```
**********************************************************
  533367 Sat Jul 27 17:23:33 2013
new/usr/src/uts/common/inet/ip/ip_if.c
3914 ill_frag_hash_tbl not allocated for loopback interfaces
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 1990 Mentat Inc.
 */
/*
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

/*
 * This file contains the interface control functions for IP.
 */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/dlpi.h>
#include <sys/stropts.h>
#include <sys/strsun.h>
#include <sys/sysmacros.h>
#include <sys/strsubr.h>
#include <sys/strlog.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cmn_err.h>
#include <sys/kstat.h>
#include <sys/debug.h>
#include <sys/zone.h>
#include <sys/sunldi.h>
#include <sys/file.h>
#include <sys/bitmap.h>
#include <sys/cpuvar.h>
#include <sys/time.h>
#include <sys/ctype.h>
#include <sys/kmem.h>
#include <sys/systm.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <sys/isa_defs.h>
#include <net/if.h>
#include <net/if_arp.h>
#include <net/if_types.h>
```



```
#include <net/if_dl.h>
#include <net/route.h>
#include <sys/sockio.h>
#include <netinet/in.h>
#include <netinet/ip6.h>
#include <netinet/icmp6.h>
#include <netinet/igmp_var.h>
#include <sys/policy.h>
#include <sys/ethernet.h>
#include <sys/callb.h>
#include <sys/md5.h>

#include <inet/common.h>   /* for various inet/mi.h and inet/nd.h needs */
#include <inet/mi.h>
#include <inet/nd.h>
#include <inet/tunables.h>
#include <inet/arp.h>
#include <inet/ip_arp.h>
#include <inet/mib2.h>
#include <inet/ip.h>
#include <inet/ip6.h>
#include <inet/ip6_asp.h>
#include <inet/tcp.h>
#include <inet/ip_multi.h>
#include <inet/ip_ire.h>
#include <inet/ip_ftable.h>
#include <inet/ip_rts.h>
#include <inet/ip_ndp.h>
#include <inet/ip_if.h>
#include <inet/ip_impl.h>
#include <inet/sctp_ip.h>
#include <inet/ip_netinfo.h>
#include <inet/ilb_ip.h>

#include <netinet/igmp.h>
#include <inet/ip_listutils.h>
#include <inet/ipclassifier.h>
#include <sys/mac_client.h>
#include <sys/dld.h>
#include <sys/mac_flow.h>

#include <sys/systeminfo.h>
#include <sys/bootconf.h>

#include <sys/tsol/tndb.h>
#include <sys/tsol/tnet.h>

#include <inet/rawip_impl.h> /* needed for icmp_stack_t */
#include <inet/udp_impl.h> /* needed for udp_stack_t */

/* The character which tells where the ill_name ends */
#define IPIF_SEPARATOR_CHAR     ':'

/* IP ioctl function table entry */
typedef struct ipft_s {
        int     ipft_cmd;
        pfi_t   ipft_pfi;
        int     ipft_min_size;
        int     ipft_flags;
} ipft_t;
#define IPFT_F_NO_REPLY         0x1     /* IP ioctl does not expect any reply */
#define IPFT_F_SELF_REPLY       0x2     /* ioctl callee does the ioctl reply */

static int      nd_ill_forward_get(queue_t *, mblk_t *, caddr_t, cred_t *);
static int      nd_ill_forward_set(queue_t *q, mblk_t *mp,
                    char *value, caddr_t cp, cred_t *ioc_cr);
```

```
128 static boolean_t ill_is_quiescent(ill_t *);
129 static boolean_t ip_addr_ok_v4(ipaddr_t addr, ipaddr_t subnet_mask);
130 static ip_m_t   *ip_m_lookup(t_uscalar_t mac_type);
131 static int      ip_sioctl_addr_tail(ipif_t *ipif, sin_t *sin, queue_t *q,
132     mblk_t *mp, boolean_t need_up);
133 static int      ip_sioctl_dstaddr_tail(ipif_t *ipif, sin_t *sin, queue_t *q,
134     mblk_t *mp, boolean_t need_up);
135 static int      ip_sioctl_slifzone_tail(ipif_t *ipif, zoneid_t zoneid,
136     queue_t *q, mblk_t *mp, boolean_t need_up);
137 static int      ip_sioctl_flags_tail(ipif_t *ipif, uint64_t flags, queue_t *q,
138     mblk_t *mp);
139 static int      ip_sioctl_netmask_tail(ipif_t *ipif, sin_t *sin, queue_t *q,
140     mblk_t *mp);
141 static int      ip_sioctl_subnet_tail(ipif_t *ipif, in6_addr_t, in6_addr_t,
142     queue_t *q, mblk_t *mp, boolean_t need_up);
143 static int      ip_sioctl_plink_ipmod(ipsq_t *ipsq, queue_t *q, mblk_t *mp,
144     int ioccmd, struct linkblk *li);
145 static ipaddr_t ip_subnet_mask(ipaddr_t addr, ipif_t **, ip_stack_t *);
146 static void     ip_wput_ioctl(queue_t *q, mblk_t *mp);
147 static void     ipsq_flush(ill_t *ill);

149 static  int     ip_sioctl_token_tail(ipif_t *ipif, sin6_t *sin6, int addrlen,
150     queue_t *q, mblk_t *mp, boolean_t need_up);
151 static void     ipsq_delete(ipsq_t *);

153 static ipif_t   *ipif_allocate(ill_t *ill, int id, uint_t ire_type,
154     boolean_t initialize, boolean_t insert, int *errorp);
155 static ire_t    **ipif_create_bcast_ires(ipif_t *ipif, ire_t **irep);
156 static void     ipif_delete_bcast_ires(ipif_t *ipif);
157 static int      ipif_add_ires_v4(ipif_t *, boolean_t);
158 static boolean_t ipif_comp_multi(ipif_t *old_ipif, ipif_t *new_ipif,
159                     boolean_t isv6);
160 static int      ipif_logical_down(ipif_t *ipif, queue_t *q, mblk_t *mp);
161 static void     ipif_free(ipif_t *ipif);
162 static void     ipif_free_tail(ipif_t *ipif);
163 static void     ipif_set_default(ipif_t *ipif);
164 static int      ipif_set_values(queue_t *q, mblk_t *mp,
165     char *interf_name, uint_t *ppa);
166 static int      ipif_set_values_tail(ill_t *ill, ipif_t *ipif, mblk_t *mp,
167     queue_t *q);
168 static ipif_t   *ipif_lookup_on_name(char *name, size_t namelen,
169     boolean_t do_alloc, boolean_t *exists, boolean_t isv6, zoneid_t zoneid,
170     ip_stack_t *);
171 static ipif_t   *ipif_lookup_on_name_async(char *name, size_t namelen,
172     boolean_t isv6, zoneid_t zoneid, queue_t *q, mblk_t *mp, ipsq_func_t func,
173     int *error, ip_stack_t *);

175 static int      ill_alloc_ppa(ill_if_t *, ill_t *);
176 static void     ill_delete_interface_type(ill_if_t *);
177 static int      ill_dl_up(ill_t *ill, ipif_t *ipif, mblk_t *mp, queue_t *q);
178 static void     ill_dl_down(ill_t *ill);
179 static void     ill_down(ill_t *ill);
180 static void     ill_down_ipifs(ill_t *, boolean_t);
181 static void     ill_free_mib(ill_t *ill);
182 static void     ill_glist_delete(ill_t *);
183 static void     ill_phyint_reinit(ill_t *ill);
184 static void     ill_set_nce_router_flags(ill_t *, boolean_t);
185 static void     ill_set_phys_addr_tail(ipsq_t *, queue_t *, mblk_t *, void *);
186 static void     ill_replumb_tail(ipsq_t *, queue_t *, mblk_t *, void *);

188 static ip_v6intfid_func_t ip_ether_v6intfid, ip_ib_v6intfid;
189 static ip_v6intfid_func_t ip_ipv4_v6intfid, ip_ipv6_v6intfid;
190 static ip_v6intfid_func_t ip_ipmp_v6intfid, ip_nodef_v6intfid;
191 static ip_v6intfid_func_t ip_ipv4_v6destintfid, ip_ipv6_v6destintfid;
192 static ip_v4mapinfo_func_t ip_ether_v4_mapping;
```



```
193 static ip_v6mapinfo_func_t ip_ether_v6_mapping;
194 static ip_v4mapinfo_func_t ip_ib_v4_mapping;
195 static ip_v6mapinfo_func_t ip_ib_v6_mapping;
196 static ip_v4mapinfo_func_t ip_mbcast_mapping;
197 static void     ip_cgtp_bcast_add(ire_t *, ip_stack_t *);
198 static void     ip_cgtp_bcast_delete(ire_t *, ip_stack_t *);
199 static void     phyint_free(phyint_t *);

201 static void ill_capability_dispatch(ill_t *, mblk_t *, dl_capability_sub_t *);
202 static void ill_capability_id_ack(ill_t *, mblk_t *, dl_capability_sub_t *);
203 static void ill_capability_vrrp_ack(ill_t *, mblk_t *, dl_capability_sub_t *);
204 static void ill_capability_hcksum_ack(ill_t *, mblk_t *, dl_capability_sub_t *);
205 static void ill_capability_hcksum_reset_fill(ill_t *, mblk_t *);
206 static void ill_capability_zerocopy_ack(ill_t *, mblk_t *,
207     dl_capability_sub_t *);
208 static void ill_capability_zerocopy_reset_fill(ill_t *, mblk_t *);
209 static void     ill_capability_dld_reset_fill(ill_t *, mblk_t *);
210 static void     ill_capability_dld_ack(ill_t *, mblk_t *,
211                     dl_capability_sub_t *);
212 static void     ill_capability_dld_enable(ill_t *);
213 static void     ill_capability_ack_thr(void *);
214 static void     ill_capability_lso_enable(ill_t *);

216 static ill_t    *ill_prev_usesrc(ill_t *);
217 static int      ill_relink_usesrc_ills(ill_t *, ill_t *, uint_t);
218 static void     ill_disband_usesrc_group(ill_t *);
219 static void     ip_sioctl_garp_reply(mblk_t *, ill_t *, void *, int);

221 #ifdef DEBUG
222 static  void    ill_trace_cleanup(const ill_t *);
223 static  void    ipif_trace_cleanup(const ipif_t *);
224 #endif

226 static  void    ill_dlpi_clear_deferred(ill_t *ill);

228 static  void    phyint_flags_init(phyint_t *, t_uscalar_t);

230 /*
231  * if we go over the memory footprint limit more than once in this msec
232  * interval, we'll start pruning aggressively.
