```
**********************************************************
   39241 Wed Jan  4 15:16:36 2012
new/usr/src/uts/common/io/dld/dld_proto.c
1918 stack overflow from mac_promisc_dispatch()
**********************************************************
```

```
/*
 * 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) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2011, Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Data-Link Driver
 */
#include <sys/sysmacros.h>
#include <sys/strsubr.h>
#include <sys/strsun.h>
#include <sys/vlan.h>
#include <sys/dld_impl.h>
#include <sys/mac_client.h>
#include <sys/mac_client_impl.h>
#include <sys/mac_client_priv.h>

typedef void proto_reqfunc_t(dld_str_t *, mblk_t *);

static proto_reqfunc_t proto_info_req, proto_attach_req, proto_detach_req,
    proto_bind_req, proto_unbind_req, proto_promiscon_req, proto_promiscoff_req,
    proto_enabmulti_req, proto_disabmulti_req, proto_physaddr_req,
    proto_setphysaddr_req, proto_udqos_req, proto_req, proto_capability_req,
    proto_notify_req, proto_passive_req;

static void proto_capability_advertise(dld_str_t *, mblk_t *);
static int dld_capab_poll_disable(dld_str_t *, dld_capab_poll_t *);
static boolean_t check_mod_above(queue_t *, const char *);

#define DL_ACK_PENDING(state) \
        ((state) == DL_ATTACH_PENDING || \
        (state) == DL_DETACH_PENDING || \
        (state) == DL_BIND_PENDING || \
        (state) == DL_UNBIND_PENDING)

/*
 * Process a DLPI protocol message.
 * The primitives DL_BIND_REQ, DL_ENABMULTI_REQ, DL_PROMISCON_REQ,
 * DL_SET_PHYS_ADDR_REQ put the data link below our dld_str_t into an
 * 'active' state. The primitive DL_PASSIVE_REQ marks our dld_str_t
 * as 'passive' and forbids it from being subsequently made 'active'
```



```
 * by the above primitives.
 */
void
dld_proto(dld_str_t *dsp, mblk_t *mp)
{
        t_uscalar_t             prim;

        if (MBLKL(mp) < sizeof (t_uscalar_t)) {
                freemsg(mp);
                return;
        }
        prim = ((union DL_primitives *)mp->b_rptr)->dl_primitive;

        switch (prim) {
        case DL_INFO_REQ:
                proto_info_req(dsp, mp);
                break;
        case DL_BIND_REQ:
                proto_bind_req(dsp, mp);
                break;
        case DL_UNBIND_REQ:
                proto_unbind_req(dsp, mp);
                break;
        case DL_UNITDATA_REQ:
                proto_unitdata_req(dsp, mp);
                break;
        case DL_UDQOS_REQ:
                proto_udqos_req(dsp, mp);
                break;
        case DL_ATTACH_REQ:
                proto_attach_req(dsp, mp);
                break;
        case DL_DETACH_REQ:
                proto_detach_req(dsp, mp);
                break;
        case DL_ENABMULTI_REQ:
                proto_enabmulti_req(dsp, mp);
                break;
        case DL_DISABMULTI_REQ:
                proto_disabmulti_req(dsp, mp);
                break;
        case DL_PROMISCON_REQ:
                proto_promiscon_req(dsp, mp);
                break;
        case DL_PROMISCOFF_REQ:
                proto_promiscoff_req(dsp, mp);
                break;
        case DL_PHYS_ADDR_REQ:
                proto_physaddr_req(dsp, mp);
                break;
        case DL_SET_PHYS_ADDR_REQ:
                proto_setphysaddr_req(dsp, mp);
                break;
        case DL_NOTIFY_REQ:
                proto_notify_req(dsp, mp);
                break;
        case DL_CAPABILITY_REQ:
                proto_capability_req(dsp, mp);
                break;
        case DL_PASSIVE_REQ:
                proto_passive_req(dsp, mp);
                break;
        default:
                proto_req(dsp, mp);
                break;
        }
```

```
 128 }
_____unchanged_portion_omitted_

 568 /*
 569  * DL_PROMISCON_REQ
 570  */
 571 static void
 572 proto_promiscon_req(dld_str_t *dsp, mblk_t *mp)
 573 {
 574         dl_promiscon_req_t *dlp = (dl_promiscon_req_t *)mp->b_rptr;
 575         int             err = 0;
 576         t_uscalar_t     dl_err;
 577         uint32_t        new_flags, promisc_saved;
 576         uint32_t        promisc_saved;
 578         queue_t         *q = dsp->ds_wq;
 579         mac_perim_handle_t      mph;

 581         if (MBLKL(mp) < sizeof (dl_promiscon_req_t)) {
 582                 dl_err = DL_BADPRIM;
 583                 goto failed;
 584         }

 586         if (dsp->ds_dlstate == DL_UNATTACHED ||
 587             DL_ACK_PENDING(dsp->ds_dlstate)) {
 588                 dl_err = DL_OUTSTATE;
 589                 goto failed;
 590         }

 592         mac_perim_enter_by_mh(dsp->ds_mh, &mph);

 594         new_flags = promisc_saved = dsp->ds_promisc;
 591         promisc_saved = dsp->ds_promisc;
 595         switch (dlp->dl_level) {
 596         case DL_PROMISC_SAP:
 597                 new_flags |= DLS_PROMISC_SAP;
 594                 dsp->ds_promisc |= DLS_PROMISC_SAP;
 598                 break;

 600         case DL_PROMISC_MULTI:
 601                 new_flags |= DLS_PROMISC_MULTI;
 598                 dsp->ds_promisc |= DLS_PROMISC_MULTI;
 602                 break;

 604         case DL_PROMISC_PHYS:
 605                 new_flags |= DLS_PROMISC_PHYS;
 602                 dsp->ds_promisc |= DLS_PROMISC_PHYS;
 606                 break;

 608         default:
 609                 dl_err = DL_NOTSUPPORTED;
 610                 goto failed;
 611         }

 610         mac_perim_enter_by_mh(dsp->ds_mh, &mph);

 613         if ((promisc_saved == 0) && (err = dls_active_set(dsp)) != 0) {
 614                 ASSERT(dsp->ds_promisc == promisc_saved);
 613                 dsp->ds_promisc = promisc_saved;
 615                 dl_err = DL_SYSERR;
 616                 goto failed2;
 617         }

 619         /*
 620          * Adjust channel promiscuity.
 621          */
 622         err = dls_promisc(dsp, new_flags);
```



```
 621         err = dls_promisc(dsp, promisc_saved);

 624         if (err != 0) {
 625                 dl_err = DL_SYSERR;
 626                 dsp->ds_promisc = promisc_saved;
 627                 if (promisc_saved == 0)
 628                         dls_active_clear(dsp, B_FALSE);
 629                 goto failed2;
 630         }

 632         mac_perim_exit(mph);

 634         dlokack(q, mp, DL_PROMISCON_REQ);
 635         return;

 637 failed2:
 638         mac_perim_exit(mph);
 639 failed:
 640         dlerrorack(q, mp, DL_PROMISCON_REQ, dl_err, (t_uscalar_t)err);
 641 }

 643 /*
 644  * DL_PROMISCOFF_REQ
 645  */
 646 static void
 647 proto_promiscoff_req(dld_str_t *dsp, mblk_t *mp)
 648 {
 649         dl_promiscoff_req_t *dlp = (dl_promiscoff_req_t *)mp->b_rptr;
 650         int             err = 0;
 651         t_uscalar_t     dl_err;
 652         uint32_t        new_flags;
 651         uint32_t        promisc_saved;
 653         queue_t         *q = dsp->ds_wq;
 654         mac_perim_handle_t      mph;

 656         if (MBLKL(mp) < sizeof (dl_promiscoff_req_t)) {
 657                 dl_err = DL_BADPRIM;
 658                 goto failed;
 659         }

 661         if (dsp->ds_dlstate == DL_UNATTACHED ||
