```
**********************************************************
  154105 Tue Jan 15 10:34:30 2019
new/usr/src/uts/common/inet/ip/icmp.c
10096 kstat update routines shouldn't check for NULL kstat
**********************************************************
/*
 * 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) 2013 by Delphix. All rights reserved.
 * Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */
/* Copyright (c) 1990 Mentat Inc. */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/strlog.h>
#include <sys/strsun.h>
#define _SUN_TPI_VERSION 2
#include <sys/tihdr.h>
#include <sys/timod.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/strsubr.h>
#include <sys/suntpi.h>
#include <sys/xti_inet.h>
#include <sys/cmn_err.h>
#include <sys/kmem.h>
#include <sys/cred.h>
#include <sys/policy.h>
#include <sys/priv.h>
#include <sys/ucred.h>
#include <sys/zone.h>

#include <sys/sockio.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/vtrace.h>
#include <sys/sdt.h>
#include <sys/debug.h>
#include <sys/isa_defs.h>
#include <sys/random.h>
#include <netinet/in.h>
#include <netinet/ip6.h>
#include <netinet/icmp6.h>
#include <netinet/udp.h>
```

```
#include <inet/common.h>
#include <inet/ip.h>
#include <inet/ip_impl.h>
#include <inet/ipsec_impl.h>
#include <inet/ip6.h>
#include <inet/ip_ire.h>
#include <inet/ip_if.h>
#include <inet/ip_multi.h>
#include <inet/ip_ndp.h>
#include <inet/proto_set.h>
#include <inet/mib2.h>
#include <inet/nd.h>
#include <inet/optcom.h>
#include <inet/snmpcom.h>
#include <inet/kstatcom.h>
#include <inet/ipclassifier.h>

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

#include <inet/rawip_impl.h>

#include <sys/disp.h>

/*
 * Synchronization notes:
 *
 * RAWIP is MT and uses the usual kernel synchronization primitives. We use
 * conn_lock to protect the icmp_t.
 *
 * Plumbing notes:
 * ICMP is always a device driver. For compatibility with mibopen() code
 * it is possible to I_PUSH "icmp", but that results in pushing a passthrough
 * dummy module.
 */
static void     icmp_addr_req(queue_t *q, mblk_t *mp);
static void     icmp_tpi_bind(queue_t *q, mblk_t *mp);
static void     icmp_bind_proto(icmp_t *icmp);
static int      icmp_build_hdr_template(conn_t *, const in6_addr_t *,
    const in6_addr_t *, uint32_t);
static void     icmp_capability_req(queue_t *q, mblk_t *mp);
static int      icmp_close(queue_t *q, int flags, cred_t *);
static void     icmp_close_free(conn_t *);
static void     icmp_tpi_connect(queue_t *q, mblk_t *mp);
static void     icmp_tpi_disconnect(queue_t *q, mblk_t *mp);
static void     icmp_err_ack(queue_t *q, mblk_t *mp, t_scalar_t t_error,
    int sys_error);
static void     icmp_err_ack_prim(queue_t *q, mblk_t *mp, t_scalar_t primitive,
    t_scalar_t tlierr, int sys_error);
static void     icmp_icmp_input(void *arg1, mblk_t *mp, void *arg2,
    ip_recv_attr_t *);
static void     icmp_icmp_error_ipv6(conn_t *connp, mblk_t *mp,
    ip_recv_attr_t *);
static void     icmp_info_req(queue_t *q, mblk_t *mp);
static void     icmp_input(void *, mblk_t *, void *, ip_recv_attr_t *);
static conn_t   *icmp_open(int family, cred_t *credp, int *err, int flags);
static int      icmp_openv4(queue_t *q, dev_t *devp, int flag, int sflag,
                    cred_t *credp);
static int      icmp_openv6(queue_t *q, dev_t *devp, int flag, int sflag,
                    cred_t *credp);
static boolean_t icmp_opt_allow_udr_set(t_scalar_t level, t_scalar_t name);
int             icmp_opt_set(conn_t *connp, uint_t optset_context,