233  */
234 int ip_min_frag_prune_time = 0;

236 static ipft_t   ip_ioctl_ftbl[] = {
237         { IP_IOC_IRE_DELETE, ip_ire_delete, sizeof (ipid_t), 0 },
238         { IP_IOC_IRE_DELETE_NO_REPLY, ip_ire_delete, sizeof (ipid_t),
239                 IPFT_F_NO_REPLY },
240         { IP_IOC_RTS_REQUEST, ip_rts_request, 0, IPFT_F_SELF_REPLY },
241         { 0 }
242 };
```

*unchanged_portion_omitted_*

```
283 static ill_t    ill_null;              /* Empty ILL for init. */
288 char    ipif_loopback_name[] = "lo0";

290 /* These are used by all IP network modules. */
291 sin6_t  sin6_null;      /* Zero address for quick clears */
292 sin_t   sin_null;       /* Zero address for quick clears */

294 /* When set search for unused ipif_seqid */
295 static ipif_t   ipif_zero;

297 /*
298  * ppa arena is created after these many
299  * interfaces have been plumbed.
300  */
```

```
301 uint_t  ill_no_arena = 12;      /* Setable in /etc/system */

303 /*
304  * Allocate per-interface mibs.
305  * Returns true if ok. False otherwise.
306  *  ipsq  may not yet be allocated (loopback case ).
307  */
308 static boolean_t
309 ill_allocate_mibs(ill_t *ill)
310 {
311         /* Already allocated? */
312         if (ill->ill_ip_mib != NULL) {
313                 if (ill->ill_isv6)
314                         ASSERT(ill->ill_icmp6_mib != NULL);
315                 return (B_TRUE);
316         }

318         ill->ill_ip_mib = kmem_zalloc(sizeof (*ill->ill_ip_mib),
319             KM_NOSLEEP);
320         if (ill->ill_ip_mib == NULL) {
321                 return (B_FALSE);
322         }

324         /* Setup static information */
325         SET_MIB(ill->ill_ip_mib->ipIfStatsEntrySize,
326             sizeof (mib2_ipIfStatsEntry_t));
327         if (ill->ill_isv6) {
328                 ill->ill_ip_mib->ipIfStatsIPVersion = MIB2_INETADDRESSTYPE_ipv6;
329                 SET_MIB(ill->ill_ip_mib->ipIfStatsAddrEntrySize,
330                     sizeof (mib2_ipv6AddrEntry_t));
331                 SET_MIB(ill->ill_ip_mib->ipIfStatsRouteEntrySize,
332                     sizeof (mib2_ipv6RouteEntry_t));
333                 SET_MIB(ill->ill_ip_mib->ipIfStatsNetToMediaEntrySize,
334                     sizeof (mib2_ipv6NetToMediaEntry_t));
335                 SET_MIB(ill->ill_ip_mib->ipIfStatsMemberEntrySize,
336                     sizeof (ipv6_member_t));
337                 SET_MIB(ill->ill_ip_mib->ipIfStatsGroupSourceEntrySize,
338                     sizeof (ipv6_grpsrc_t));
339         } else {
340                 ill->ill_ip_mib->ipIfStatsIPVersion = MIB2_INETADDRESSTYPE_ipv4;
341                 SET_MIB(ill->ill_ip_mib->ipIfStatsAddrEntrySize,
342                     sizeof (mib2_ipAddrEntry_t));
343                 SET_MIB(ill->ill_ip_mib->ipIfStatsRouteEntrySize,
344                     sizeof (mib2_ipRouteEntry_t));
345                 SET_MIB(ill->ill_ip_mib->ipIfStatsNetToMediaEntrySize,
346                     sizeof (mib2_ipNetToMediaEntry_t));
347                 SET_MIB(ill->ill_ip_mib->ipIfStatsMemberEntrySize,
348                     sizeof (ip_member_t));
349                 SET_MIB(ill->ill_ip_mib->ipIfStatsGroupSourceEntrySize,
350                     sizeof (ip_grpsrc_t));

352                 /*
353                  * For a v4 ill, we are done at this point, because per ill
354                  * icmp mibs are only used for v6.
355                  */
356                 return (B_TRUE);
357         }

359         ill->ill_icmp6_mib = kmem_zalloc(sizeof (*ill->ill_icmp6_mib),
360             KM_NOSLEEP);
361         if (ill->ill_icmp6_mib == NULL) {
362                 kmem_free(ill->ill_ip_mib, sizeof (*ill->ill_ip_mib));
363                 ill->ill_ip_mib = NULL;
364                 return (B_FALSE);
365         }
366         /* static icmp info */
```



```
367         ill->ill_icmp6_mib->ipv6IfIcmpEntrySize =
368             sizeof (mib2_ipv6IfIcmpEntry_t);
369         /*
370          * The ipIfStatsIfindex and ipv6IfIcmpIndex will be assigned later
371          * after the phyint merge occurs in ipif_set_values -> ill_glist_insert
372          * -> ill_phyint_reinit
373          */
374         return (B_TRUE);
375 }
_____unchanged_portion_omitted_

3335 /*
3336  * Here we perform initialisation of the ill_t common to both regular
3337  * interface ILLs and the special loopback ILL created by ill_lookup_on_name.
3332  * ill_init is called by ip_open when a device control stream is opened.
3333  * It does a few initializations, and shoots a DL_INFO_REQ message down
3334  * to the driver.  The response is later picked up in ip_rput_dlpi and
3335  * used to set up default mechanisms for talking to the driver.  (Always
3336  * called as writer.)
3337  *
3338  * If this function returns error, ip_open will call ip_close which in
3339  * turn will call ill_delete to clean up any memory allocated here that
3340  * is not yet freed.
3338  */
3339 static int
3340 ill_init_common(ill_t *ill, queue_t *q, boolean_t isv6, boolean_t is_loopback,
3341     boolean_t ipsq_enter)
3342 int
3343 ill_init(queue_t *q, ill_t *ill)
3342 {
3343         int count;
3346         dl_info_req_t   *dlir;
3347         mblk_t  *info_mp;
3344         uchar_t *frag_ptr;

3350         /*
3351          * The ill is initialized to zero by mi_alloc*(). In addition
3352          * some fields already contain valid values, initialized in
3353          * ip_open(), before we reach here.
3354          */
3346         mutex_init(&ill->ill_lock, NULL, MUTEX_DEFAULT, 0);
3347         mutex_init(&ill->ill_saved_ire_lock, NULL, MUTEX_DEFAULT, NULL);
3348         ill->ill_saved_ire_cnt = 0;

3350         if (is_loopback) {
3351                 ill->ill_max_frag = isv6 ? ip_loopback_mtu_v6plus :
3352                     ip_loopback_mtuplus;
3353                 /*
3354                  * No resolver here.
3355                  */
3356                 ill->ill_net_type = IRE_LOOPBACK;
3357         } else {
3358                 ill->ill_rq = q;
3359                 ill->ill_wq = WR(q);
3360                 ill->ill_ppa = UINT_MAX;
3361         }

3363         ill->ill_isv6 = isv6;
3362         info_mp = allocb(MAX(sizeof (dl_info_req_t), sizeof (dl_info_ack_t)),
3363             BPRI_HI);
3364         if (info_mp == NULL)
3365                 return (ENOMEM);

3365         /*
3366          * Allocate sufficient space to contain our fragment hash table and
3367          * the device name.
