c & (INETFRAGS_HASHSZ - 1);
 }

-static void ip6_frag_secret_rebuild(unsigned long dummy)
+static unsigned int ip6_hashfn(struct inet_frag_queue *q)
 {
- unsigned long now = jiffies;
- int i;
-
- write_lock(&ip6_frags.lock);
- get_random_bytes(&ip6_frags.rnd, sizeof(u32));
- for (i = 0; i < INETFRAGS_HASHSZ; i++) {
```

```
-  struct frag_queue *q;
-  struct hlist_node *p, *n;
-
-  hlist_for_each_entry_safe(q, p, n, &ip6_frags.hash[i], q.list) {
-   unsigned int hval = ip6qhashfn(q->id,
-            &q->saddr,
-            &q->daddr);
-
-   if (hval != i) {
-    hlist_del(&q->q.list);
-
-    /* Relink to new hash chain. */
-    hlist_add_head(&q->q.list,
-           &ip6_frags.hash[hval]);
-
-   }
-  }
- }
- write_unlock(&ip6_frags.lock);
+ struct frag_queue *fq;

- mod_timer(&ip6_frags.secret_timer, now + ip6_frags_ctl.secret_interval);
+ fq = container_of(q, struct frag_queue, q);
+ return ip6qhashfn(fq->id, &fq->saddr, &fq->daddr);
 }

 /* Memory Tracking Functions. */
@@ -742,11 +719,7 @@ void __init ipv6_frag_init(void)
  if (inet6_add_protocol(&frag_protocol, IPPROTO_FRAGMENT) < 0)
   printk(KERN_ERR "ipv6_frag_init: Could not register protocol\n");