                    int level, int name, uint_t inlen,
                    uchar_t *invalp, uint_t *outlenp, uchar_t *outvalp,
                    void *thisdg_attrs, cred_t *cr);
```

```
int             icmp_opt_get(conn_t *connp, int level, int name,
                    uchar_t *ptr);
static int      icmp_output_newdst(conn_t *connp, mblk_t *data_mp, sin_t *sin,
                    sin6_t *sin6, cred_t *cr, pid_t pid, ip_xmit_attr_t *ixa);
static mblk_t   *icmp_prepend_hdr(conn_t *, ip_xmit_attr_t *, const ip_pkt_t *,
    const in6_addr_t *, const in6_addr_t *, uint32_t, mblk_t *, int *);
static mblk_t   *icmp_prepend_header_template(conn_t *, ip_xmit_attr_t *,
    mblk_t *, const in6_addr_t *, uint32_t, int *);
static int      icmp_snmp_set(queue_t *q, t_scalar_t level, t_scalar_t name,
                    uchar_t *ptr, int len);
static void     icmp_ud_err(queue_t *q, mblk_t *mp, t_scalar_t err);
static void     icmp_tpi_unbind(queue_t *q, mblk_t *mp);
static int      icmp_wput(queue_t *q, mblk_t *mp);
static int      icmp_wput_fallback(queue_t *q, mblk_t *mp);
static void     icmp_wput_other(queue_t *q, mblk_t *mp);
static void     icmp_wput_iocdata(queue_t *q, mblk_t *mp);
static void     icmp_wput_restricted(queue_t *q, mblk_t *mp);
static void     icmp_ulp_recv(conn_t *, mblk_t *, uint_t);

static void     *rawip_stack_init(netstackid_t stackid, netstack_t *ns);
static void     rawip_stack_fini(netstackid_t stackid, void *arg);

static void     *rawip_kstat_init(netstackid_t stackid);
static void     rawip_kstat_fini(netstackid_t stackid, kstat_t *ksp);
static int      rawip_kstat_update(kstat_t *kp, int rw);
static void     rawip_stack_shutdown(netstackid_t stackid, void *arg);

/* Common routines for TPI and socket module */
static conn_t   *rawip_do_open(int, cred_t *, int *, int);
static void     rawip_do_close(conn_t *);
static int      rawip_do_bind(conn_t *, struct sockaddr *, socklen_t);
static int      rawip_do_unbind(conn_t *);
static int      rawip_do_connect(conn_t *, const struct sockaddr *, socklen_t,
    cred_t *, pid_t);

int             rawip_getsockname(sock_lower_handle_t, struct sockaddr *,
                    socklen_t *, cred_t *);
int             rawip_getpeername(sock_lower_handle_t, struct sockaddr *,
                    socklen_t *, cred_t *);

static struct module_info icmp_mod_info =  {
        5707, "icmp", 1, INFPSZ, 512, 128
};
```

______***unchanged_portion_omitted_***

```
static int
rawip_kstat_update(kstat_t *ksp, int rw)
{
        rawip_named_kstat_t *rawipkp;
        netstackid_t    stackid = (netstackid_t)(uintptr_t)ksp->ks_private;
        netstack_t      *ns;
        icmp_stack_t    *is;

        if (ksp->ks_data == NULL)
        if ((ksp == NULL) || (ksp->ks_data == NULL))
                return (EIO);

        if (rw == KSTAT_WRITE)
                return (EACCES);

        rawipkp = (rawip_named_kstat_t *)ksp->ks_data;

        ns = netstack_find_by_stackid(stackid);
        if (ns == NULL)
                return (-1);
        is = ns->netstack_icmp;
```

```
        if (is == NULL) {
                netstack_rele(ns);
                return (-1);
        }
        rawipkp->inDatagrams.value.ui32 =  is->is_rawip_mib.rawipInDatagrams;
        rawipkp->inCksumErrs.value.ui32 =  is->is_rawip_mib.rawipInCksumErrs;
        rawipkp->inErrors.value.ui32 =     is->is_rawip_mib.rawipInErrors;
        rawipkp->outDatagrams.value.ui32 = is->is_rawip_mib.rawipOutDatagrams;
        rawipkp->outErrors.value.ui32 =    is->is_rawip_mib.rawipOutErrors;
        netstack_rele(ns);