```

```
3368          */
3369         frag_ptr = (uchar_t *)mi_zalloc(ILL_FRAG_HASH_TBL_SIZE + 2 * LIFNAMSIZ);
3370         if (frag_ptr == NULL)
3372         if (frag_ptr == NULL) {
3373                 freemsg(info_mp);
3371                 return (ENOMEM);
3375         }
3372         ill->ill_frag_ptr = frag_ptr;
3373         ill->ill_frag_free_num_pkts = 0;
3374         ill->ill_last_frag_clean_time = 0;
3375         ill->ill_frag_hash_tbl = (ipfb_t *)frag_ptr;
3376         ill->ill_name = (char *)(frag_ptr + ILL_FRAG_HASH_TBL_SIZE);
3377         for (count = 0; count < ILL_FRAG_HASH_TBL_COUNT; count++) {
3378                 mutex_init(&ill->ill_frag_hash_tbl[count].ipfb_lock,
3379                     NULL, MUTEX_DEFAULT, NULL);
3380         }

3382         ill->ill_phyint = (phyint_t *)mi_zalloc(sizeof (phyint_t));
3383         if (ill->ill_phyint == NULL) {
3388                 freemsg(info_mp);
3384                 mi_free(frag_ptr);
3385                 return (ENOMEM);
3386         }

3388         mutex_init(&ill->ill_phyint->phyint_lock, NULL, MUTEX_DEFAULT, 0);
3389         if (isv6) {
3390                 ill->ill_phyint->phyint_illv6 = ill;
3391         } else {
3394         /*
3395          * For now pretend this is a v4 ill. We need to set phyint_ill*
3396          * at this point because of the following reason. If we can't
3397          * enter the ipsq at some point and cv_wait, the writer that
3398          * wakes us up tries to locate us using the list of all phyints
3399          * in an ipsq and the ills from the phyint thru the phyint_ill*.
3400          * If we don't set it now, we risk a missed wakeup.
3401          */
3392                 ill->ill_phyint->phyint_illv4 = ill;
3393         }
3394         if (is_loopback) {
3395                 phyint_flags_init(ill->ill_phyint, DL_LOOP);
3396         }

3403         ill->ill_ppa = UINT_MAX;
3398         list_create(&ill->ill_nce, sizeof (nce_t), offsetof(nce_t, nce_node));

3400         ill_set_inputfn(ill);

3402         if (!ipsq_init(ill, ipsq_enter)) {
3408         if (!ipsq_init(ill, B_TRUE)) {
3409                 freemsg(info_mp);
3403                 mi_free(frag_ptr);
3404                 mi_free(ill->ill_phyint);
3405                 return (ENOMEM);
3406         }

3415         ill->ill_state_flags |= ILL_LL_SUBNET_PENDING;

3408         /* Frag queue limit stuff */
3409         ill->ill_frag_count = 0;
3410         ill->ill_ipf_gen = 0;

3412         rw_init(&ill->ill_mcast_lock, NULL, RW_DEFAULT, NULL);
3413         mutex_init(&ill->ill_mcast_serializer, NULL, MUTEX_DEFAULT, NULL);
3414         ill->ill_global_timer = INFINITY;
3415         ill->ill_mcast_v1_time = ill->ill_mcast_v2_time = 0;
3416         ill->ill_mcast_v1_tset = ill->ill_mcast_v2_tset = 0;
```



```
3417         ill->ill_mcast_rv = MCAST_DEF_ROBUSTNESS;
3418         ill->ill_mcast_qi = MCAST_DEF_QUERY_INTERVAL;

3420         /*
3421          * Initialize IPv6 configuration variables.  The IP module is always
3422          * opened as an IPv4 module.  Instead tracking down the cases where
3423          * it switches to do ipv6, we'll just initialize the IPv6 configuration
3424          * here for convenience, this has no effect until the ill is set to do
3425          * IPv6.
3426          */
3427         ill->ill_reachable_time = ND_REACHABLE_TIME;
3428         ill->ill_xmit_count = ND_MAX_MULTICAST_SOLICIT;
3429         ill->ill_max_buf = ND_MAX_Q;
3430         ill->ill_refcnt = 0;

3432         return (0);
3433 }

3435 /*
3436  * ill_init is called by ip_open when a device control stream is opened.
3437  * It does a few initializations, and shoots a DL_INFO_REQ message down
3438  * to the driver.  The response is later picked up in ip_rput_dlpi and
3439  * used to set up default mechanisms for talking to the driver.  (Always
3440  * called as writer.)
3441  *
3442  * If this function returns error, ip_open will call ip_close which in
3443  * turn will call ill_delete to clean up any memory allocated here that
3444  * is not yet freed.
3445  *
3446  * Note: ill_ipst and ill_zoneid must be set before calling ill_init.
3447  */
3448 int
3449 ill_init(queue_t *q, ill_t *ill)
3450 {
3451         int ret;
3452         dl_info_req_t   *dlir;
3453         mblk_t  *info_mp;

3455         info_mp = allocb(MAX(sizeof (dl_info_req_t), sizeof (dl_info_ack_t)),
3456             BPRI_HI);
3457         if (info_mp == NULL)
3458                 return (ENOMEM);

3460         /*
3461          * The ill is initialized to zero by mi_alloc*(). In addition
3462          * some fields already contain valid values, initialized in
3463          * ip_open(), before we reach here.
3464          *
3465          * For now pretend this is a v4 ill. We need to set phyint_ill*
3466          * at this point because of the following reason. If we can't
3467          * enter the ipsq at some point and cv_wait, the writer that
3468          * wakes us up tries to locate us using the list of all phyints
3469          * in an ipsq and the ills from the phyint thru the phyint_ill*.
3470          * If we don't set it now, we risk a missed wakeup.
3471          */
3472         if ((ret = ill_init_common(ill, q, B_FALSE, B_FALSE, B_TRUE)) != 0) {
3473                 freemsg(info_mp);
3474                 return (ret);
3475         }

3477         ill->ill_state_flags |= ILL_LL_SUBNET_PENDING;

3479         /* Send down the Info Request to the driver. */
3480         info_mp->b_datap->db_type = M_PCPROTO;
3481         dlir = (dl_info_req_t *)info_mp->b_rptr;
3482         info_mp->b_wptr = (uchar_t *)&dlir[1];
```

```
3483         dlir->dl_primitive = DL_INFO_REQ;

3485         ill->ill_dlpi_pending = DL_PRIM_INVAL;

3487         qprocson(q);
3488         ill_dlpi_send(ill, info_mp);

3490         return (0);
3491 }
```

*unchanged_portion_omitted_*

```
3679 /*
3680  * Return a pointer to the ill which matches the supplied name.  Note that
3681  * the ill name length includes the null termination character.  (May be
3682  * called as writer.)
3683  * If do_alloc and the interface is "lo0" it will be automatically created.
3684  * Cannot bump up reference on condemned ills. So dup detect can't be done
3685  * using this func.
3686  */
3687 ill_t *
3688 ill_lookup_on_name(char *name, boolean_t do_alloc, boolean_t isv6,
3689     boolean_t *did_alloc, ip_stack_t *ipst)
3690 {
3691         ill_t   *ill;
3692         ipif_t  *ipif;
3693         ipsq_t  *ipsq;
3694         kstat_named_t   *kn;
3695         boolean_t isloopback;
3696         in6_addr_t ov6addr;

3698         isloopback = mi_strcmp(name, ipif_loopback_name) == 0;

3700         rw_enter(&ipst->ips_ill_g_lock, RW_READER);
3701         ill = ill_find_by_name(name, isv6, ipst);
3702         rw_exit(&ipst->ips_ill_g_lock);
3703         if (ill != NULL)
3704                 return (ill);

3706         /*
3707          * Couldn't find it.  Does this happen to be a lookup for the
3708          * loopback device and are we allowed to allocate it?
3709          */
3710         if (!isloopback || !do_alloc)
3711                 return (NULL);

3713         rw_enter(&ipst->ips_ill_g_lock, RW_WRITER);
3714         ill = ill_find_by_name(name, isv6, ipst);
3715         if (ill != NULL) {
3716                 rw_exit(&ipst->ips_ill_g_lock);
3717                 return (ill);
3718         }

3720         /* Create the loopback device on demand */
3721         ill = (ill_t *)(mi_alloc(sizeof (ill_t) +
3722             sizeof (ipif_loopback_name), BPRI_MED));
3723         if (ill == NULL)
3724                 goto done;

3726         bzero(ill, sizeof (*ill));
3688         *ill = ill_null;
3689         mutex_init(&ill->ill_lock, NULL, MUTEX_DEFAULT, NULL);
3727         ill->ill_ipst = ipst;
3691         list_create(&ill->ill_nce, sizeof (nce_t), offsetof(nce_t, nce_node));
3728         netstack_hold(ipst->ips_netstack);
3729         /*
3730          * For exclusive stacks we set the zoneid to zero
```



```
3731          * to make IP operate as if in the global zone.
