```
**********************************************************
   41276 Mon Jan 14 13:17:11 2019
new/usr/src/uts/common/inet/sctp/sctp_opt_data.c
10085 sctp_getpeeraddrs() doesn't need to check for a NULL primary
**********************************************************
```

```
/*
 * 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) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/stream.h>
#define _SUN_TPI_VERSION 2
#include <sys/tihdr.h>
#include <sys/socket.h>
#include <sys/xti_inet.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/kmem.h>
#include <sys/strsubr.h>
#include <sys/strsun.h>
#include <sys/policy.h>

#include <inet/common.h>
#include <netinet/ip6.h>
#include <inet/ip.h>
#include <inet/ip_ire.h>
#include <inet/ip_if.h>
#include <inet/proto_set.h>
#include <inet/ipclassifier.h>
#include <inet/ipsec_impl.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

#include <inet/common.h>
#include <inet/ip.h>
#include <inet/ip6.h>
#include <inet/sctp_itf.h>
#include "sctp_impl.h"
```



```
#include "sctp_asconf.h"
#include "sctp_addr.h"

static int      sctp_getpeeraddrs(sctp_t *, void *, int *);

static int
sctp_get_status(sctp_t *sctp, void *ptr)
{
        struct sctp_status *sstat = ptr;
        sctp_faddr_t *fp;
        struct sockaddr_in *sin;
        struct sockaddr_in6 *sin6;
        struct sctp_paddrinfo *sp;
        mblk_t *meta, *mp;
        int i;
        conn_t  *connp = sctp->sctp_connp;

        sstat->sstat_state = sctp->sctp_state;
        sstat->sstat_rwnd = sctp->sctp_frwnd;

        sp = &sstat->sstat_primary;
        if (!sctp->sctp_primary) {
                bzero(sp, sizeof (*sp));
                goto noprim;
        }
        fp = sctp->sctp_primary;

        if (fp->sf_isv4) {
                sin = (struct sockaddr_in *)&sp->spinfo_address;
                sin->sin_family = AF_INET;
                sin->sin_port = connp->conn_fport;
                IN6_V4MAPPED_TO_INADDR(&fp->sf_faddr, &sin->sin_addr);
                sp->spinfo_mtu = sctp->sctp_hdr_len;
        } else {
                sin6 = (struct sockaddr_in6 *)&sp->spinfo_address;
                sin6->sin6_family = AF_INET6;
                sin6->sin6_port = connp->conn_fport;
                sin6->sin6_addr = fp->sf_faddr;
                sp->spinfo_mtu = sctp->sctp_hdr6_len;
        }
        sp->spinfo_state = fp->sf_state == SCTP_FADDRS_ALIVE ? SCTP_ACTIVE :
            SCTP_INACTIVE;
        sp->spinfo_cwnd = fp->sf_cwnd;
        sp->spinfo_srtt = fp->sf_srtt;
        sp->spinfo_rto = fp->sf_rto;
        sp->spinfo_mtu += fp->sf_pmss;

noprim:
        sstat->sstat_unackdata = 0;
        sstat->sstat_penddata = 0;
        sstat->sstat_instrms = sctp->sctp_num_istr;
        sstat->sstat_outstrms = sctp->sctp_num_ostr;
        sstat->sstat_fragmentation_point = sctp->sctp_mss -
            sizeof (sctp_data_hdr_t);

        /* count unack'd */
        for (meta = sctp->sctp_xmit_head; meta; meta = meta->b_next) {
                for (mp = meta->b_cont; mp; mp = mp->b_next) {
                        if (!SCTP_CHUNK_ISSENT(mp)) {
                                break;
                        }
                        if (!SCTP_CHUNK_ISACKED(mp)) {
                                sstat->sstat_unackdata++;
                        }
                }
        }
```

```
 129         /*
 130          * Count penddata chunks. We can only count chunks in SCTP (not
 131          * data already delivered to socket layer).