        return (0);
}
```

______***unchanged_portion_omitted_***

****
  449512 Tue Jan 15 10:34:30 2019
new/usr/src/uts/common/inet/ip/ip.c
10096 kstat update routines shouldn't check for NULL kstat
****
______unchanged_portion_omitted_

```
static int
ip_kstat_update(kstat_t *kp, int rw)
{
        ip_named_kstat_t *ipkp;
        mib2_ipIfStatsEntry_t ipmib;
        ill_walk_context_t ctx;
        ill_t *ill;
        netstackid_t    stackid = (zoneid_t)(uintptr_t)kp->ks_private;
        netstack_t      *ns;
        ip_stack_t      *ipst;

        if (kp->ks_data == NULL)
        if (kp == NULL || kp->ks_data == NULL)
                return (EIO);

        if (rw == KSTAT_WRITE)
                return (EACCES);

        ns = netstack_find_by_stackid(stackid);
        if (ns == NULL)
                return (-1);
        ipst = ns->netstack_ip;
        if (ipst == NULL) {
                netstack_rele(ns);
                return (-1);
        }
        ipkp = (ip_named_kstat_t *)kp->ks_data;

        bcopy(&ipst->ips_ip_mib, &ipmib, sizeof (ipmib));
        rw_enter(&ipst->ips_ill_g_lock, RW_READER);
        ill = ILL_START_WALK_V4(&ctx, ipst);
        for (; ill != NULL; ill = ill_next(&ctx, ill))
                ip_mib2_add_ip_stats(&ipmib, ill->ill_ip_mib);
        rw_exit(&ipst->ips_ill_g_lock);

        ipkp->forwarding.value.ui32 =           ipmib.ipIfStatsForwarding;
        ipkp->defaultTTL.value.ui32 =           ipmib.ipIfStatsDefaultTTL;
        ipkp->inReceives.value.ui64 =           ipmib.ipIfStatsHCInReceives;
        ipkp->inHdrErrors.value.ui32 =          ipmib.ipIfStatsInHdrErrors;
        ipkp->inAddrErrors.value.ui32 =         ipmib.ipIfStatsInAddrErrors;
        ipkp->forwDatagrams.value.ui64 = ipmib.ipIfStatsHCOutForwDatagrams;
        ipkp->inUnknownProtos.value.ui32 =      ipmib.ipIfStatsInUnknownProtos;
        ipkp->inDiscards.value.ui32 =           ipmib.ipIfStatsInDiscards;
        ipkp->inDelivers.value.ui64 =           ipmib.ipIfStatsHCInDelivers;
        ipkp->outRequests.value.ui64 =          ipmib.ipIfStatsHCOutRequests;
        ipkp->outDiscards.value.ui32 =          ipmib.ipIfStatsOutDiscards;
        ipkp->outNoRoutes.value.ui32 =          ipmib.ipIfStatsOutNoRoutes;
        ipkp->reasmTimeout.value.ui32 =         ipst->ips_ip_reassembly_timeout;
        ipkp->reasmReqds.value.ui32 =           ipmib.ipIfStatsReasmReqds;
        ipkp->reasmOKs.value.ui32 =             ipmib.ipIfStatsReasmOKs;
        ipkp->reasmFails.value.ui32 =           ipmib.ipIfStatsReasmFails;
        ipkp->fragOKs.value.ui32 =              ipmib.ipIfStatsOutFragOKs;
        ipkp->fragFails.value.ui32 =            ipmib.ipIfStatsOutFragFails;
        ipkp->fragCreates.value.ui32 =          ipmib.ipIfStatsOutFragCreates;

        ipkp->routingDiscards.value.ui32 =      0;
        ipkp->inErrs.value.ui32 =               ipmib.tcpIfStatsInErrs;
        ipkp->noPorts.value.ui32 =              ipmib.udpIfStatsNoPorts;
        ipkp->inCksumErrs.value.ui32 =          ipmib.ipIfStatsInCksumErrs;
```

```
        ipkp->reasmDuplicates.value.ui32 =      ipmib.ipIfStatsReasmDuplicates;
        ipkp->reasmPartDups.value.ui32 =        ipmib.ipIfStatsReasmPartDups;
        ipkp->forwProhibits.value.ui32 =        ipmib.ipIfStatsForwProhibits;
        ipkp->udpInCksumErrs.value.ui32 =       ipmib.udpIfStatsInCksumErrs;
        ipkp->udpInOverflows.value.ui32 =       ipmib.udpIfStatsInOverflows;
        ipkp->rawipInOverflows.value.ui32 =     ipmib.rawipIfStatsInOverflows;