3732          */
3733         ill->ill_zoneid = GLOBAL_ZONEID;

3735         if (ill_init_common(ill, NULL, isv6, B_TRUE, B_FALSE) != 0)
3699         ill->ill_phyint = (phyint_t *)mi_zalloc(sizeof (phyint_t));
3700         if (ill->ill_phyint == NULL)
3736                 goto done;

3703         if (isv6)
3704                 ill->ill_phyint->phyint_illv6 = ill;
3705         else
3706                 ill->ill_phyint->phyint_illv4 = ill;
3707         mutex_init(&ill->ill_phyint->phyint_lock, NULL, MUTEX_DEFAULT, 0);
3708         phyint_flags_init(ill->ill_phyint, DL_LOOP);

3710         if (isv6) {
3711                 ill->ill_isv6 = B_TRUE;
3712                 ill->ill_max_frag = ip_loopback_mtu_v6plus;
3713         } else {
3714                 ill->ill_max_frag = ip_loopback_mtuplus;
3715         }
3738         if (!ill_allocate_mibs(ill))
3739                 goto done;

3741         ill->ill_current_frag = ill->ill_max_frag;
3742         ill->ill_mtu = ill->ill_max_frag;       /* Initial value */
3743         ill->ill_mc_mtu = ill->ill_mtu;
3744         /*
3745          * ipif_loopback_name can't be pointed at directly because its used
3746          * by both the ipv4 and ipv6 interfaces.  When the ill is removed
3747          * from the glist, ill_glist_delete() sets the first character of
3748          * ill_name to '\0'.
3749          */
3750         ill->ill_name = (char *)ill + sizeof (*ill);
3751         (void) strcpy(ill->ill_name, ipif_loopback_name);
3752         ill->ill_name_length = sizeof (ipif_loopback_name);
3753         /* Set ill_dlpi_pending for ipsq_current_finish() to work properly */
3754         ill->ill_dlpi_pending = DL_PRIM_INVAL;

3733         rw_init(&ill->ill_mcast_lock, NULL, RW_DEFAULT, NULL);
3734         mutex_init(&ill->ill_mcast_serializer, NULL, MUTEX_DEFAULT, NULL);
3735         ill->ill_global_timer = INFINITY;
3736         ill->ill_mcast_v1_time = ill->ill_mcast_v2_time = 0;
3737         ill->ill_mcast_v1_tset = ill->ill_mcast_v2_tset = 0;
3738         ill->ill_mcast_rv = MCAST_DEF_ROBUSTNESS;
3739         ill->ill_mcast_qi = MCAST_DEF_QUERY_INTERVAL;

3741         /* No resolver here. */
3742         ill->ill_net_type = IRE_LOOPBACK;

3744         /* Initialize the ipsq */
3745         if (!ipsq_init(ill, B_FALSE))
3746                 goto done;

3756         ipif = ipif_allocate(ill, 0L, IRE_LOOPBACK, B_TRUE, B_TRUE, NULL);
3757         if (ipif == NULL)
3758                 goto done;

3760         ill->ill_flags = ILLF_MULTICAST;

3762         ov6addr = ipif->ipif_v6lcl_addr;
3763         /* Set up default loopback address and mask. */
3764         if (!isv6) {
3765                 ipaddr_t inaddr_loopback = htonl(INADDR_LOOPBACK);
```

```
3767                 IN6_IPADDR_TO_V4MAPPED(inaddr_loopback, &ipif->ipif_v6lcl_addr);
3768                 V4MASK_TO_V6(htonl(IN_CLASSA_NET), ipif->ipif_v6net_mask);
3769                 V6_MASK_COPY(ipif->ipif_v6lcl_addr, ipif->ipif_v6net_mask,
3770                     ipif->ipif_v6subnet);
3771                 ill->ill_flags |= ILLF_IPV4;
3772         } else {
3773                 ipif->ipif_v6lcl_addr = ipv6_loopback;
3774                 ipif->ipif_v6net_mask = ipv6_all_ones;
3775                 V6_MASK_COPY(ipif->ipif_v6lcl_addr, ipif->ipif_v6net_mask,
3776                     ipif->ipif_v6subnet);
3777                 ill->ill_flags |= ILLF_IPV6;
3778         }

3780         /*
3781          * Chain us in at the end of the ill list. hold the ill
3782          * before we make it globally visible. 1 for the lookup.
3783          */
3776         ill->ill_refcnt = 0;
3784         ill_refhold(ill);

3779         ill->ill_frag_count = 0;
3780         ill->ill_frag_free_num_pkts = 0;
3781         ill->ill_last_frag_clean_time = 0;

3786         ipsq = ill->ill_phyint->phyint_ipsq;

3785         ill_set_inputfn(ill);

3788         if (ill_glist_insert(ill, "lo", isv6) != 0)
3789                 cmn_err(CE_PANIC, "cannot insert loopback interface");

3791         /* Let SCTP know so that it can add this to its list */
3792         sctp_update_ill(ill, SCTP_ILL_INSERT);

3794         /*
3795          * We have already assigned ipif_v6lcl_addr above, but we need to
3796          * call sctp_update_ipif_addr() after SCTP_ILL_INSERT, which
3797          * requires to be after ill_glist_insert() since we need the
3798          * ill_index set. Pass on ipv6_loopback as the old address.
3799          */
3800         sctp_update_ipif_addr(ipif, ov6addr);

3802         ip_rts_newaddrmsg(RTM_CHGADDR, 0, ipif, RTSQ_DEFAULT);

3804         /*
3805          * ill_glist_insert() -> ill_phyint_reinit() may have merged IPSQs.
3806          * If so, free our original one.
3807          */
3808         if (ipsq != ill->ill_phyint->phyint_ipsq)
3809                 ipsq_delete(ipsq);

3811         if (ipst->ips_loopback_ksp == NULL) {
3812                 /* Export loopback interface statistics */
3813                 ipst->ips_loopback_ksp = kstat_create_netstack("lo", 0,
3814                     ipif_loopback_name, "net",
3815                     KSTAT_TYPE_NAMED, 2, 0,
3816                     ipst->ips_netstack->netstack_stackid);
3817                 if (ipst->ips_loopback_ksp != NULL) {
3818                         ipst->ips_loopback_ksp->ks_update =
3819                             loopback_kstat_update;
3820                         kn = KSTAT_NAMED_PTR(ipst->ips_loopback_ksp);
3821                         kstat_named_init(&kn[0], "ipackets", KSTAT_DATA_UINT32);
3822                         kstat_named_init(&kn[1], "opackets", KSTAT_DATA_UINT32);
3823                         ipst->ips_loopback_ksp->ks_private =
3824                             (void *)(uintptr_t)ipst->ips_netstack->
3825                             netstack_stackid;
```



```
3826                         kstat_install(ipst->ips_loopback_ksp);
3827                 }
3828         }

3830         *did_alloc = B_TRUE;
3831         rw_exit(&ipst->ips_ill_g_lock);
3832         ill_nic_event_dispatch(ill, MAP_IPIF_ID(ill->ill_ipif->ipif_id),
3833             NE_PLUMB, ill->ill_name, ill->ill_name_length);
