Subject: [PATCH 2/7] Consolidate xxx_frag_intern
Posted by Pavel Emelianov on Tue, 16 Oct 2007 13:53:03 GMT
View Forum Message <> Reply to Message

This routine checks for the existence of a given entry
in the hash table and inserts the new one if needed.

The ->equal callback is used to compare two frag_queue-s
together, but this one is temporary and will be removed
later. The netfilter code and the ipv6 one use the same
routine to compare frags.

The inet_frag_intern() always returns non-NULL pointer,
so convert the inet_frag_queue into protocol specific
one (with the container_of) without any checks.

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

---

```
diff --git a/include/net/inet_frag.h b/include/net/inet_frag.h
index 911c2cd..133e187 100644
--- a/include/net/inet_frag.h
+++ b/include/net/inet_frag.h
@@ -41,6 +41,8 @@ struct inet_frags {
  unsigned int  (*hashfn)(struct inet_frag_queue *);
  void   (*destructor)(struct inet_frag_queue *);
  void   (*skb_free)(struct sk_buff *);
+ int   (*equal)(struct inet_frag_queue *q1,
+       struct inet_frag_queue *q2);
 };

 void inet_frags_init(struct inet_frags *);
@@ -50,6 +52,8 @@ 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);
+struct inet_frag_queue *inet_frag_intern(struct inet_frag_queue *q,
+  struct inet_frags *f, unsigned int hash);

 static inline void inet_frag_put(struct inet_frag_queue *q, struct inet_frags *f)
 {
diff --git a/include/net/ipv6.h b/include/net/ipv6.h
index cc796cb..ff12697 100644
--- a/include/net/ipv6.h
+++ b/include/net/ipv6.h
@@ -377,6 +377,9 @@ static inline int ipv6_prefix_equal(const struct in6_addr *a1,
        prefixlen);
```

```
 }

+struct inet_frag_queue;
+int ip6_frag_equal(struct inet_frag_queue *q1, struct inet_frag_queue *q2);
+
 static inline int ipv6_addr_any(const struct in6_addr *a)
 {
  return ((a->s6_addr32[0] | a->s6_addr32[1] |
diff --git a/net/ipv4/inet_fragment.c b/net/ipv4/inet_fragment.c
index 484cf51..15054eb 100644
--- a/net/ipv4/inet_fragment.c
+++ b/net/ipv4/inet_fragment.c
@@ -172,3 +172,40 @@ int inet_frag_evictor(struct inet_frags *f)
  return evicted;
 }
 EXPORT_SYMBOL(inet_frag_evictor);
+
+struct inet_frag_queue *inet_frag_intern(struct inet_frag_queue *qp_in,
+  struct inet_frags *f, unsigned int hash)
+{
+ struct inet_frag_queue *qp;
+#ifdef CONFIG_SMP
+ struct hlist_node *n;
+#endif
+
+ write_lock(&f->lock);
+#ifdef CONFIG_SMP
+ /* With SMP race we have to recheck hash table, because
+  * such entry could be created on other cpu, while we
+  * promoted read lock to write lock.
+  */
+ hlist_for_each_entry(qp, n, &f->hash[hash], list) {
+  if (f->equal(qp, qp_in)) {
+   atomic_inc(&qp->refcnt);
+   write_unlock(&f->lock);
+   qp_in->last_in |= COMPLETE;
+   inet_frag_put(qp_in, f);
+   return qp;
+  }
+ }
+#endif
+ qp = qp_in;
+ if (!mod_timer(&qp->timer, jiffies + f->ctl->timeout))
+  atomic_inc(&qp->refcnt);
+
+ atomic_inc(&qp->refcnt);
+ hlist_add_head(&qp->list, &f->hash[hash]);
+ list_add_tail(&qp->lru_list, &f->lru_list);
```

```
+ f->nqueues++;
+ write_unlock(&f->lock);
+ return qp;
+}
+EXPORT_SYMBOL(inet_frag_intern);
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index d12a18b..4b1bbbe 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -123,6 +123,20 @@ static unsigned int ip4_hashfn(struct inet_frag_queue *q)
  return ipqhashfn(ipq->id, ipq->saddr, ipq->daddr, ipq->protocol);
 }

+static int ip4_frag_equal(struct inet_frag_queue *q1,
+  struct inet_frag_queue *q2)
+{
+ struct ipq *qp1, *qp2;
+
+ qp1 = container_of(q1, struct ipq, q);
+ qp2 = container_of(q2, struct ipq, q);
+ return (qp1->id == qp2->id &&
+    qp1->saddr == qp2->saddr &&
+    qp1->daddr == qp2->daddr &&
+    qp1->protocol == qp2->protocol &&
+    qp1->user == qp2->user);
+}
+
 /* Memory Tracking Functions. */
 static __inline__ void frag_kfree_skb(struct sk_buff *skb, int *work)
 {
@@ -214,43 +228,10 @@ out:

 static struct ipq *ip_frag_intern(struct ipq *qp_in, unsigned int hash)
 {
- struct ipq *qp;
-#ifdef CONFIG_SMP
- struct hlist_node *n;
-#endif
+ struct inet_frag_queue *q;

- write_lock(&ip4_frags.lock);
-#ifdef CONFIG_SMP
- /* With SMP race we have to recheck hash table, because