        ipkp->ipsecInSucceeded.value.ui32 =     ipmib.ipsecIfStatsInSucceeded;
        ipkp->ipsecInFailed.value.i32 =         ipmib.ipsecIfStatsInFailed;

        ipkp->inIPv6.value.ui32 =       ipmib.ipIfStatsInWrongIPVersion;
        ipkp->outIPv6.value.ui32 =      ipmib.ipIfStatsOutWrongIPVersion;
        ipkp->outSwitchIPv6.value.ui32 = ipmib.ipIfStatsOutSwitchIPVersion;

        netstack_rele(ns);

        return (0);
}
```

______unchanged_portion_omitted_

```
static int
icmp_kstat_update(kstat_t *kp, int rw)
{
        icmp_named_kstat_t *icmpkp;
        netstackid_t    stackid = (zoneid_t)(uintptr_t)kp->ks_private;
        netstack_t      *ns;
        ip_stack_t      *ipst;

        if (kp->ks_data == NULL)
        if ((kp == NULL) || (kp->ks_data == NULL))
                return (EIO);

        if (rw == KSTAT_WRITE)
                return (EACCES);

        ns = netstack_find_by_stackid(stackid);
        if (ns == NULL)
                return (-1);
        ipst = ns->netstack_ip;
        if (ipst == NULL) {
                netstack_rele(ns);
                return (-1);
        }
        icmpkp = (icmp_named_kstat_t *)kp->ks_data;

        icmpkp->inMsgs.value.ui32 =         ipst->ips_icmp_mib.icmpInMsgs;
        icmpkp->inErrors.value.ui32 =       ipst->ips_icmp_mib.icmpInErrors;
        icmpkp->inDestUnreachs.value.ui32 =
            ipst->ips_icmp_mib.icmpInDestUnreachs;
        icmpkp->inTimeExcds.value.ui32 =    ipst->ips_icmp_mib.icmpInTimeExcds;
        icmpkp->inParmProbs.value.ui32 =    ipst->ips_icmp_mib.icmpInParmProbs;
        icmpkp->inSrcQuenchs.value.ui32 =   ipst->ips_icmp_mib.icmpInSrcQuenchs;
        icmpkp->inRedirects.value.ui32 =    ipst->ips_icmp_mib.icmpInRedirects;
        icmpkp->inEchos.value.ui32 =        ipst->ips_icmp_mib.icmpInEchos;
        icmpkp->inEchoReps.value.ui32 =     ipst->ips_icmp_mib.icmpInEchoReps;
        icmpkp->inTimestamps.value.ui32 =   ipst->ips_icmp_mib.icmpInTimestamps;
        icmpkp->inTimestampReps.value.ui32 =
            ipst->ips_icmp_mib.icmpInTimestampReps;
        icmpkp->inAddrMasks.value.ui32 =    ipst->ips_icmp_mib.icmpInAddrMasks;
        icmpkp->inAddrMaskReps.value.ui32 =
            ipst->ips_icmp_mib.icmpInAddrMaskReps;
        icmpkp->outMsgs.value.ui32 =        ipst->ips_icmp_mib.icmpOutMsgs;
        icmpkp->outErrors.value.ui32 =      ipst->ips_icmp_mib.icmpOutErrors;
        icmpkp->outDestUnreachs.value.ui32 =
            ipst->ips_icmp_mib.icmpOutDestUnreachs;
        icmpkp->outTimeExcds.value.ui32 =   ipst->ips_icmp_mib.icmpOutTimeExcds;
        icmpkp->outParmProbs.value.ui32 =   ipst->ips_icmp_mib.icmpOutParmProbs;
```

```
	icmpkp->outSrcQuenchs.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutSrcQuenchs;
	icmpkp->outRedirects.value.ui32 =	ipst->ips_icmp_mib.icmpOutRedirects;
	icmpkp->outEchos.value.ui32 =		ipst->ips_icmp_mib.icmpOutEchos;
	icmpkp->outEchoReps.value.ui32 =	ipst->ips_icmp_mib.icmpOutEchoReps;
	icmpkp->outTimestamps.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutTimestamps;
	icmpkp->outTimestampReps.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutTimestampReps;
	icmpkp->outAddrMasks.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutAddrMasks;
	icmpkp->outAddrMaskReps.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutAddrMaskReps;
	icmpkp->inCksumErrs.value.ui32 =	ipst->ips_icmp_mib.icmpInCksumErrs;
	icmpkp->inUnknowns.value.ui32 =		ipst->ips_icmp_mib.icmpInUnknowns;
	icmpkp->inFragNeeded.value.ui32 =	ipst->ips_icmp_mib.icmpInFragNeeded;
	icmpkp->outFragNeeded.value.ui32 =