 132          */
 133         if (sctp->sctp_instr != NULL) {
 134                 for (i = 0; i < sctp->sctp_num_istr; i++) {
 135                         for (meta = sctp->sctp_instr[i].istr_reass;
 136                             meta != NULL; meta = meta->b_next) {
 137                                 for (mp = meta->b_cont; mp; mp = mp->b_cont) {
 138                                         if (DB_TYPE(mp) != M_CTL) {
 139                                                 sstat->sstat_penddata++;
 140                                         }
 141                                 }
 142                         }
 143                 }
 144         }
 145         /* Un-Ordered Frag list */
 146         for (meta = sctp->sctp_uo_frags; meta != NULL; meta = meta->b_next)
 147                 sstat->sstat_penddata++;

 149         return (sizeof (*sstat));
 150 }
```

*unchanged_portion_omitted_*

```
1457 /*
1458  * Return a list of IP addresses of the peer endpoint of this sctp_t.
1459  * The parameter paddrs is supposed to be either (struct sockaddr_in *) or
1460  * (struct sockaddr_in6 *) depending on the address family of the sctp_t.
1461  */
1462 int
1463 sctp_getpeeraddrs(sctp_t *sctp, void *paddrs, int *addrcnt)
1464 {
1465         int                     family;
1466         struct sockaddr_in      *sin4;
1467         struct sockaddr_in6     *sin6;
1468         int                     max;
1469         int                     cnt;
1470         sctp_faddr_t            *fp = sctp->sctp_faddrs;
1471         in6_addr_t              addr;
1472         conn_t                  *connp = sctp->sctp_connp;

1474         ASSERT(sctp != NULL);

1476         if (sctp->sctp_faddrs == NULL)
1477                 return (ENOTCONN);

1479         family = connp->conn_family;
1480         max = *addrcnt;

1482         /* If we want only one, give the primary */
1483         if (max == 1) {
1484                 addr = sctp->sctp_primary->sf_faddr;
1485                 switch (family) {
1486                 case AF_INET:
1487                         sin4 = paddrs;
1488                         IN6_V4MAPPED_TO_INADDR(&addr, &sin4->sin_addr);
1489                         sin4->sin_port = connp->conn_fport;
1490                         sin4->sin_family = AF_INET;
1491                         break;

1493                 case AF_INET6:
1494                         sin6 = paddrs;
1495                         sin6->sin6_addr = addr;
1496                         sin6->sin6_port = connp->conn_fport;
1497                         sin6->sin6_family = AF_INET6;
```



```
1498                         sin6->sin6_flowinfo = connp->conn_flowinfo;
1499                         if (IN6_IS_ADDR_LINKSCOPE(&addr) &&
1496                             sctp->sctp_primary != NULL &&
1500                             (sctp->sctp_primary->sf_ixa->ixa_flags &
1501                             IXAF_SCOPEID_SET)) {
1502                                 sin6->sin6_scope_id =
1503                                     sctp->sctp_primary->sf_ixa->ixa_scopeid;
1504                         } else {
1505                                 sin6->sin6_scope_id = 0;
1506                         }
1507                         sin6->__sin6_src_id = 0;
1508                         break;
1509                 }
1510                 return (0);
1511         }

1513         for (cnt = 0; cnt < max && fp != NULL; cnt++, fp = fp->sf_next) {
1514                 addr = fp->sf_faddr;
1515                 switch (family) {
1516                 case AF_INET:
1517                         ASSERT(IN6_IS_ADDR_V4MAPPED(&addr));
1518                         sin4 = (struct sockaddr_in *)paddrs + cnt;
1519                         IN6_V4MAPPED_TO_INADDR(&addr, &sin4->sin_addr);
1520                         sin4->sin_port = connp->conn_fport;
1521                         sin4->sin_family = AF_INET;
1522                         break;
1523                 case AF_INET6:
1524                         sin6 = (struct sockaddr_in6 *)paddrs + cnt;
1525                         sin6->sin6_addr = addr;
1526                         sin6->sin6_port = connp->conn_fport;
1527                         sin6->sin6_family = AF_INET6;
1528                         sin6->sin6_flowinfo = connp->conn_flowinfo;
1529                         if (IN6_IS_ADDR_LINKSCOPE(&addr) &&
1530                             (fp->sf_ixa->ixa_flags & IXAF_SCOPEID_SET))
1531                                 sin6->sin6_scope_id = fp->sf_ixa->ixa_scopeid;
1532                         else
1533                                 sin6->sin6_scope_id = 0;
1534                         sin6->__sin6_src_id = 0;
1535                         break;
1536                 }
1537         }
1538         *addrcnt = cnt;
1539         return (0);
1540 }
```

*unchanged_portion_omitted_*