- init_timer(&ip6_frags.secret_timer);
- ip6_frags.secret_timer.function = ip6_frag_secret_rebuild;
- ip6_frags.secret_timer.expires = jiffies + ip6_frags_ctl.secret_interval;
- add_timer(&ip6_frags.secret_timer);
-
  ip6_frags.ctl = &ip6_frags_ctl;
+ ip6_frags.hashfn = ip6_hashfn;
  inet_frags_init(&ip6_frags);
 }
--
1.5.3.4
```

Subject: [PATCH 6/9] Consolidate the xxx_frag_destroy
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:21:06 GMT
View Forum Message <> Reply to Message

To make in possible we need to know the exact frag queue
size for inet_frags->mem management and two callbacks:

 * to destoy the skb (optional, used in conntracks only)
 * to free the queue itself (mandatory, but later I plan to
   move the allocation and the destruction of frag_queues
   into the common place, so this callback will most likely
   be optional too).

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index e374412..2dd1cd4 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -33,16 +33,21 @@ struct inet_frags {
  rwlock_t  lock;
  u32   rnd;
  int   nqueues;
+ int   qsize;
  atomic_t  mem;
  struct timer_list secret_timer;
  struct inet_frags_ctl *ctl;

  unsigned int  (*hashfn)(struct inet_frag_queue *);
+ void   (*destructor)(struct inet_frag_queue *);
+ void   (*skb_free)(struct sk_buff *);
 };

 void inet_frags_init(struct inet_frags *);
 void inet_frags_fini(struct inet_frags *);

 void inet_frag_kill(struct inet_frag_queue *q, struct inet_frags *f);
+void inet_frag_destroy(struct inet_frag_queue *q,
+    struct inet_frags *f, int *work);

 #endif
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
index ec10e05..15fb2c4 100644
--- a/net/ipv4/inet_fragment.c
+++ b/net/ipv4/inet_fragment.c
@@ -17,6 +17,8 @@
 #include <linux/timer.h>
 #include <linux/mm.h>
 #include <linux/random.h>
+#include <linux/skbuff.h>
```

```
+#include <linux/rtnetlink.h>

 #include <net/inet_frag.h>

@@ -100,3 +102,41 @@ void inet_frag_kill(struct inet_frag_queue *fq, struct inet_frags *f)
 }

 EXPORT_SYMBOL(inet_frag_kill);
+
+static inline void frag_kfree_skb(struct inet_frags *f, struct sk_buff *skb,
+      int *work)
+{
+ if (work)
+  *work -= skb->truesize;
+
+ atomic_sub(skb->truesize, &f->mem);
+ if (f->skb_free)
+  f->skb_free(skb);
+ kfree_skb(skb);
+}
+
+void inet_frag_destroy(struct inet_frag_queue *q, struct inet_frags *f,
+     int *work)
+{
+ struct sk_buff *fp;
+
+ BUG_TRAP(q->last_in & COMPLETE);
+ BUG_TRAP(del_timer(&q->timer) == 0);
+
+ /* Release all fragment data. */
+ fp = q->fragments;
+ while (fp) {
+  struct sk_buff *xp = fp->next;
+
+  frag_kfree_skb(f, fp, work);
+  fp = xp;
+ }
+
+ if (work)
+  *work -= f->qsize;
+ atomic_sub(f->qsize, &f->mem);
+
+ f->destructor(q);
+
+}
+EXPORT_SYMBOL(inet_frag_destroy);
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 7aee137..a59ac39 100644
```

```
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -129,11 +129,13 @@ static __inline__ void frag_kfree_skb(struct sk_buff *skb, int *work)
  kfree_skb(skb);
 }

-static __inline__ void frag_free_queue(struct ipq *qp, int *work)
+static __inline__ void ip4_frag_free(struct inet_frag_queue *q)
 {
- if (work)
-  *work -= sizeof(struct ipq);
- atomic_sub(sizeof(struct ipq), &ip4_frags.mem);
+ struct ipq *qp;
+
+ qp = container_of(q, struct ipq, q);
+ if (qp->peer)
+  inet_putpeer(qp->peer);
  kfree(qp);
 }

@@ -150,34 +152,10 @@ static __inline__ struct ipq *frag_alloc_queue(void)

 /* Destruction primitives. */

-/* Complete destruction of ipq. */
-static void ip_frag_destroy(struct ipq *qp, int *work)
-{
- struct sk_buff *fp;
-
- 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->q.fragments;
- while (fp) {
-  struct sk_buff *xp = fp->next;
-
-  frag_kfree_skb(fp, work);
-  fp = xp;
- }
-
- /* Finally, release the queue descriptor itself. */
- frag_free_queue(qp, work);
-}
-
```

```
 static __inline__ void ipq_put(struct ipq *ipq, int *work)
 {
  if (atomic_dec_and_test(&ipq->q.refcnt))
-  ip_frag_destroy(ipq, work);
+  inet_frag_destroy(&ipq->q, &ip4_frags, work);
 }

 /* Kill ipq entry. It is not destroyed immediately,