	    ipst->ips_icmp_mib.icmpOutFragNeeded;
	icmpkp->outDrops.value.ui32 =		ipst->ips_icmp_mib.icmpOutDrops;
	icmpkp->inOverflows.value.ui32 =	ipst->ips_icmp_mib.icmpInOverflows;
	icmpkp->inBadRedirects.value.ui32 =
	    ipst->ips_icmp_mib.icmpInBadRedirects;

	netstack_rele(ns);
	return (0);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   31576 Tue Jan 15 10:34:31 2019
new/usr/src/uts/common/inet/sctp/sctp_snmp.c
10096 kstat update routines shouldn't check for NULL kstat
**********************************************************
/*
 * 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>
#include <sys/cmn_err.h>
#define _SUN_TPI_VERSION 2
#include <sys/tihdr.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/tsol/tndb.h>

#include <netinet/in.h>

#include <inet/common.h>
#include <inet/ip.h>
#include <inet/mib2.h>
#include <inet/snmpcom.h>
#include <inet/kstatcom.h>
#include <inet/ipclassifier.h>
#include "sctp_impl.h"
#include "sctp_addr.h"

static void sctp_clr_kstats2(sctp_kstat_t *);
static void sctp_add_kstats2(sctp_kstat_counter_t *, sctp_kstat_t *);
static int sctp_snmp_state(sctp_t *);
static void sctp_sum_mib(sctp_stack_t *, mib2_sctp_t *);
static void sctp_add_mib(mib2_sctp_t *, mib2_sctp_t *);

static int
sctp_kstat_update(kstat_t *kp, int rw)
{
        sctp_named_kstat_t      *sctpkp;
        sctp_t                  *sctp, *sctp_prev;
        zoneid_t        myzoneid;
```

```
        netstackid_t    stackid = (netstackid_t)(uintptr_t)kp->ks_private;
        netstack_t      *ns;
        sctp_stack_t    *sctps;
        mib2_sctp_t     sctp_mib;

        if (kp->ks_data == NULL)
        if (kp == NULL|| kp->ks_data == NULL)
                return (EIO);

        if (rw == KSTAT_WRITE)
                return (EACCES);

        ns = netstack_find_by_stackid(stackid);
        if (ns == NULL)
                return (-1);
        sctps = ns->netstack_sctp;
        if (sctps == NULL) {
                netstack_rele(ns);
                return (-1);
        }

        /*
         * For all exclusive netstacks, the zone ID is always GLOBAL_ZONEID.
         */
        if (stackid != GLOBAL_NETSTACKID)
                myzoneid = GLOBAL_ZONEID;
        else
                myzoneid = curproc->p_zone->zone_id;

        bzero(&sctp_mib, sizeof (sctp_mib));

        /*
         * Get the number of current associations and gather their
         * individual set of statistics.
         */
        sctp_prev = NULL;
        mutex_enter(&sctps->sctps_g_lock);
        sctp = list_head(&sctps->sctps_g_list);
        while (sctp != NULL) {
                mutex_enter(&sctp->sctp_reflock);
                if (sctp->sctp_condemned) {
                        mutex_exit(&sctp->sctp_reflock);
                        sctp = list_next(&sctps->sctps_g_list, sctp);
                        continue;
                }
                sctp->sctp_refcnt++;
                mutex_exit(&sctp->sctp_reflock);
                mutex_exit(&sctps->sctps_g_lock);
                if (sctp_prev != NULL)
                        SCTP_REFRELE(sctp_prev);
                if (sctp->sctp_connp->conn_zoneid != myzoneid)
                        goto next_sctp;
                if (sctp->sctp_state == SCTPS_ESTABLISHED ||
                    sctp->sctp_state == SCTPS_SHUTDOWN_PENDING ||
                    sctp->sctp_state == SCTPS_SHUTDOWN_RECEIVED) {
                        /*
                         * Just bump the local sctp_mib.  The number of
                         * existing associations is not kept in kernel.