-  * such entry could be created on other cpu, while we
-  * promoted read lock to write lock.
-  */
- 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(&ip4_frags.lock);
-    qp_in->q.last_in |= COMPLETE;
-    ipq_put(qp_in);
-    return qp;
-  }
- }
-#endif
- qp = qp_in;
-
- if (!mod_timer(&qp->q.timer, jiffies + ip4_frags_ctl.timeout))
-  atomic_inc(&qp->q.refcnt);
-
- atomic_inc(&qp->q.refcnt);
- hlist_add_head(&qp->q.list, &ip4_frags.hash[hash]);
- INIT_LIST_HEAD(&qp->q.lru_list);
- list_add_tail(&qp->q.lru_list, &ip4_frags.lru_list);
- ip4_frags.nqueues++;
- write_unlock(&ip4_frags.lock);
- return qp;
+ q = inet_frag_intern(&qp_in->q, &ip4_frags, hash);
+ return container_of(q, struct ipq, q);
 }

 /* Add an entry to the 'ipq' queue for a newly received IP datagram. */
@@ -671,6 +652,7 @@ void __init ipfrag_init(void)
  ip4_frags.destructor = ip4_frag_free;
  ip4_frags.skb_free = NULL;
  ip4_frags.qsize = sizeof(struct ipq);
+ ip4_frags.equal = ip4_frag_equal;
  inet_frags_init(&ip4_frags);
 }

diff --git a/net/ipv6/netfilter/nf_conntrack_reasm.c b/net/ipv6/netfilter/nf_conntrack_reasm.c
index 726fafd..d7dc444 100644
--- a/net/ipv6/netfilter/nf_conntrack_reasm.c
+++ b/net/ipv6/netfilter/nf_conntrack_reasm.c
@@ -187,37 +187,10 @@ out:
 static struct nf_ct_frag6_queue *nf_ct_frag6_intern(unsigned int hash,
        struct nf_ct_frag6_queue *fq_in)
 {
- struct nf_ct_frag6_queue *fq;
-#ifdef CONFIG_SMP
- struct hlist_node *n;
```

```
-#endif
-
- write_lock(&nf_frags.lock);
-#ifdef CONFIG_SMP
- 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_frags.lock);
-   fq_in->q.last_in |= COMPLETE;
-   fq_put(fq_in);
-   return fq;
-  }
- }
-#endif
- fq = fq_in;
+ struct inet_frag_queue *q;

- if (!mod_timer(&fq->q.timer, jiffies + nf_frags_ctl.timeout))
-  atomic_inc(&fq->q.refcnt);
-
- atomic_inc(&fq->q.refcnt);
- 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_frags.lru_list);
- nf_frags.nqueues++;
- write_unlock(&nf_frags.lock);
- return fq;
+ q = inet_frag_intern(&fq_in->q, &nf_frags, hash);
+ return container_of(q, struct nf_ct_frag6_queue, q);
 }


@@ -752,6 +725,7 @@ int nf_ct_frag6_init(void)
  nf_frags.destructor = nf_frag_free;
  nf_frags.skb_free = nf_skb_free;
  nf_frags.qsize = sizeof(struct nf_ct_frag6_queue);
+ nf_frags.equal = ip6_frag_equal;
  inet_frags_init(&nf_frags);

  return 0;
diff --git a/net/ipv6/reassembly.c b/net/ipv6/reassembly.c
index 0a1bf43..73ea204 100644
--- a/net/ipv6/reassembly.c
+++ b/net/ipv6/reassembly.c
@@ -143,6 +143,18 @@ static unsigned int ip6_hashfn(struct inet_frag_queue *q)
  return ip6qhashfn(fq->id, &fq->saddr, &fq->daddr);
```

```
 }

+int ip6_frag_equal(struct inet_frag_queue *q1, struct inet_frag_queue *q2)
+{
+ struct frag_queue *fq1, *fq2;
+
+ fq1 = container_of(q1, struct frag_queue, q);
+ fq2 = container_of(q2, struct frag_queue, q);
+ return (fq1->id == fq2->id &&
+   ipv6_addr_equal(&fq2->saddr, &fq1->saddr) &&
+   ipv6_addr_equal(&fq2->daddr, &fq1->daddr));
+}
+EXPORT_SYMBOL(ip6_frag_equal);
+
 /* Memory Tracking Functions. */
 static inline void frag_kfree_skb(struct sk_buff *skb, int *work)
 {
@@ -236,37 +248,10 @@ out:
 static struct frag_queue *ip6_frag_intern(struct frag_queue *fq_in,
   unsigned int hash)
 {
- struct frag_queue *fq;
-#ifdef CONFIG_SMP
- struct hlist_node *n;
-#endif
-
- write_lock(&ip6_frags.lock);
-#ifdef CONFIG_SMP
- 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_frags.lock);
-   fq_in->q.last_in |= COMPLETE;
-   fq_put(fq_in);
-   return fq;
-  }
- }
-#endif
- fq = fq_in;
-
- if (!mod_timer(&fq->q.timer, jiffies + ip6_frags_ctl.timeout))
-  atomic_inc(&fq->q.refcnt);
+ struct inet_frag_queue *q;

- atomic_inc(&fq->q.refcnt);
- 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_frags.lru_list);
- ip6_frags.nqueues++;
- write_unlock(&ip6_frags.lock);
- return fq;
+ q = inet_frag_intern(&fq_in->q, &ip6_frags, hash);
+ return container_of(q, struct frag_queue, q);
 }


@@ -699,5 +684,6 @@ void __init ipv6_frag_init(void)
  ip6_frags.destructor = ip6_frag_free;
  ip6_frags.skb_free = NULL;
  ip6_frags.qsize = sizeof(struct frag_queue);
+ ip6_frags.equal = ip6_frag_equal;
  inet_frags_init(&ip6_frags);
 }
--
1.5.3.4
```