@@ -687,6 +665,9 @@ void __init ipfrag_init(void)
 {
  ip4_frags.ctl = &ip4_frags_ctl;
  ip4_frags.hashfn = ip4_hashfn;
+ ip4_frags.destructor = ip4_frag_free;
+ ip4_frags.skb_free = NULL;
+ ip4_frags.qsize = sizeof(struct ipq);
  inet_frags_init(&ip4_frags);
 }

diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index a3aef38..785f5cd 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -114,25 +114,25 @@ static unsigned int nf_hashfn(struct inet_frag_queue *q)
  return ip6qhashfn(nq->id, &nq->saddr, &nq->daddr);
 }

+static void nf_skb_free(struct sk_buff *skb)
+{
+ if (NFCT_FRAG6_CB(skb)->orig)
+  kfree_skb(NFCT_FRAG6_CB(skb)->orig);
+}
+
 /* Memory Tracking Functions. */
 static inline void frag_kfree_skb(struct sk_buff *skb, unsigned int *work)
 {
  if (work)
   *work -= skb->truesize;
  atomic_sub(skb->truesize, &nf_frags.mem);
- if (NFCT_FRAG6_CB(skb)->orig)
-  kfree_skb(NFCT_FRAG6_CB(skb)->orig);
-
+ nf_skb_free(skb);
  kfree_skb(skb);
 }

-static inline void frag_free_queue(struct nf_ct_frag6_queue *fq,
-       unsigned int *work)
+static void nf_frag_free(struct inet_frag_queue *q)
```

```
 {
- if (work)
-  *work -= sizeof(struct nf_ct_frag6_queue);
- atomic_sub(sizeof(struct nf_ct_frag6_queue), &nf_frags.mem);
- kfree(fq);
+ kfree(container_of(q, struct nf_ct_frag6_queue, q));
 }

 static inline struct nf_ct_frag6_queue *frag_alloc_queue(void)
@@ -147,31 +147,10 @@ static inline struct nf_ct_frag6_queue *frag_alloc_queue(void)

 /* Destruction primitives. */

-/* Complete destruction of fq. */
-static void nf_ct_frag6_destroy(struct nf_ct_frag6_queue *fq,
-    unsigned int *work)
-{
- struct sk_buff *fp;
-
- BUG_TRAP(fq->q.last_in&COMPLETE);
- BUG_TRAP(del_timer(&fq->q.timer) == 0);
-
- /* Release all fragment data. */
- fp = fq->q.fragments;
- while (fp) {
-  struct sk_buff *xp = fp->next;
-
-  frag_kfree_skb(fp, work);
-  fp = xp;
- }
-
- frag_free_queue(fq, work);
-}
-
 static __inline__ void fq_put(struct nf_ct_frag6_queue *fq, unsigned int *work)
 {
  if (atomic_dec_and_test(&fq->q.refcnt))
-  nf_ct_frag6_destroy(fq, work);
+  inet_frag_destroy(&fq->q, &nf_frags, work);
 }

 /* Kill fq entry. It is not destroyed immediately,
@@ -799,6 +778,9 @@ int nf_ct_frag6_init(void)
 {
  nf_frags.ctl = &nf_frags_ctl;
  nf_frags.hashfn = nf_hashfn;
+ nf_frags.destructor = nf_frag_free;
+ nf_frags.skb_free = nf_skb_free;
```

```
+ nf_frags.qsize = sizeof(struct nf_ct_frag6_queue);
  inet_frags_init(&nf_frags);

  return 0;
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 7a357fd..74b2113 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -148,12 +148,9 @@ static inline void frag_kfree_skb(struct sk_buff *skb, int *work)
  kfree_skb(skb);
 }

-static inline void frag_free_queue(struct frag_queue *fq, int *work)
+static void ip6_frag_free(struct inet_frag_queue *fq)
 {
- if (work)
-  *work -= sizeof(struct frag_queue);
- atomic_sub(sizeof(struct frag_queue), &ip6_frags.mem);
- kfree(fq);
+ kfree(container_of(fq, struct frag_queue, q));
 }

 static inline struct frag_queue *frag_alloc_queue(void)
@@ -168,30 +165,10 @@ static inline struct frag_queue *frag_alloc_queue(void)

 /* Destruction primitives. */

-/* Complete destruction of fq. */
-static void ip6_frag_destroy(struct frag_queue *fq, int *work)
-{
- struct sk_buff *fp;
-
- BUG_TRAP(fq->q.last_in&COMPLETE);
- BUG_TRAP(del_timer(&fq->q.timer) == 0);
-
- /* Release all fragment data. */
- fp = fq->q.fragments;
- while (fp) {
-  struct sk_buff *xp = fp->next;
-
-  frag_kfree_skb(fp, work);
-  fp = xp;
- }
-
- frag_free_queue(fq, work);
-}
-
 static __inline__ void fq_put(struct frag_queue *fq, int *work)
```

```
 {
  if (atomic_dec_and_test(&fq->q.refcnt))
-  ip6_frag_destroy(fq, work);
+  inet_frag_destroy(&fq->q, &ip6_frags, work);
 }

 /* Kill fq entry. It is not destroyed immediately,
@@ -721,5 +698,8 @@ void __init ipv6_frag_init(void)

  ip6_frags.ctl = &ip6_frags_ctl;
  ip6_frags.hashfn = ip6_hashfn;
+ ip6_frags.destructor = ip6_frag_free;
+ ip6_frags.skb_free = NULL;
+ ip6_frags.qsize = sizeof(struct frag_queue);
  inet_frags_init(&ip6_frags);
 }
--
1.5.3.4
```