3834         return (ill);
3835 done:
3836         if (ill != NULL) {
3837                 if (ill->ill_phyint != NULL) {
3838                         ipsq = ill->ill_phyint->phyint_ipsq;
3839                         if (ipsq != NULL) {
3840                                 ipsq->ipsq_phyint = NULL;
3841                                 ipsq_delete(ipsq);
3842                         }
3843                         mi_free(ill->ill_phyint);
3844                 }
3845                 ill_free_mib(ill);
3846                 if (ill->ill_ipst != NULL)
3847                         netstack_rele(ill->ill_ipst->ips_netstack);
3848                 mi_free(ill);
3849         }
3850         rw_exit(&ipst->ips_ill_g_lock);
3851         return (NULL);
3852 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    5751 Sat Jul 27 17:23:36 2013
new/usr/src/uts/intel/ip/ip.global-objs.debug64
3914 ill_frag_hash_tbl not allocated for loopback interfaces
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2011 Nexenta Systems, Inc. All rights reserved
#

arp_m_tbl
arp_mod_info
arp_netinfo
arp_no_defense
arpinfo
cb_inet_devops
cl_inet_bind
cl_inet_checkspi
cl_inet_connect2
cl_inet_deletespi
cl_inet_disconnect
cl_inet_getspi
cl_inet_idlesa
cl_inet_ipident
cl_inet_isclusterwide
cl_inet_listen
cl_inet_unbind
cl_inet_unlisten
cl_sctp_assoc_change
cl_sctp_check_addrs
cl_sctp_connect
cl_sctp_disconnect
cl_sctp_listen
cl_sctp_unlisten
conn_drain_nthreads
dce_cache
default_ip6_asp_table
do_tcp_fusion
do_tcpzcopy
dohwcksum
dummy_mod_info
dummymodinfo
dummyrmodinit
dummywmodinit
eventq_queue_in
```



```
eventq_queue_nic
eventq_queue_out
fsw
gcdb_hash
gcdb_hash_size
gcdb_lock
gcgrp4_hash
gcgrp6_hash
gcgrp_hash_size
gcgrp_lock
icmp_fallback_sock_winit
icmp_frag_size_table
icmp_g_t_info_ack
icmp_ipha
icmp_max_optsize
icmp_mod_info
icmp_opt_arr
icmp_opt_obj
icmp_propinfo_tbl
icmp_valid_levels_arr
icmpinfov4
icmpinfov6
icmprinitv4
icmprinitv6
icmpwinit
ilb_conn_cache
ilb_conn_cache_timeout
ilb_conn_hash_size
ilb_conn_tcp_expiry
ilb_conn_timer_size
ilb_conn_udp_expiry
ilb_kstat_instance
ilb_kmem_flags
ilb_nat_src_hash_size
ilb_nat_src_instance
ilb_rule_hash_size
ilb_sticky_cache
ilb_sticky_hash_size
ilb_sticky_expiry
ilb_sticky_timer_size
ilb_sticky_timeout
ill_no_arena
ill_null
inet_dev_info
inet_devops
ip6_ftable_hash_size
ip6opt_ls
ip_cgtp_filter_rev
ip_conn_cache
ip_debug
ip_g_all_ones
ip_helper_stream_info
ip_helper_stream_rinit
ip_helper_stream_winit
ip_ioctl_ftbl
ip_loopback_mtu_v6plus
ip_loopback_mtuplus
ip_m_tbl
ip_max_frag_dups
ip_min_frag_prune_time
ip_minor_arena_la
ip_minor_arena_sa
ip_misc_ioctl_count
ip_misc_ioctl_table
ip_mod_info
ip_modclose_ackwait_ms
```

```
126 ip_ndx_ioctl_count
127 ip_ndx_ioctl_table
128 ip_poll_normal_ms
129 ip_poll_normal_ticks
130 ip_propinfo_tbl
131 ip_propinfo_count
132 ip_rput_pullups
133 ip_six_byte_all_ones
134 ip_squeue_create_callback
135 ip_squeue_enter
136 ip_squeue_fanout
137 ip_squeue_flag
138 ip_squeue_worker_wait
139 ip_thread_data
140 ip_thread_list
141 ip_thread_rwlock
142 ipcl_bind_fanout_size
143 ipcl_conn_hash_maxsize
144 ipcl_conn_hash_memfactor
145 ipcl_conn_hash_size
146 ipcl_iptun_fanout_size
147 ipcl_raw_fanout_size
148 ipcl_udp_fanout_size
149 ipif_loopback_name
150 ipif_zero
151 ipinfov4
152 ipinfov6
153 iplrinit
154 iplwinit
155 ipmp_kstats
156 iprinitv4
157 iprinitv6
158 ipsec_action_cache
159 ipsec_hdr_pullup_needed
160 ipsec_pol_cache
161 ipsec_policy_failure_msgs
162 ipsec_sel_cache
163 ipsec_spd_hashsize
164 ipsec_weird_null_inbound_policy
165 ipv4info
166 ipv6_all_hosts_mcast
167 ipv6_all_ones
168 ipv6_all_rtrs_mcast
169 ipv6_all_v2rtrs_mcast
170 ipv6_all_zeros
171 ipv6_ll_template
172 ipv6_loopback
173 ipv6_solicited_node_mcast
174 ipv6_unspecified_group
175 ipv6info
176 ipwinit
177 ire_cache
178 ire_gw_secattr_cache
179 ire_null
180 ire_nv_arr
181 ire_nv_tbl
182 lcl_param_arr
183 mask_rnhead
184 max_keylen
185 modldrv
186 modlinkage
187 modlstrmod
188 multicast_encap_iphdr
189 nce_cache
190 ncec_cache
191 netdev_privs
```



```
192 prov_update_handle
193 radix_mask_cache
194 radix_node_cache
195 rawip_conn_cache
196 recvq_call
197 recvq_loop_cnt
198 req_arr
199 rinit_arp
200 rn_mkfreelist
201 rn_ones
202 rn_zeros
203 rt_entry_cache
204 rts_conn_cache
205 rts_g_t_info_ack
206 rts_max_optsize
207 rts_mod_info
208 rts_opt_arr
209 rts_opt_obj
210 rts_valid_levels_arr
211 rtsinfo
212 rtsrinit
213 rtswinit
214 sctp_asconf_default_dispatch
215 sctp_asconf_dispatch_tbl
216 sctp_conn_cache
217 sctp_conn_hash_size
218 sctp_do_reclaim
219 sctp_kmem_faddr_cache
220 sctp_kmem_ftsn_set_cache
221 sctp_kmem_set_cache
222 sctp_min_assoc_listener
223 sctp_opt_arr
224 sctp_opt_arr_size
225 sctp_pa_early_abort
226 sctp_pp_early_abort
227 sctp_propinfo_tbl
228 sctp_propinfo_count
229 sctp_recvq_tq_list_max
230 sctp_recvq_tq_task_min
231 sctp_recvq_tq_thr_max
232 sctp_recvq_tq_thr_min
233 sctp_sin6_null
234 sctpdebug
235 sin6_null
236 sin_null
237 skip_sctp_cksum
238 sock_rawip_downcalls
239 sock_rts_downcalls
240 sock_tcp_downcalls
241 sock_udp_downcalls
242 sqset_global_list
243 sqset_global_size
244 sqset_lock
245 squeue_cache
246 squeue_drain_ms
247 squeue_drain_ns
248 squeue_workerwait_ms
249 squeue_workerwait_tick
250 tcp_acceptor_rinit
251 tcp_acceptor_winit
252 tcp_conn_cache
253 tcp_conn_hash_size
254 tcp_do_reclaim
255 tcp_drop_ack_unsent_cnt
256 tcp_dummy_upcalls
257 tcp_early_abort
```

```
 258 tcp_fallback_sock_winit
 259 tcp_free_list_max_cnt
 260 tcp_g_kstat
 261 tcp_g_statistics
 262 tcp_g_t_info_ack
 263 tcp_g_t_info_ack_v6
 264 tcp_icmp_source_quench
 265 tcp_init_wnd_chk
 266 tcp_max_init_cwnd
 267 tcp_max_optsize
 268 tcp_min_conn_listener
 269 tcp_notsack_blk_cache
 270 tcp_opt_arr
 271 tcp_opt_obj
 272 tcp_outbound_squeue_switch
 273 tcp_propinfo_tbl
 274 tcp_propinfo_count
 275 tcp_random_anon_port
 276 tcp_random_end_ptr
 277 tcp_random_fptr
 278 tcp_random_lock
 279 tcp_random_rptr
 280 tcp_random_state
 281 tcp_randtbl
 282 tcp_rinfo
 283 tcp_rinitv4
 284 tcp_rinitv6
 285 tcp_sock_winit
 286 tcp_squeue_flag
 287 tcp_squeue_wput
 288 tcp_static_maxpsz
 289 tcp_timercache
 290 tcp_tx_pull_len
 291 tcp_valid_levels_arr
