

\*\*\*\*\*

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  
 \*\*\*\*\*

```

1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2013 by Delphix. All rights reserved.
24 * Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
25 * Copyright (c) 2018, Joyent, Inc.
26 */
27 /* Copyright (c) 1990 Mentat Inc. */

29 #include <sys/types.h>
30 #include <sys/stream.h>
31 #include <sys/stropts.h>
32 #include <sys/strlog.h>
33 #include <sys/strsun.h>
34 #define _SUN_TPI_VERSION 2
35 #include <sys/tihdr.h>
36 #include <sys/timod.h>
37 #include <sys/ddi.h>
38 #include <sys/sunddi.h>
39 #include <sys/strsubr.h>
40 #include <sys/suntpi.h>
41 #include <sys/xti_inet.h>
42 #include <sys/cmn_err.h>
43 #include <sys/kmem.h>
44 #include <sys/cred.h>
45 #include <sys/policy.h>
46 #include <sys/priv.h>
47 #include <sys/ucred.h>
48 #include <sys/zone.h>

50 #include <sys/sockio.h>
51 #include <sys/socket.h>
52 #include <sys/socketvar.h>
53 #include <sys/vtrace.h>
54 #include <sys/sdt.h>
55 #include <sys/debug.h>
56 #include <sys/isa_defs.h>
57 #include <sys/random.h>
58 #include <netinet/in.h>
59 #include <netinet/ip6.h>
60 #include <netinet/icmp6.h>
61 #include <netinet/udp.h>

```

```

63 #include <inet/common.h>
64 #include <inet/ip.h>
65 #include <inet/ip_impl.h>
66 #include <inet/ipsec_impl.h>
67 #include <inet/ip6.h>
68 #include <inet/ip_ire.h>
69 #include <inet/ip_if.h>
70 #include <inet/ip_multi.h>
71 #include <inet/ip_ndp.h>
72 #include <inet/proto_set.h>
73 #include <inet/mib2.h>
74 #include <inet/nd.h>
75 #include <inet/optcom.h>
76 #include <inet/snmpcom.h>
77 #include <inet/kstatcom.h>
78 #include <inet/ipclassifier.h>

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

83 #include <inet/rawip_impl.h>

85 #include <sys/disp.h>

87 /*
88 * Synchronization notes:
89 *
90 * RAWIP is MT and uses the usual kernel synchronization primitives. We use
91 * conn_lock to protect the icmp_t.
92 *
93 * Plumbing notes:
94 * ICMP is always a device driver. For compatibility with mibopen() code
95 * it is possible to I_PUSH "icmp", but that results in pushing a passthrough
96 * dummy module.
97 */
98 static void icmp_addr_req(queue_t *q, mblk_t *mp);
99 static void icmp_tpi_bind(queue_t *q, mblk_t *mp);
100 static void icmp_bind_proto(icmp_t *icmp);
101 static int icmp_build_hdr_template(conn_t *, const in6_addr_t *,
102 const in6_addr_t *, uint32_t);
103 static void icmp_capability_req(queue_t *q, mblk_t *mp);
104 static int icmp_close(queue_t *q, int flags, cred_t *);
105 static void icmp_close_free(conn_t *);
106 static void icmp_tpi_connect(queue_t *q, mblk_t *mp);
107 static void icmp_tpi_disconnect(queue_t *q, mblk_t *mp);
108 static void icmp_err_ack(queue_t *q, mblk_t *mp, t_scalar_t t_error,
109 int sys_error);
110 static void icmp_err_ack_prim(queue_t *q, mblk_t *mp, t_scalar_t primitive,
111 t_scalar_t tlierr, int sys_error);
112 static void icmp_icmp_input(void *arg1, mblk_t *mp, void *arg2,
113 ip_recv_attr_t *);
114 static void icmp_icmp_error_ipv6(conn_t *connp, mblk_t *mp,
115 ip_recv_attr_t *);
116 static void icmp_info_req(queue_t *q, mblk_t *mp);
117 static void icmp_input(void *, mblk_t *, void *, ip_recv_attr_t *);
118 static conn_t *icmp_open(int family, cred_t *credp, int *err, int flags);
119 static int icmp_openv4(queue_t *q, dev_t *devp, int flag, int sflag,
120 cred_t *credp);
121 static int icmp_openv6(queue_t *q, dev_t *devp, int flag, int sflag,
122 cred_t *credp);
123 static boolean_t icmp_opt_allow_udr_set(t_scalar_t level, t_scalar_t name);
124 int icmp_opt_set(conn_t *connp, uint_t optset_context,
125 int level, int name, uint_t inlen,
126 uchar_t *invalp, uint_t *outlenp, uchar_t *outvalp,
127 void *thisdg_attrs, cred_t *cr);