Subject: [PATCH 7/9] Consolidate the xxx_evictor
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:24:27 GMT
View Forum Message <> Reply to Message

The evictors collect some statistics for ipv4 and ipv6,
so make it return the number of evicted queues and account
them all at once in the caller.

The XXX_ADD_STATS_BH() macros are just for this case,
but maybe there are places in code, that can make use of
them as well.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index 2dd1cd4..cf583cf 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -49,5 +49,6 @@ void inet_frags_fini(struct inet_frags *);
 void inet_frag_kill(struct inet_frag_queue *q, struct inet_frags *f);
 void inet_frag_destroy(struct inet_frag_queue *q,
     struct inet_frags *f, int *work);
+int inet_frag_evictor(struct inet_frags *f);

 #endif
diff --git a/include/net/ip.h b/include/net/ip.h
```

```
index e7b0feb..00ed4f3 100644
--- a/include/net/ip.h
+++ b/include/net/ip.h
@@ -160,6 +160,7 @@ DECLARE_SNMP_STAT(struct ipstats_mib, ip_statistics);
 #define IP_INC_STATS(field)  SNMP_INC_STATS(ip_statistics, field)
 #define IP_INC_STATS_BH(field)  SNMP_INC_STATS_BH(ip_statistics, field)
 #define IP_INC_STATS_USER(field)  SNMP_INC_STATS_USER(ip_statistics, field)
+#define IP_ADD_STATS_BH(field, val) SNMP_ADD_STATS_BH(ip_statistics, field, val)
 DECLARE_SNMP_STAT(struct linux_mib, net_statistics);
 #define NET_INC_STATS(field)  SNMP_INC_STATS(net_statistics, field)
 #define NET_INC_STATS_BH(field)  SNMP_INC_STATS_BH(net_statistics, field)
diff --git a/include/net/ipv6.h b/include/net/ipv6.h
index b29d76c..a0f1042 100644
--- a/include/net/ipv6.h
+++ b/include/net/ipv6.h
@@ -120,12 +120,21 @@ extern int sysctl_mld_max_msf;
  SNMP_INC_STATS##modifier(statname##_statistics, (field)); \
 })

+#define _DEVADD(statname, modifier, idev, field, val)   \
+({        \
+ struct inet6_dev *_idev = (idev);    \
+ if (likely(_idev != NULL))     \
+  SNMP_ADD_STATS##modifier((_idev)->stats.statname, (field), (val)); \
+ SNMP_ADD_STATS##modifier(statname##_statistics, (field), (val));\
+})
+
 /* MIBs */
 DECLARE_SNMP_STAT(struct ipstats_mib, ipv6_statistics);

 #define IP6_INC_STATS(idev,field) _DEVINC(ipv6, , idev, field)
 #define IP6_INC_STATS_BH(idev,field) _DEVINC(ipv6, _BH, idev, field)
 #define IP6_INC_STATS_USER(idev,field) _DEVINC(ipv6, _USER, idev, field)
+#define IP6_ADD_STATS_BH(idev,field,val) _DEVADD(ipv6, _BH, idev, field, val)

 DECLARE_SNMP_STAT(struct icmpv6_mib, icmpv6_statistics);
 DECLARE_SNMP_STAT(struct icmpv6msg_mib, icmpv6msg_statistics);
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
index 15fb2c4..484cf51 100644
--- a/net/ipv4/inet_fragment.c
+++ b/net/ipv4/inet_fragment.c
@@ -140,3 +140,35 @@ void inet_frag_destroy(struct inet_frag_queue *q, struct inet_frags *f,

 }
 EXPORT_SYMBOL(inet_frag_destroy);
+
+int inet_frag_evictor(struct inet_frags *f)
+{
```