 292 tcp_winfo
 293 tcp_winit
 294 tcpinfov4
 295 tcpinfov6
 296 tli_errs
 297 tsol_strict_error
 298 tun_spd_hashsize
 299 udp_bind_fanout_size
 300 udp_conn_cache
 301 udp_fallback_sock_winit
 302 udp_g_t_info_ack_ipv4
 303 udp_g_t_info_ack_ipv6
 304 udp_lrinit
 305 udp_lwinit
 306 udp_max_optsize
 307 udp_mod_info
 308 udp_opt_arr
 309 udp_opt_obj
 310 udp_propinfo_tbl
 311 udp_propinfo_count
 312 udp_random_anon_port
 313 udp_rinitv4
 314 udp_rinitv6
 315 udp_valid_levels_arr
 316 udp_winit
 317 udpinfov4
 318 udpinfov6
 319 winit_arp
 320 nxge_cksum_workaround
```

```
**********************************************************
    5709 Sat Jul 27 17:23:38 2013
new/usr/src/uts/intel/ip/ip.global-objs.obj64
3914 ill_frag_hash_tbl not allocated for loopback interfaces
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2011 Nexenta Systems, Inc. All rights reserved
#

arp_m_tbl
arp_mod_info
arp_netinfo
arp_no_defense
arpinfo
cb_inet_devops
cl_inet_bind
cl_inet_checkspi
cl_inet_connect2
cl_inet_deletespi
cl_inet_disconnect
cl_inet_getspi
cl_inet_idlesa
cl_inet_ipident
cl_inet_isclusterwide
cl_inet_listen
cl_inet_unbind
cl_inet_unlisten
cl_sctp_assoc_change
cl_sctp_check_addrs
cl_sctp_connect
cl_sctp_disconnect
cl_sctp_listen
cl_sctp_unlisten
conn_drain_nthreads
dce_cache
default_ip6_asp_table
do_tcp_fusion
do_tcpzcopy
dohwcksum
dummy_mod_info
dummymodinfo
dummyrmodinit
dummywmodinit
eventq_queue_in
```



```
eventq_queue_nic
eventq_queue_out
fsw
gcdb_hash
gcdb_hash_size
gcdb_lock
gcgrp4_hash
gcgrp6_hash
gcgrp_hash_size
gcgrp_lock
icmp_fallback_sock_winit
icmp_frag_size_table
icmp_g_t_info_ack
icmp_ipha
icmp_max_optsize
icmp_mod_info
icmp_opt_arr
icmp_opt_obj
icmp_propinfo_tbl
icmp_valid_levels_arr
icmpinfov4
icmpinfov6
icmprinitv4
icmprinitv6
icmpwinit
ilb_conn_cache
ilb_conn_cache_timeout
ilb_conn_hash_size
ilb_conn_tcp_expiry
ilb_conn_timer_size
ilb_conn_udp_expiry
ilb_kstat_instance
ilb_kmem_flags
ilb_nat_src_hash_size
ilb_nat_src_instance
ilb_rule_hash_size
ilb_sticky_cache
ilb_sticky_hash_size
ilb_sticky_expiry
ilb_sticky_timer_size
ilb_sticky_timeout
ill_no_arena
ill_null
inet_dev_info
inet_devops
ip6_ftable_hash_size
ip6opt_ls
ip_cgtp_filter_rev
ip_conn_cache
ip_debug
ip_g_all_ones
ip_helper_stream_info
ip_helper_stream_rinit
ip_helper_stream_winit
ip_ioctl_ftbl
ip_loopback_mtu_v6plus
ip_loopback_mtuplus
ip_m_tbl
ip_max_frag_dups
ip_min_frag_prune_time
ip_minor_arena_la
ip_minor_arena_sa
ip_misc_ioctl_count
ip_misc_ioctl_table
ip_mod_info
ip_modclose_ackwait_ms
```

```
126 ip_ndx_ioctl_count
127 ip_ndx_ioctl_table
128 ip_poll_normal_ms
129 ip_poll_normal_ticks
130 ip_propinfo_tbl
131 ip_propinfo_count
132 ip_rput_pullups
133 ip_six_byte_all_ones
134 ip_squeue_create_callback
135 ip_squeue_enter
136 ip_squeue_fanout
137 ip_squeue_flag
138 ip_squeue_worker_wait
139 ip_thread_data
140 ip_thread_list
141 ip_thread_rwlock
142 ipcl_bind_fanout_size
143 ipcl_conn_hash_maxsize
144 ipcl_conn_hash_memfactor
145 ipcl_conn_hash_size
146 ipcl_iptun_fanout_size
147 ipcl_raw_fanout_size
148 ipcl_udp_fanout_size
149 ipif_loopback_name
150 ipif_zero
151 ipinfov4
152 ipinfov6
153 iplrinit
154 iplwinit
155 ipmp_kstats
156 iprinitv4
157 iprinitv6
158 ipsec_action_cache
159 ipsec_hdr_pullup_needed
160 ipsec_pol_cache
161 ipsec_policy_failure_msgs
162 ipsec_sel_cache
163 ipsec_spd_hashsize
164 ipsec_weird_null_inbound_policy
165 ipv4info
166 ipv6_all_hosts_mcast
167 ipv6_all_ones
168 ipv6_all_rtrs_mcast
169 ipv6_all_v2rtrs_mcast
170 ipv6_all_zeros
171 ipv6_ll_template
172 ipv6_loopback
173 ipv6_solicited_node_mcast
174 ipv6_unspecified_group
175 ipv6info
176 ipwinit
177 ire_cache
178 ire_gw_secattr_cache
179 ire_null
180 ire_nv_arr
181 ire_nv_tbl
182 lcl_param_arr
183 mask_rnhead
184 max_keylen
185 modldrv
186 modlinkage
187 modlstrmod
188 multicast_encap_iphdr
189 nce_cache
190 ncec_cache
191 netdev_privs
```



```
192 prov_update_handle
193 radix_mask_cache
194 radix_node_cache
195 rawip_conn_cache
196 req_arr
197 rinit_arp
198 rn_mkfreelist
199 rn_ones
200 rn_zeros
201 rt_entry_cache
202 rts_conn_cache
203 rts_g_t_info_ack
204 rts_max_optsize
205 rts_mod_info
206 rts_opt_arr
207 rts_opt_obj
208 rts_valid_levels_arr
209 rtsinfo
210 rtsrinit
211 rtswinit
212 sctp_asconf_default_dispatch
213 sctp_asconf_dispatch_tbl
214 sctp_conn_cache
215 sctp_conn_hash_size
216 sctp_do_reclaim
217 sctp_kmem_faddr_cache
218 sctp_kmem_ftsn_set_cache
219 sctp_kmem_set_cache
220 sctp_min_assoc_listener
221 sctp_opt_arr
222 sctp_opt_arr_size
223 sctp_pa_early_abort
224 sctp_pp_early_abort
225 sctp_propinfo_tbl
226 sctp_propinfo_count
227 sctp_recvq_tq_list_max
228 sctp_recvq_tq_task_min
229 sctp_recvq_tq_thr_max
230 sctp_recvq_tq_thr_min
231 sctp_sin6_null
232 sctpdebug
233 sin6_null
234 sin_null
235 sock_rawip_downcalls
236 sock_rts_downcalls
237 sock_tcp_downcalls
238 sock_udp_downcalls
239 sqset_global_list
240 sqset_global_size
241 sqset_lock
242 squeue_cache
243 squeue_drain_ms
244 squeue_drain_ns
245 squeue_workerwait_ms
246 squeue_workerwait_tick
247 tcp_acceptor_rinit
248 tcp_acceptor_winit
249 tcp_conn_cache
250 tcp_conn_hash_size
251 tcp_do_reclaim
252 tcp_drop_ack_unsent_cnt
253 tcp_dummy_upcalls
254 tcp_early_abort
255 tcp_fallback_sock_winit
256 tcp_free_list_max_cnt
257 tcp_g_kstat
```

```
 258 tcp_g_statistics
 259 tcp_g_t_info_ack
 260 tcp_g_t_info_ack_v6
 261 tcp_icmp_source_quench
 262 tcp_init_wnd_chk
 263 tcp_max_init_cwnd
 264 tcp_max_optsize
 265 tcp_min_conn_listener
 266 tcp_notsack_blk_cache
 267 tcp_opt_arr
 268 tcp_opt_obj
 269 tcp_outbound_squeue_switch
 270 tcp_propinfo_tbl
 271 tcp_propinfo_count
 272 tcp_random_anon_port
 273 tcp_random_end_ptr
 274 tcp_random_fptr
 275 tcp_random_lock
 276 tcp_random_rptr
 277 tcp_random_state
 278 tcp_randtbl
 279 tcp_rinfo
 280 tcp_rinitv4
 281 tcp_rinitv6
 282 tcp_sock_winit
 283 tcp_squeue_flag
 284 tcp_squeue_wput
 285 tcp_static_maxpsz
 286 tcp_timercache
 287 tcp_tx_pull_len
 288 tcp_valid_levels_arr
 289 tcp_winfo
 290 tcp_winit
 291 tcpinfov4
 292 tcpinfov6
 293 tli_errs
 294 tsol_strict_error
 295 tun_spd_hashsize
 296 udp_bind_fanout_size
 297 udp_conn_cache
 298 udp_fallback_sock_winit
 299 udp_g_t_info_ack_ipv4
