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
```