 662             DL_ACK_PENDING(dsp->ds_dlstate)) {
 663                 dl_err = DL_OUTSTATE;
 664                 goto failed;
 665         }

 667         mac_perim_enter_by_mh(dsp->ds_mh, &mph);

 669         new_flags = dsp->ds_promisc;
 666         promisc_saved = dsp->ds_promisc;
 670         switch (dlp->dl_level) {
 671         case DL_PROMISC_SAP:
 672                 if (!(dsp->ds_promisc & DLS_PROMISC_SAP)) {
 673                         dl_err = DL_NOTENAB;
 674                         goto failed;
 675                 }
 676                 new_flags &= ~DLS_PROMISC_SAP;
 673                 dsp->ds_promisc &= ~DLS_PROMISC_SAP;
 677                 break;

 679         case DL_PROMISC_MULTI:
 680                 if (!(dsp->ds_promisc & DLS_PROMISC_MULTI)) {
 681                         dl_err = DL_NOTENAB;
 682                         goto failed;
 683                 }
 684                 new_flags &= ~DLS_PROMISC_MULTI;
```

```
681                 dsp->ds_promisc &= ~DLS_PROMISC_MULTI;
685                 break;

687         case DL_PROMISC_PHYS:
688                 if (!(dsp->ds_promisc & DLS_PROMISC_PHYS)) {
689                         dl_err = DL_NOTENAB;
690                         goto failed;
691                 }
692                 new_flags &= ~DLS_PROMISC_PHYS;
689                 dsp->ds_promisc &= ~DLS_PROMISC_PHYS;
693                 break;

695         default:
696                 dl_err = DL_NOTSUPPORTED;
697                 goto failed;
698         }

697         mac_perim_enter_by_mh(dsp->ds_mh, &mph);
700         /*
701          * Adjust channel promiscuity.
702          */
703         err = dls_promisc(dsp, new_flags);
701         err = dls_promisc(dsp, promisc_saved);

705         if (err != 0) {
706                 mac_perim_exit(mph);
707                 dl_err = DL_SYSERR;
708                 goto failed;
709         }

711         ASSERT(dsp->ds_promisc == new_flags);
712         if (dsp->ds_promisc == 0)
713                 dls_active_clear(dsp, B_FALSE);

715         mac_perim_exit(mph);

717         dlokack(q, mp, DL_PROMISCOFF_REQ);
718         return;
719 failed:
720         dlerrorack(q, mp, DL_PROMISCOFF_REQ, dl_err, (t_uscalar_t)err);
721 }
```

unchanged_portion_omitted_

```
**********************************************************
   18600 Wed Jan  4 15:16:37 2012
new/usr/src/uts/common/io/dls/dls.c
1918 stack overflow from mac_promisc_dispatch()
**********************************************************
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2011, Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Data-Link Services Module
 */

#include        <sys/strsun.h>
#include        <sys/vlan.h>
#include        <sys/dld_impl.h>
#include        <sys/mac_client_priv.h>

int
dls_open(dls_link_t *dlp, dls_dl_handle_t ddh, dld_str_t *dsp)
{
        zoneid_t        zid = getzoneid();
        boolean_t       local;
        int             err;

        /*
         * Check whether this client belongs to the zone of this dlp. Note that
         * a global zone client is allowed to open a local zone dlp.
         */
        if (zid != GLOBAL_ZONEID && dlp->dl_zid != zid)
                return (ENOENT);

        /*
         * mac_start() is required for non-legacy MACs to show accurate
         * kstats even before the interface is brought up. For legacy
         * drivers, this is not needed. Further, calling mac_start() for
         * legacy drivers would make the shared-lower-stream to stay in
         * the DL_IDLE state, which in turn causes performance regression.