                         */
                        BUMP_MIB(&sctp_mib, sctpCurrEstab);
                }

                if (sctp->sctp_opkts) {
                        SCTPS_UPDATE_MIB(sctps, sctpOutSCTPPkts,
                            sctp->sctp_opkts);
                        sctp->sctp_opkts = 0;
```

```
		}

		if (sctp->sctp_obchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpOutCtrlChunks,
			    sctp->sctp_obchunks);
			UPDATE_LOCAL(sctp->sctp_cum_obchunks,
			    sctp->sctp_obchunks);
			sctp->sctp_obchunks = 0;
		}

		if (sctp->sctp_odchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpOutOrderChunks,
			    sctp->sctp_odchunks);
			UPDATE_LOCAL(sctp->sctp_cum_odchunks,
			    sctp->sctp_odchunks);
			sctp->sctp_odchunks = 0;
		}

		if (sctp->sctp_oudchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpOutUnorderChunks,
			    sctp->sctp_oudchunks);
			UPDATE_LOCAL(sctp->sctp_cum_oudchunks,
			    sctp->sctp_oudchunks);
			sctp->sctp_oudchunks = 0;
		}

		if (sctp->sctp_rxtchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpRetransChunks,
			    sctp->sctp_rxtchunks);
			UPDATE_LOCAL(sctp->sctp_cum_rxtchunks,
			    sctp->sctp_rxtchunks);
			sctp->sctp_rxtchunks = 0;
		}

		if (sctp->sctp_ipkts) {
			SCTPS_UPDATE_MIB(sctps, sctpInSCTPPkts,
			    sctp->sctp_ipkts);
			sctp->sctp_ipkts = 0;
		}

		if (sctp->sctp_ibchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpInCtrlChunks,
			    sctp->sctp_ibchunks);
			UPDATE_LOCAL(sctp->sctp_cum_ibchunks,
			    sctp->sctp_ibchunks);
			sctp->sctp_ibchunks = 0;
		}

		if (sctp->sctp_idchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpInOrderChunks,
			    sctp->sctp_idchunks);
			UPDATE_LOCAL(sctp->sctp_cum_idchunks,
			    sctp->sctp_idchunks);
			sctp->sctp_idchunks = 0;
		}

		if (sctp->sctp_iudchunks) {
			SCTPS_UPDATE_MIB(sctps, sctpInUnorderChunks,
			    sctp->sctp_iudchunks);
			UPDATE_LOCAL(sctp->sctp_cum_iudchunks,
			    sctp->sctp_iudchunks);
			sctp->sctp_iudchunks = 0;
		}

		if (sctp->sctp_fragdmsgs) {
			SCTPS_UPDATE_MIB(sctps, sctpFragUsrMsgs,
```

```
			    sctp->sctp_fragdmsgs);
			sctp->sctp_fragdmsgs = 0;
		}

		if (sctp->sctp_reassmsgs) {
			SCTPS_UPDATE_MIB(sctps, sctpReasmUsrMsgs,
			    sctp->sctp_reassmsgs);
			sctp->sctp_reassmsgs = 0;
		}

next_sctp:
		sctp_prev = sctp;
		mutex_enter(&sctps->sctps_g_lock);
		sctp = list_next(&sctps->sctps_g_list, sctp);
	}
	mutex_exit(&sctps->sctps_g_lock);
	if (sctp_prev != NULL)
		SCTP_REFRELE(sctp_prev);

	sctp_sum_mib(sctps, &sctp_mib);

	/* Copy data from the SCTP MIB */
	sctpkp = (sctp_named_kstat_t *)kp->ks_data;