 300 udp_g_t_info_ack_ipv6
 301 udp_lrinit
 302 udp_lwinit
 303 udp_max_optsize
 304 udp_mod_info
 305 udp_opt_arr
 306 udp_opt_obj
 307 udp_propinfo_tbl
 308 udp_propinfo_count
 309 udp_random_anon_port
 310 udp_rinitv4
 311 udp_rinitv6
 312 udp_valid_levels_arr
 313 udp_winit
 314 udpinfov4
 315 udpinfov6
 316 winit_arp
 317 nxge_cksum_workaround
```

```
**********************************************************
    5751 Sat Jul 27 17:23:38 2013
new/usr/src/uts/sparc/ip/ip.global-objs.debug64
3914 ill_frag_hash_tbl not allocated for loopback interfaces
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2011 Nexenta Systems, Inc. All rights reserved
#

arp_m_tbl
arp_mod_info
arp_netinfo
arp_no_defense
arpinfo
cb_inet_devops
cl_inet_bind
cl_inet_checkspi
cl_inet_connect2
cl_inet_deletespi
cl_inet_disconnect
cl_inet_getspi
cl_inet_idlesa
cl_inet_ipident
cl_inet_isclusterwide
cl_inet_listen
cl_inet_unbind
cl_inet_unlisten
cl_sctp_assoc_change
cl_sctp_check_addrs
cl_sctp_connect
cl_sctp_disconnect
cl_sctp_listen
cl_sctp_unlisten
conn_drain_nthreads
dce_cache
default_ip6_asp_table
do_tcp_fusion
do_tcpzcopy
dohwcksum
dummy_mod_info
dummymodinfo
dummyrmodinit
dummywmodinit
eventq_queue_in
```



```
eventq_queue_nic
eventq_queue_out
fsw
gcdb_hash
gcdb_hash_size
gcdb_lock
gcgrp4_hash
gcgrp6_hash
gcgrp_hash_size
gcgrp_lock
icmp_fallback_sock_winit
icmp_frag_size_table
icmp_g_t_info_ack
icmp_ipha
icmp_max_optsize
icmp_mod_info
icmp_opt_arr
icmp_opt_obj
icmp_propinfo_tbl
icmp_valid_levels_arr
icmpinfov4
icmpinfov6
icmprinitv4
icmprinitv6
icmpwinit
ilb_conn_cache
ilb_conn_cache_timeout
ilb_conn_hash_size
ilb_conn_tcp_expiry
ilb_conn_timer_size
ilb_conn_udp_expiry
ilb_kstat_instance
ilb_kmem_flags
ilb_nat_src_hash_size
ilb_nat_src_instance
ilb_rule_hash_size
ilb_sticky_cache
ilb_sticky_hash_size
ilb_sticky_expiry
ilb_sticky_timer_size
ilb_sticky_timeout
ill_no_arena
ill_null
inet_dev_info
inet_devops
ip6_ftable_hash_size
ip6opt_ls
ip_cgtp_filter_rev
ip_conn_cache
ip_debug
ip_g_all_ones
ip_helper_stream_info
ip_helper_stream_rinit
ip_helper_stream_winit
ip_ioctl_ftbl
ip_loopback_mtu_v6plus
ip_loopback_mtuplus
ip_m_tbl
ip_max_frag_dups
ip_min_frag_prune_time
ip_minor_arena_la
ip_minor_arena_sa
ip_misc_ioctl_count
ip_misc_ioctl_table
ip_mod_info
ip_modclose_ackwait_ms
```

```
126 ip_ndx_ioctl_count
127 ip_ndx_ioctl_table
128 ip_poll_normal_ms
129 ip_poll_normal_ticks
130 ip_propinfo_tbl
131 ip_propinfo_count
132 ip_rput_pullups
133 ip_six_byte_all_ones
134 ip_squeue_create_callback
135 ip_squeue_enter
136 ip_squeue_fanout
137 ip_squeue_flag
138 ip_squeue_worker_wait
139 ip_thread_data
140 ip_thread_list
141 ip_thread_rwlock
142 ipcl_bind_fanout_size
143 ipcl_conn_hash_maxsize
144 ipcl_conn_hash_memfactor
145 ipcl_conn_hash_size
146 ipcl_iptun_fanout_size
147 ipcl_raw_fanout_size
148 ipcl_udp_fanout_size
149 ipif_loopback_name
150 ipif_zero
151 ipinfov4
152 ipinfov6
153 iplrinit
154 iplwinit
155 ipmp_kstats
156 iprinitv4
157 iprinitv6
158 ipsec_action_cache
159 ipsec_hdr_pullup_needed
160 ipsec_pol_cache
161 ipsec_policy_failure_msgs
162 ipsec_sel_cache
163 ipsec_spd_hashsize
164 ipsec_weird_null_inbound_policy
165 ipv4info
166 ipv6_all_hosts_mcast
167 ipv6_all_ones
168 ipv6_all_rtrs_mcast
169 ipv6_all_v2rtrs_mcast
170 ipv6_all_zeros
171 ipv6_ll_template
172 ipv6_loopback
173 ipv6_solicited_node_mcast
174 ipv6_unspecified_group
175 ipv6info
176 ipwinit
177 ire_cache
178 ire_gw_secattr_cache
179 ire_null
180 ire_nv_arr
181 ire_nv_tbl
182 lcl_param_arr
183 mask_rnhead
184 max_keylen
185 modldrv
186 modlinkage
187 modlstrmod
188 multicast_encap_iphdr
189 nce_cache
190 ncec_cache
191 netdev_privs
```



```
192 prov_update_handle
193 radix_mask_cache
194 radix_node_cache
195 rawip_conn_cache
196 recvq_call
197 recvq_loop_cnt
198 req_arr
199 rinit_arp
200 rn_mkfreelist
201 rn_ones
202 rn_zeros
203 rt_entry_cache
204 rts_conn_cache
205 rts_g_t_info_ack
206 rts_max_optsize
207 rts_mod_info
208 rts_opt_arr
209 rts_opt_obj
210 rts_valid_levels_arr
211 rtsinfo
212 rtsrinit
213 rtswinit
214 sctp_asconf_default_dispatch
215 sctp_asconf_dispatch_tbl
216 sctp_conn_cache
217 sctp_conn_hash_size
218 sctp_do_reclaim
219 sctp_kmem_faddr_cache
220 sctp_kmem_ftsn_set_cache
221 sctp_kmem_set_cache
222 sctp_min_assoc_listener
223 sctp_opt_arr
224 sctp_opt_arr_size
225 sctp_pa_early_abort
226 sctp_pp_early_abort
227 sctp_propinfo_tbl
228 sctp_propinfo_count
229 sctp_recvq_tq_list_max
230 sctp_recvq_tq_task_min
231 sctp_recvq_tq_thr_max
232 sctp_recvq_tq_thr_min
233 sctp_sin6_null
234 sctpdebug
235 sin6_null
236 sin_null
237 skip_sctp_cksum
238 sock_rawip_downcalls
239 sock_rts_downcalls
240 sock_tcp_downcalls
241 sock_udp_downcalls
242 sqset_global_list
243 sqset_global_size
244 sqset_lock
245 squeue_cache
246 squeue_drain_ms
247 squeue_drain_ns
248 squeue_workerwait_ms
249 squeue_workerwait_tick
250 tcp_acceptor_rinit
251 tcp_acceptor_winit
252 tcp_conn_cache
253 tcp_conn_hash_size
254 tcp_do_reclaim
255 tcp_drop_ack_unsent_cnt
256 tcp_dummy_upcalls
257 tcp_early_abort
```

```
 258 tcp_fallback_sock_winit
 259 tcp_free_list_max_cnt
 260 tcp_g_kstat
 261 tcp_g_statistics
 262 tcp_g_t_info_ack
 263 tcp_g_t_info_ack_v6
 264 tcp_icmp_source_quench
 265 tcp_init_wnd_chk
 266 tcp_max_init_cwnd
 267 tcp_max_optsize
 268 tcp_min_conn_listener
 269 tcp_notsack_blk_cache
 270 tcp_opt_arr
 271 tcp_opt_obj
 272 tcp_outbound_squeue_switch
 273 tcp_propinfo_tbl
 274 tcp_propinfo_count
 275 tcp_random_anon_port
 276 tcp_random_end_ptr
 277 tcp_random_fptr
 278 tcp_random_lock
 279 tcp_random_rptr
 280 tcp_random_state
 281 tcp_randtbl
 282 tcp_rinfo
 283 tcp_rinitv4
 284 tcp_rinitv6
 285 tcp_sock_winit
 286 tcp_squeue_flag
 287 tcp_squeue_wput
 288 tcp_static_maxpsz
 289 tcp_timercache
 290 tcp_tx_pull_len
 291 tcp_valid_levels_arr
 292 tcp_winfo
 293 tcp_winit
 294 tcpinfov4
 295 tcpinfov6
 296 tli_errs
 297 tsol_strict_error
 298 tun_spd_hashsize
 299 udp_bind_fanout_size
 300 udp_conn_cache
 301 udp_fallback_sock_winit
 302 udp_g_t_info_ack_ipv4
 303 udp_g_t_info_ack_ipv6
 304 udp_lrinit
 305 udp_lwinit
 306 udp_max_optsize
 307 udp_mod_info
 308 udp_opt_arr
 309 udp_opt_obj