```
+ struct inet_frag_queue *q;
+ int work, evicted = 0;
+
+ work = atomic_read(&f->mem) - f->ctl->low_thresh;
+ while (work > 0) {
+  read_lock(&f->lock);
+  if (list_empty(&f->lru_list)) {
+   read_unlock(&f->lock);
+   break;
+  }
+
+  q = list_first_entry(&f->lru_list,
+    struct inet_frag_queue, lru_list);
+  atomic_inc(&q->refcnt);
+  read_unlock(&f->lock);
+
+  spin_lock(&q->lock);
+  if (!(q->last_in & COMPLETE))
+   inet_frag_kill(q, f);
+  spin_unlock(&q->lock);
+
+  if (atomic_dec_and_test(&q->refcnt))
+   inet_frag_destroy(q, f, &work);
+  evicted++;
+ }
+
+ return evicted;
+}
+EXPORT_SYMBOL(inet_frag_evictor);
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index a59ac39..4ea7948 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -171,33 +171,11 @@ static void ipq_kill(struct ipq *ipq)
  */
 static void ip_evictor(void)
 {
- struct ipq *qp;
- struct list_head *tmp;
- int work;
-
- work = atomic_read(&ip4_frags.mem) - ip4_frags_ctl.low_thresh;
- if (work <= 0)
-  return;
-
- while (work > 0) {
-  read_lock(&ip4_frags.lock);
-  if (list_empty(&ip4_frags.lru_list)) {
```

```
-   read_unlock(&ip4_frags.lock);
-   return;
-  }
-  tmp = ip4_frags.lru_list.next;
-  qp = list_entry(tmp, struct ipq, q.lru_list);
-  atomic_inc(&qp->q.refcnt);
-  read_unlock(&ip4_frags.lock);
+ int evicted;

-  spin_lock(&qp->q.lock);
-  if (!(qp->q.last_in&COMPLETE))
-   ipq_kill(qp);
-  spin_unlock(&qp->q.lock);
-
-  ipq_put(qp, &work);
-  IP_INC_STATS_BH(IPSTATS_MIB_REASMFAILS);
- }
+ evicted = inet_frag_evictor(&ip4_frags);
+ if (evicted)
+  IP_ADD_STATS_BH(IPSTATS_MIB_REASMFAILS, evicted);
 }

 /*
diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 785f5cd..862d089 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -163,34 +163,7 @@ static __inline__ void fq_kill(struct nf_ct_frag6_queue *fq)

 static void nf_ct_frag6_evictor(void)
 {
- struct nf_ct_frag6_queue *fq;
- struct list_head *tmp;
- unsigned int work;
-
- work = atomic_read(&nf_frags.mem);
- if (work <= nf_frags_ctl.low_thresh)
-  return;
-
- work -= nf_frags_ctl.low_thresh;
- while (work > 0) {
-  read_lock(&nf_frags.lock);
-  if (list_empty(&nf_frags.lru_list)) {
-   read_unlock(&nf_frags.lock);
-   return;
-  }
-  tmp = nf_frags.lru_list.next;
-  BUG_ON(tmp == NULL);
```

```
-  fq = list_entry(tmp, struct nf_ct_frag6_queue, q.lru_list);
-  atomic_inc(&fq->q.refcnt);
-  read_unlock(&nf_frags.lock);
-
-  spin_lock(&fq->q.lock);
-  if (!(fq->q.last_in&COMPLETE))
-   fq_kill(fq);
-  spin_unlock(&fq->q.lock);
-
-  fq_put(fq, &work);
- }
+ inet_frag_evictor(&nf_frags);
 }

 static void nf_ct_frag6_expire(unsigned long data)
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 74b2113..454db16 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -181,33 +181,11 @@ static __inline__ void fq_kill(struct frag_queue *fq)

 static void ip6_evictor(struct inet6_dev *idev)
 {
- struct frag_queue *fq;
- struct list_head *tmp;
- int work;
-
- work = atomic_read(&ip6_frags.mem) - ip6_frags_ctl.low_thresh;
- if (work <= 0)
-  return;
-
- while(work > 0) {
-  read_lock(&ip6_frags.lock);
-  if (list_empty(&ip6_frags.lru_list)) {
-   read_unlock(&ip6_frags.lock);
-   return;
-  }
-  tmp = ip6_frags.lru_list.next;
-  fq = list_entry(tmp, struct frag_queue, q.lru_list);
-  atomic_inc(&fq->q.refcnt);
-  read_unlock(&ip6_frags.lock);
-
-  spin_lock(&fq->q.lock);
-  if (!(fq->q.last_in&COMPLETE))
-   fq_kill(fq);
-  spin_unlock(&fq->q.lock);
+ int evicted;
```