         */
        if (!mac_capab_get(dlp->dl_mh, MAC_CAPAB_LEGACY, NULL) &&
            ((err = mac_start(dlp->dl_mh)) != 0)) {
                return (err);
        }
```



```
        local = (zid == dlp->dl_zid);
        dlp->dl_zone_ref += (local ? 1 : 0);

        /*
         * Cache a copy of the MAC interface handle, a pointer to the
         * immutable MAC info.
         */
        dsp->ds_dlp = dlp;
        dsp->ds_mh = dlp->dl_mh;
        dsp->ds_mch = dlp->dl_mch;
        dsp->ds_mip = dlp->dl_mip;
        dsp->ds_ddh = ddh;
        dsp->ds_local = local;

        ASSERT(MAC_PERIM_HELD(dsp->ds_mh));
        return (0);
}

void
dls_close(dld_str_t *dsp)
{
        dls_link_t              *dlp = dsp->ds_dlp;
        dls_multicst_addr_t     *p;
        dls_multicst_addr_t     *nextp;
        uint32_t                old_flags;

        ASSERT(dsp->ds_datathr_cnt == 0);
        ASSERT(MAC_PERIM_HELD(dsp->ds_mh));

        if (dsp->ds_local)
                dlp->dl_zone_ref--;
        dsp->ds_local = B_FALSE;

        /*
         * Walk the list of multicast addresses, disabling each at the MAC.
         * Note that we must remove multicast address before
         * mac_unicast_remove() (called by dls_active_clear()) because
         * mac_multicast_remove() relies on the unicast flows on the mac
         * client.
         */
        for (p = dsp->ds_dmap; p != NULL; p = nextp) {
                (void) mac_multicast_remove(dsp->ds_mch, p->dma_addr);
                nextp = p->dma_nextp;
                kmem_free(p, sizeof (dls_multicst_addr_t));
        }
        dsp->ds_dmap = NULL;

        dls_active_clear(dsp, B_TRUE);

        /*
         * If the dld_str_t is bound then unbind it.
         */
        if (dsp->ds_dlstate == DL_IDLE) {
                dls_unbind(dsp);
                dsp->ds_dlstate = DL_UNBOUND;
        }

        /*
         * If the MAC has been set in promiscuous mode then disable it.
         * This needs to be done before resetting ds_rx.
         */
        (void) dls_promisc(dsp, 0);
        old_flags = dsp->ds_promisc;
        dsp->ds_promisc = 0;
        (void) dls_promisc(dsp, old_flags);
```

```
124         /*
125          * At this point we have cutoff inbound packet flow from the mac
126          * for this 'dsp'. The dls_link_remove above cut off packets meant
127          * for us and waited for upcalls to finish. Similarly the dls_promisc
128          * reset above waited for promisc callbacks to finish. Now we can
129          * safely reset ds_rx to NULL
130          */
131         dsp->ds_rx = NULL;
132         dsp->ds_rx_arg = NULL;

134         dsp->ds_dlp = NULL;

136         if (!mac_capab_get(dsp->ds_mh, MAC_CAPAB_LEGACY, NULL))
137                 mac_stop(dsp->ds_mh);

139         /*
140          * Release our reference to the dls_link_t allowing that to be
141          * destroyed if there are no more dls_impl_t.
142          */
143         dls_link_rele(dlp);
144 }
```

*unchanged_portion_omitted_*

```
235 /*
236  * In order to prevent promiscuous-mode processing with dsp->ds_promisc
237  * set to inaccurate values, this function sets dsp->ds_promisc with new
238  * flags.  For enabling (mac_promisc_add), the flags are set prior to the
239  * actual enabling.  For disabling (mac_promisc_remove), the flags are set
240  * after the actual disabling.