 310 udp_propinfo_tbl
 311 udp_propinfo_count
 312 udp_random_anon_port
 313 udp_rinitv4
 314 udp_rinitv6
 315 udp_valid_levels_arr
 316 udp_winit
 317 udpinfov4
 318 udpinfov6
 319 winit_arp
 320 nxge_cksum_workaround
```

```
**********************************************************
    5709 Sat Jul 27 17:23:40 2013
new/usr/src/uts/sparc/ip/ip.global-objs.obj64
3914 ill_frag_hash_tbl not allocated for loopback interfaces
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2011 Nexenta Systems, Inc. All rights reserved
#

arp_m_tbl
arp_mod_info
arp_netinfo
arp_no_defense
arpinfo
cb_inet_devops
cl_inet_bind
cl_inet_checkspi
cl_inet_connect2
cl_inet_deletespi
cl_inet_disconnect
cl_inet_getspi
cl_inet_idlesa
cl_inet_ipident
cl_inet_isclusterwide
cl_inet_listen
cl_inet_unbind
cl_inet_unlisten
cl_sctp_assoc_change
cl_sctp_check_addrs
cl_sctp_connect
cl_sctp_disconnect
cl_sctp_listen
cl_sctp_unlisten
conn_drain_nthreads
dce_cache
default_ip6_asp_table
do_tcp_fusion
do_tcpzcopy
dohwcksum
dummy_mod_info
dummymodinfo
dummyrmodinit
dummywmodinit
eventq_queue_in
```



```
eventq_queue_nic
eventq_queue_out
fsw
gcdb_hash
gcdb_hash_size
gcdb_lock
gcgrp4_hash
gcgrp6_hash
gcgrp_hash_size
gcgrp_lock
icmp_fallback_sock_winit
icmp_frag_size_table
icmp_g_t_info_ack
icmp_ipha
icmp_max_optsize
icmp_mod_info
icmp_opt_arr
icmp_opt_obj
icmp_propinfo_tbl
icmp_valid_levels_arr
icmpinfov4
icmpinfov6
icmprinitv4
icmprinitv6
icmpwinit
ilb_conn_cache
ilb_conn_cache_timeout
ilb_conn_hash_size
ilb_conn_tcp_expiry
ilb_conn_timer_size
ilb_conn_udp_expiry
ilb_kstat_instance
ilb_kmem_flags
ilb_nat_src_hash_size
ilb_nat_src_instance
ilb_rule_hash_size
ilb_sticky_cache
ilb_sticky_hash_size
ilb_sticky_expiry
ilb_sticky_timer_size
ilb_sticky_timeout
ill_no_arena
ill_null
inet_dev_info
inet_devops
ip6_ftable_hash_size
ip6opt_ls
ip_cgtp_filter_rev
ip_conn_cache
ip_debug
ip_g_all_ones
ip_helper_stream_info
ip_helper_stream_rinit
ip_helper_stream_winit
ip_ioctl_ftbl
ip_loopback_mtu_v6plus
ip_loopback_mtuplus
ip_m_tbl
ip_max_frag_dups
ip_min_frag_prune_time
ip_minor_arena_la
ip_minor_arena_sa
ip_misc_ioctl_count
ip_misc_ioctl_table
ip_mod_info
ip_modclose_ackwait_ms
```

```
126 ip_ndx_ioctl_count
127 ip_ndx_ioctl_table
128 ip_poll_normal_ms
129 ip_poll_normal_ticks
130 ip_propinfo_tbl
131 ip_propinfo_count
132 ip_rput_pullups
133 ip_six_byte_all_ones
134 ip_squeue_create_callback
135 ip_squeue_enter
136 ip_squeue_fanout
137 ip_squeue_flag
138 ip_squeue_worker_wait
139 ip_thread_data
140 ip_thread_list
141 ip_thread_rwlock
142 ipcl_bind_fanout_size
143 ipcl_conn_hash_maxsize
144 ipcl_conn_hash_memfactor
145 ipcl_conn_hash_size
146 ipcl_iptun_fanout_size
147 ipcl_raw_fanout_size
148 ipcl_udp_fanout_size
149 ipif_loopback_name
150 ipif_zero
151 ipinfov4
152 ipinfov6
153 iplrinit
154 iplwinit
155 ipmp_kstats
156 iprinitv4
157 iprinitv6
158 ipsec_action_cache
159 ipsec_hdr_pullup_needed
160 ipsec_pol_cache
161 ipsec_policy_failure_msgs
162 ipsec_sel_cache
163 ipsec_spd_hashsize
164 ipsec_weird_null_inbound_policy
165 ipv4info
166 ipv6_all_hosts_mcast
167 ipv6_all_ones
168 ipv6_all_rtrs_mcast
169 ipv6_all_v2rtrs_mcast
170 ipv6_all_zeros
171 ipv6_ll_template
172 ipv6_loopback
173 ipv6_solicited_node_mcast
174 ipv6_unspecified_group
175 ipv6info
176 ipwinit
177 ire_cache
178 ire_gw_secattr_cache
179 ire_null
180 ire_nv_arr
181 ire_nv_tbl
182 lcl_param_arr
183 mask_rnhead
184 max_keylen
185 modldrv
186 modlinkage
187 modlstrmod
188 multicast_encap_iphdr
189 nce_cache
190 ncec_cache
191 netdev_privs
```



```
192 prov_update_handle
193 radix_mask_cache
194 radix_node_cache
195 rawip_conn_cache
196 req_arr
197 rinit_arp
198 rn_mkfreelist
199 rn_ones
200 rn_zeros
201 rt_entry_cache
202 rts_conn_cache
203 rts_g_t_info_ack
204 rts_max_optsize
205 rts_mod_info
206 rts_opt_arr
207 rts_opt_obj
208 rts_valid_levels_arr
209 rtsinfo
210 rtsrinit
211 rtswinit
212 sctp_asconf_default_dispatch
213 sctp_asconf_dispatch_tbl
214 sctp_conn_cache
215 sctp_conn_hash_size
216 sctp_do_reclaim
217 sctp_kmem_faddr_cache
218 sctp_kmem_ftsn_set_cache
219 sctp_kmem_set_cache
220 sctp_min_assoc_listener
221 sctp_opt_arr
222 sctp_opt_arr_size
223 sctp_pa_early_abort
224 sctp_pp_early_abort
225 sctp_propinfo_tbl
226 sctp_propinfo_count
227 sctp_recvq_tq_list_max
228 sctp_recvq_tq_task_min
229 sctp_recvq_tq_thr_max
230 sctp_recvq_tq_thr_min
231 sctp_sin6_null
232 sctpdebug
233 sin6_null
234 sin_null
235 sock_rawip_downcalls
236 sock_rts_downcalls
237 sock_tcp_downcalls
238 sock_udp_downcalls
239 sqset_global_list
240 sqset_global_size
241 sqset_lock
242 squeue_cache
243 squeue_drain_ms
244 squeue_drain_ns
245 squeue_workerwait_ms
246 squeue_workerwait_tick
247 tcp_acceptor_rinit
248 tcp_acceptor_winit
249 tcp_conn_cache
250 tcp_conn_hash_size
251 tcp_do_reclaim
252 tcp_drop_ack_unsent_cnt
253 tcp_dummy_upcalls
254 tcp_early_abort
255 tcp_fallback_sock_winit
256 tcp_free_list_max_cnt
257 tcp_g_kstat
```

```
 258 tcp_g_statistics
 259 tcp_g_t_info_ack
 260 tcp_g_t_info_ack_v6
 261 tcp_icmp_source_quench
 262 tcp_init_wnd_chk
 263 tcp_max_init_cwnd
 264 tcp_max_optsize
 265 tcp_min_conn_listener
 266 tcp_notsack_blk_cache
 267 tcp_opt_arr
 268 tcp_opt_obj
 269 tcp_outbound_squeue_switch
 270 tcp_propinfo_tbl
 271 tcp_propinfo_count
 272 tcp_random_anon_port
 273 tcp_random_end_ptr
 274 tcp_random_fptr
 275 tcp_random_lock
 276 tcp_random_rptr
 277 tcp_random_state
 278 tcp_randtbl
 279 tcp_rinfo
 280 tcp_rinitv4
 281 tcp_rinitv6
 282 tcp_sock_winit
 283 tcp_squeue_flag
 284 tcp_squeue_wput
 285 tcp_static_maxpsz
 286 tcp_timercache
 287 tcp_tx_pull_len
 288 tcp_valid_levels_arr
 289 tcp_winfo
 290 tcp_winit
 291 tcpinfov4
 292 tcpinfov6
 293 tli_errs
 294 tsol_strict_error
 295 tun_spd_hashsize
 296 udp_bind_fanout_size
 297 udp_conn_cache
 298 udp_fallback_sock_winit
 299 udp_g_t_info_ack_ipv4
 300 udp_g_t_info_ack_ipv6
 301 udp_lrinit
 302 udp_lwinit
 303 udp_max_optsize
 304 udp_mod_info
 305 udp_opt_arr
 306 udp_opt_obj
 307 udp_propinfo_tbl
 308 udp_propinfo_count
 309 udp_random_anon_port
 310 udp_rinitv4
 311 udp_rinitv6
 312 udp_valid_levels_arr
 313 udp_winit
 314 udpinfov4
 315 udpinfov6
 316 winit_arp
 317 nxge_cksum_workaround
```