```
-  fq_put(fq, &work);
-  IP6_INC_STATS_BH(idev, IPSTATS_MIB_REASMFAILS);
- }
+ evicted = inet_frag_evictor(&ip6_frags);
+ if (evicted)
+  IP6_ADD_STATS_BH(idev, IPSTATS_MIB_REASMFAILS, evicted);
 }

 static void ip6_frag_expire(unsigned long data)
--
1.5.3.4
```

Subject: [PATCH 8/9] Small cleanup for xxx_put after evictor consolidation
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:27:40 GMT
View Forum Message <> Reply to Message

After the evictor code is consolidated there is no need in
passing the extra pointer to the xxx_put() functions.

The only place when it made sense was the evictor code itself.

Maybe this change must got with the previous (or with the
next) patch, but I try to make them shorter as much as
possible to simplify the review (but they are still large
anyway), so this change goes in a separate patch.

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

```
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index a59ac39..4ea7948 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -152,10 +152,10 @@ static __inline__ struct ipq *frag_alloc_queue(void)

 /* Destruction primitives. */

-static __inline__ void ipq_put(struct ipq *ipq, int *work)
+static __inline__ void ipq_put(struct ipq *ipq)
 {
  if (atomic_dec_and_test(&ipq->q.refcnt))
-  inet_frag_destroy(&ipq->q, &ip4_frags, work);
+  inet_frag_destroy(&ipq->q, &ip4_frags, NULL);
 }

 /* Kill ipq entry. It is not destroyed immediately,
```

```
@@ -227,7 +205,7 @@ static void ip_expire(unsigned long arg)
  }
 out:
  spin_unlock(&qp->q.lock);
- ipq_put(qp, NULL);
+ ipq_put(qp);
 }

 /* Creation primitives. */
@@ -257,7 +235,7 @@ static struct ipq *ip_frag_intern(struct ipq *qp_in)
    atomic_inc(&qp->q.refcnt);
    write_unlock(&ip4_frags.lock);
    qp_in->q.last_in |= COMPLETE;
-   ipq_put(qp_in, NULL);
+   ipq_put(qp_in);
    return qp;
   }
  }
@@ -652,7 +630,7 @@ struct sk_buff *ip_defrag(struct sk_buff *skb, u32 user)
    ret = ip_frag_reasm(qp, dev);

   spin_unlock(&qp->q.lock);
-  ipq_put(qp, NULL);
+  ipq_put(qp);
   return ret;
  }

diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 785f5cd..862d089 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -147,10 +147,10 @@ static inline struct nf_ct_frag6_queue *frag_alloc_queue(void)

 /* Destruction primitives. */

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

 /* Kill fq entry. It is not destroyed immediately,
@@ -206,7 +179,7 @@ static void nf_ct_frag6_expire(unsigned long data)

 out:
  spin_unlock(&fq->q.lock);
```

```
- fq_put(fq, NULL);
+ fq_put(fq);
 }

 /* Creation primitives. */
@@ -228,7 +201,7 @@ static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
    atomic_inc(&fq->q.refcnt);
    write_unlock(&nf_frags.lock);
    fq_in->q.last_in |= COMPLETE;
-   fq_put(fq_in, NULL);
+   fq_put(fq_in);
    return fq;
   }
  }
@@ -719,7 +692,7 @@ struct sk_buff *nf_ct_frag6_gather(struct sk_buff *skb)
  if (nf_ct_frag6_queue(fq, clone, fhdr, nhoff) < 0) {
   spin_unlock(&fq->q.lock);
   pr_debug("Can't insert skb to queue\n");
-  fq_put(fq, NULL);
+  fq_put(fq);
   goto ret_orig;
  }

@@ -730,7 +703,7 @@ struct sk_buff *nf_ct_frag6_gather(struct sk_buff *skb)
  }
  spin_unlock(&fq->q.lock);

- fq_put(fq, NULL);
+ fq_put(fq);
  return ret_skb;

 ret_orig:
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 74b2113..454db16 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -165,10 +165,10 @@ static inline struct frag_queue *frag_alloc_queue(void)