	/* These are from global ndd params. */
	sctpkp->sctpRtoMin.value.ui32 = sctps->sctps_rto_ming;
	sctpkp->sctpRtoMax.value.ui32 = sctps->sctps_rto_maxg;
	sctpkp->sctpRtoInitial.value.ui32 = sctps->sctps_rto_initialg;
	sctpkp->sctpValCookieLife.value.ui32 = sctps->sctps_cookie_life;
	sctpkp->sctpMaxInitRetr.value.ui32 = sctps->sctps_max_init_retr;

	/* Copy data from the local sctp_mib to the provided kstat. */
	sctpkp->sctpCurrEstab.value.i32 = sctp_mib.sctpCurrEstab;
	sctpkp->sctpActiveEstab.value.i32 = sctp_mib.sctpActiveEstab;
	sctpkp->sctpPassiveEstab.value.i32 = sctp_mib.sctpPassiveEstab;
	sctpkp->sctpAborted.value.i32 = sctp_mib.sctpAborted;
	sctpkp->sctpShutdowns.value.i32 = sctp_mib.sctpShutdowns;
	sctpkp->sctpOutOfBlue.value.i32 = sctp_mib.sctpOutOfBlue;
	sctpkp->sctpChecksumError.value.i32 = sctp_mib.sctpChecksumError;
	sctpkp->sctpOutCtrlChunks.value.i64 = sctp_mib.sctpOutCtrlChunks;
	sctpkp->sctpOutOrderChunks.value.i64 = sctp_mib.sctpOutOrderChunks;
	sctpkp->sctpOutUnorderChunks.value.i64 = sctp_mib.sctpOutUnorderChunks;
	sctpkp->sctpRetransChunks.value.i64 = sctp_mib.sctpRetransChunks;
	sctpkp->sctpOutAck.value.i32 = sctp_mib.sctpOutAck;
	sctpkp->sctpOutAckDelayed.value.i32 = sctp_mib.sctpOutAckDelayed;
	sctpkp->sctpOutWinUpdate.value.i32 = sctp_mib.sctpOutWinUpdate;
	sctpkp->sctpOutFastRetrans.value.i32 = sctp_mib.sctpOutFastRetrans;
	sctpkp->sctpOutWinProbe.value.i32 = sctp_mib.sctpOutWinProbe;
	sctpkp->sctpInCtrlChunks.value.i64 = sctp_mib.sctpInCtrlChunks;
	sctpkp->sctpInOrderChunks.value.i64 = sctp_mib.sctpInOrderChunks;
	sctpkp->sctpInUnorderChunks.value.i64 = sctp_mib.sctpInUnorderChunks;
	sctpkp->sctpInAck.value.i32 = sctp_mib.sctpInAck;
	sctpkp->sctpInDupAck.value.i32 = sctp_mib.sctpInDupAck;
	sctpkp->sctpInAckUnsent.value.i32 = sctp_mib.sctpInAckUnsent;
	sctpkp->sctpFragUsrMsgs.value.i64 = sctp_mib.sctpFragUsrMsgs;
	sctpkp->sctpReasmUsrMsgs.value.i64 = sctp_mib.sctpReasmUsrMsgs;
	sctpkp->sctpOutSCTPPkts.value.i64 = sctp_mib.sctpOutSCTPPkts;
	sctpkp->sctpInSCTPPkts.value.i64 = sctp_mib.sctpInSCTPPkts;
	sctpkp->sctpInInvalidCookie.value.i32 = sctp_mib.sctpInInvalidCookie;
	sctpkp->sctpTimRetrans.value.i32 = sctp_mib.sctpTimRetrans;
	sctpkp->sctpTimRetransDrop.value.i32 = sctp_mib.sctpTimRetransDrop;
	sctpkp->sctpTimHeartBeatProbe.value.i32 =
	    sctp_mib.sctpTimHeartBeatProbe;
	sctpkp->sctpTimHeartBeatDrop.value.i32 = sctp_mib.sctpTimHeartBeatDrop;
	sctpkp->sctpListenDrop.value.i32 = sctp_mib.sctpListenDrop;
	sctpkp->sctpInClosed.value.i32 = sctp_mib.sctpInClosed;
```

```
        netstack_rele(ns);
        return (0);
}
```
______***unchanged_portion_omitted_***