241  */
242 int
243 dls_promisc(dld_str_t *dsp, uint32_t new_flags)
238 dls_promisc(dld_str_t *dsp, uint32_t old_flags)
244 {
245         int err = 0;
246         uint32_t old_flags = dsp->ds_promisc;

248         ASSERT(MAC_PERIM_HELD(dsp->ds_mh));
249         ASSERT(!(new_flags & ~(DLS_PROMISC_SAP | DLS_PROMISC_MULTI |
243         ASSERT(!(dsp->ds_promisc & ~(DLS_PROMISC_SAP | DLS_PROMISC_MULTI |
250             DLS_PROMISC_PHYS)));

252         if (dsp->ds_promisc == 0 && new_flags != 0) {
246         if (old_flags == 0 && dsp->ds_promisc != 0) {
253                 /*
254                  * If only DLS_PROMISC_SAP, we don't turn on the
255                  * physical promisc mode
256                  */
257                 dsp->ds_promisc = new_flags;
258                 err = mac_promisc_add(dsp->ds_mch, MAC_CLIENT_PROMISC_ALL,
259                     dls_rx_promisc, dsp, &dsp->ds_mph,
260                     (new_flags != DLS_PROMISC_SAP) ? 0 :
253                     (dsp->ds_promisc != DLS_PROMISC_SAP) ? 0 :
261                     MAC_PROMISC_FLAGS_NO_PHYS);
262                 if (err != 0) {
263                         dsp->ds_promisc = old_flags;
255                 if (err != 0)
264                         return (err);
265                 }

267                 /* Remove vlan promisc handle to avoid sending dup copy up */
268                 if (dsp->ds_vlan_mph != NULL) {
269                         mac_promisc_remove(dsp->ds_vlan_mph);
270                         dsp->ds_vlan_mph = NULL;
271                 }
272         } else if (dsp->ds_promisc != 0 && new_flags == 0) {
```



```
263         } else if (old_flags != 0 && dsp->ds_promisc == 0) {
273                 ASSERT(dsp->ds_mph != NULL);

275                 mac_promisc_remove(dsp->ds_mph);
276                 dsp->ds_promisc = new_flags;
277                 dsp->ds_mph = NULL;

279                 if (dsp->ds_sap == ETHERTYPE_VLAN &&
280                     dsp->ds_dlstate != DL_UNBOUND) {
271                         int err;

281                         if (dsp->ds_vlan_mph != NULL)
282                                 return (EINVAL);
283                         err = mac_promisc_add(dsp->ds_mch,
284                             MAC_CLIENT_PROMISC_ALL, dls_rx_vlan_promisc, dsp,
285                             &dsp->ds_vlan_mph, MAC_PROMISC_FLAGS_NO_PHYS);
278                         return (err);
286                 }
287         } else if (dsp->ds_promisc == DLS_PROMISC_SAP && new_flags != 0 &&
288             new_flags != dsp->ds_promisc) {
280         } else if (old_flags == DLS_PROMISC_SAP && dsp->ds_promisc != 0 &&
281             dsp->ds_promisc != old_flags) {
289                 /*
290                  * If the old flag is PROMISC_SAP, but the current flag has
291                  * changed to some new non-zero value, we need to turn the
292                  * physical promiscuous mode.
293                  */
294                 ASSERT(dsp->ds_mph != NULL);
295                 mac_promisc_remove(dsp->ds_mph);
296                 /* Honors both after-remove and before-add semantics! */
297                 dsp->ds_promisc = new_flags;
298                 err = mac_promisc_add(dsp->ds_mch, MAC_CLIENT_PROMISC_ALL,
299                     dls_rx_promisc, dsp, &dsp->ds_mph, 0);
300                 if (err != 0)
301                         dsp->ds_promisc = old_flags;
302         } else {
303                 /* No adding or removing, but record the new flags anyway. */
304                 dsp->ds_promisc = new_flags;
305         }

307         return (err);
308 }
```

*unchanged_portion_omitted_*