 /* Destruction primitives. */

-static __inline__ void fq_put(struct frag_queue *fq, int *work)
+static __inline__ void fq_put(struct frag_queue *fq)
 {
  if (atomic_dec_and_test(&fq->q.refcnt))
-  inet_frag_destroy(&fq->q, &ip6_frags, work);
+  inet_frag_destroy(&fq->q, &ip6_frags, NULL);
 }
```

```
 /* Kill fq entry. It is not destroyed immediately,
@@ -246,7 +224,7 @@ out:
  if (dev)
   dev_put(dev);
  spin_unlock(&fq->q.lock);
- fq_put(fq, NULL);
+ fq_put(fq);
 }

 /* Creation primitives. */
@@ -270,7 +248,7 @@ static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in)
    atomic_inc(&fq->q.refcnt);
    write_unlock(&ip6_frags.lock);
    fq_in->q.last_in |= COMPLETE;
-   fq_put(fq_in, NULL);
+   fq_put(fq_in);
    return fq;
   }
  }
@@ -676,7 +654,7 @@ static int ipv6_frag_rcv(struct sk_buff **skbp)
    ret = ip6_frag_reasm(fq, skbp, dev);

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

--
1.5.3.4
```

Subject: [PATCH 9/9] Consolidate the xxx_put
Posted by Pavel Emelianov on Fri, 12 Oct 2007 13:29:01 GMT
View Forum Message <> Reply to Message

These ones use the generic data types too, so move
them in one place.

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

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index cf583cf..911c2cd 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -51,4 +51,10 @@ void inet_frag_destroy(struct inet_frag_queue *q,
```

```
     struct inet_frags *f, int *work);
 int inet_frag_evictor(struct inet_frags *f);

+static inline void inet_frag_put(struct inet_frag_queue *q, struct inet_frags *f)
+{
+ if (atomic_dec_and_test(&q->refcnt))
+  inet_frag_destroy(q, f, NULL);
+}
+
 #endif
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index 4ea7948..4470a50 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -154,8 +154,7 @@ static __inline__ struct ipq *frag_alloc_queue(void)

 static __inline__ void ipq_put(struct ipq *ipq)
 {
- if (atomic_dec_and_test(&ipq->q.refcnt))
-  inet_frag_destroy(&ipq->q, &ip4_frags, NULL);
+ inet_frag_put(&ipq->q, &ip4_frags);
 }

 /* Kill ipq entry. It is not destroyed immediately,
diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 862d089..bf6f2f0 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -149,8 +149,7 @@ static inline struct nf_ct_frag6_queue *frag_alloc_queue(void)

 static __inline__ void fq_put(struct nf_ct_frag6_queue *fq)
 {
- if (atomic_dec_and_test(&fq->q.refcnt))
-  inet_frag_destroy(&fq->q, &nf_frags, NULL);
+ inet_frag_put(&fq->q, &nf_frags);
 }

 /* Kill fq entry. It is not destroyed immediately,
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 454db16..a50fbea 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -167,8 +167,7 @@ static inline struct frag_queue *frag_alloc_queue(void)

 static __inline__ void fq_put(struct frag_queue *fq)
 {
- if (atomic_dec_and_test(&fq->q.refcnt))
-  inet_frag_destroy(&fq->q, &ip6_frags, NULL);
```

+ inet_frag_put(&fq->q, &ip6_frags);
 }

 /* Kill fq entry. It is not destroyed immediately,
--
1.5.3.4

Subject: Re: [PATCH 1/9] Move common fields from frag_queues in one place
Posted by davem on Mon, 15 Oct 2007 09:24:43 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:00:06 +0400

> 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 Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 2/9] Collect frag queues management objects together
Posted by davem on Mon, 15 Oct 2007 09:32:28 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:06:13 +0400

> There are some objects that are common in all the places
> which are used to keep track of frag queues, they are:
>
> * hash table

> * LRU list
> * rw lock
> * rnd number for hash function
> * the number of queues
> * the amount of memory occupied by queues
> * secret timer
>
> Move all this stuff into one structure (struct inet_frags)
> to make it possible use them uniformly in the future. Like
> with the previous patch this mostly consists of hunks like
>
> -   write_lock(&ipfrag_lock);
> +   write_lock(&ip4_frags.lock);
>
> To address the issue with exporting the number of queues and
> the amount of memory occupied by queues outside the .c file
> they are declared in, I introduce a couple of helpers.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

'ipfrag_secret_timer' became unused, so I deleted it in this
patch too.

Subject: Re: [PATCH 3/9] Collect common sysctl variables together
Posted by davem on Mon, 15 Oct 2007 09:33:57 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:10:05 +0400

> Some sysctl variables are used to tune the frag queues
> management and it will be useful to work with them in
> a common way in the future, so move them into one
> structure, moreover they are the same for all the frag
> management codes.
>
> I don't place them in the existing inet_frags object,
> introduced in the previous patch for two reasons:
>
> 1. to keep them in the __read_mostly section;
> 2. not to export the whole inet_frags objects outside.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 4/9] Consolidate the xxx_frag_kill
Posted by davem on Mon, 15 Oct 2007 09:37:31 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:12:43 +0400

> Since now all the xxx_frag_kill functions now work
> with the generic inet_frag_queue data type, this can
> be moved into a common place.
>
> The xxx_unlink() code is moved as well.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 5/9] Consolidate xxx_the secret_rebuild
Posted by davem on Mon, 15 Oct 2007 09:38:40 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:16:25 +0400

> This code works with the generic data types as well, so
> move this into inet_fragment.c
>
> This move makes it possible to hide the secret_timer
> management and the secret_rebuild routine completely in
> the inet_fragment.c
>
> Introduce the ->hashfn() callback in inet_frags() to get
> the hashfun for a given inet_frag_queue() object.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 6/9] Consolidate the xxx_frag_destroy
Posted by davem on Mon, 15 Oct 2007 09:39:34 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:21:06 +0400

> To make in possible we need to know the exact frag queue

> size for inet_frags->mem management and two callbacks:
>
> * to destoy the skb (optional, used in conntracks only)
> * to free the queue itself (mandatory, but later I plan to
>   move the allocation and the destruction of frag_queues
>   into the common place, so this callback will most likely
>   be optional too).
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 7/9] Consolidate the xxx_evictor
Posted by davem on Mon, 15 Oct 2007 09:40:14 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:24:27 +0400

> The evictors collect some statistics for ipv4 and ipv6,
> so make it return the number of evicted queues and account
> them all at once in the caller.
>
> The XXX_ADD_STATS_BH() macros are just for this case,
> but maybe there are places in code, that can make use of
> them as well.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied, thanks!

Subject: Re: [PATCH 8/9] Small cleanup for xxx_put after evictor consolidation
Posted by davem on Mon, 15 Oct 2007 09:41:33 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:27:40 +0400

> After the evictor code is consolidated there is no need in
> passing the extra pointer to the xxx_put() functions.
>
> The only place when it made sense was the evictor code itself.
>
> Maybe this change must got with the previous (or with the
> next) patch, but I try to make them shorter as much as

> possible to simplify the review (but they are still large
> anyway), so this change goes in a separate patch.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 9/9] Consolidate the xxx_put
Posted by davem on Mon, 15 Oct 2007 09:42:05 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 17:29:01 +0400

> These ones use the generic data types too, so move
> them in one place.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Also applied, thanks!

Subject: Re: [PATCH 0/9] Consolidate IP fragment management
Posted by davem on Mon, 15 Oct 2007 09:42:48 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 12 Oct 2007 16:55:10 +0400

> Patrick recently pointed out, that there are three places that
> perform IP fragments management. In ipv4, ipv6 and in ip6
> conntracks. Looks like these places can be a bit consolidated.
>
> The proposal is to create a common structure inet_frag_queue to
> put common fields like list heads, refcounts etc in, and include
> it into the specific fragment queues. Then such objects like
> hash tables, lists, locks etc are moved to common place (struct
> inet_frags). At the end common code is moved to the
> net/ipv4/inet_fragment.c.
>
> The inet_ prefix in file names, data structures and functions, and
> the code place (net/ipv4) was proposed by Alexey, but the exact
> names were selectd by me, so maybe there can be a better ones.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Thank you for doing this work, I applied it all.

I had to decide whether to apply Herbert's recent patches first or your's, because either way there would be some conflicts to resolve.

I handled it the best I could, but it seems OK from here.