```

```

128 int          icmp_opt_get(conn_t *connp, int level, int name,
129             uchar_t *ptr);
130 static int    icmp_output_newdst(conn_t *connp, mblk_t *data_mp, sin_t *sin,
131             sin6_t *sin6, cred_t *cr, pid_t pid, ip_xmit_attr_t *ixa);
132 static mblk_t *icmp_prepend_hdr(conn_t *, ip_xmit_attr_t *, const ip_pkt_t *,
133             const in6_addr_t *, const in6_addr_t *, uint32_t, mblk_t *, int *);
134 static mblk_t *icmp_prepend_header_template(conn_t *, ip_xmit_attr_t *,
135             mblk_t *, const in6_addr_t *, uint32_t, int *);
136 static int    icmp_snmpp_set(queue_t *q, t_scalar_t level, t_scalar_t name,
137             uchar_t *ptr, int len);
138 static void   icmp_ud_err(queue_t *q, mblk_t *mp, t_scalar_t err);
139 static void   icmp_tpi_unbind(queue_t *q, mblk_t *mp);
140 static int    icmp_wput(queue_t *q, mblk_t *mp);
141 static int    icmp_wput_fallback(queue_t *q, mblk_t *mp);
142 static void   icmp_wput_other(queue_t *q, mblk_t *mp);
143 static void   icmp_wput_ioCDATA(queue_t *q, mblk_t *mp);
144 static void   icmp_wput_restricted(queue_t *q, mblk_t *mp);
145 static void   icmp_ulp_rcv(conn_t *, mblk_t *, uint_t);

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

150 static void   *rawip_kstat_init(netstackid_t stackid);
151 static void   rawip_kstat_fini(netstackid_t stackid, kstat_t *ksp);
152 static int    rawip_kstat_update(kstat_t *ksp, int rw);
153 static void   rawip_stack_shutdown(netstackid_t stackid, void *arg);

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

163 int           rawip_getsockname(sock_lower_handle_t, struct sockaddr *,
164             socklen_t *, cred_t *);
165 int           rawip_getpeername(sock_lower_handle_t, struct sockaddr *,
166             socklen_t *, cred_t *);

168 static struct module_info icmp_mod_info = {
169     5707, "icmp", 1, INFPSZ, 512, 128
170 };

```

```

5088             if (is == NULL) {
5089                 netstack_rele(ns);
5090                 return (-1);
5091             }
5092             rawipkp->inDatagrams.value.ui32 = is->is_rawip_mib.rawipInDatagrams;
5093             rawipkp->inChecksumErrs.value.ui32 = is->is_rawip_mib.rawipInChecksumErrs;
5094             rawipkp->inErrors.value.ui32 = is->is_rawip_mib.rawipInErrors;
5095             rawipkp->outDatagrams.value.ui32 = is->is_rawip_mib.rawipOutDatagrams;
5096             rawipkp->outErrors.value.ui32 = is->is_rawip_mib.rawipOutErrors;
5097             netstack_rele(ns);
5098             return (0);
5099 }

```

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\_\_\_\_\_

```

14067 static int
14068 ip_kstat_update(kstat_t *kp, int rw)
14069 {
14070     ip_named_kstat_t *ipkp;
14071     mib2_ipIfStatsEntry_t ipmib;
14072     ill_walk_context_t ctx;
14073     ill_t *ill;
14074     netstackid_t stackid = (zoneid_t)(uintptr_t)kp->ks_private;
14075     netstack_t *ns;
14076     ip_stack_t *ipst;

14078     if (kp->ks_data == NULL)
14078     if (kp == NULL || kp->ks_data == NULL)
14079         return (EIO);

14081     if (rw == KSTAT_WRITE)
14082         return (EACCES);

14084     ns = netstack_find_by_stackid(stackid);
14085     if (ns == NULL)
14086         return (-1);
14087     ipst = ns->netstack_ip;
14088     if (ipst == NULL) {
14089         netstack_rele(ns);
14090         return (-1);
14091     }
14092     ipkp = (ip_named_kstat_t *)kp->ks_data;

14094     bcopy(&ipst->ips_ip_mib, &ipmib, sizeof(ipmib));
14095     rw_enter(&ipst->ips_ill_g_lock, RW_READER);
14096     ill = ILL_START_WALK_V4(&ctx, ipst);
14097     for (; ill != NULL; ill = ill_next(&ctx, ill))
14098         ip_mib2_add_ip_stats(&ipmib, ill->ill_ip_mib);
14099     rw_exit(&ipst->ips_ill_g_lock);

14101     ipkp->forwarding.value.ui32 = ipmib.ipIfStatsForwarding;
14102     ipkp->defaultTTL.value.ui32 = ipmib.ipIfStatsDefaultTTL;
14103     ipkp->inReceives.value.ui64 = ipmib.ipIfStatsHCInReceives;
14104     ipkp->inHdrErrors.value.ui32 = ipmib.ipIfStatsInHdrErrors;
14105     ipkp->inAddrErrors.value.ui32 = ipmib.ipIfStatsInAddrErrors;
14106     ipkp->forwDatagrams.value.ui64 = ipmib.ipIfStatsHCOutForwDatagrams;
14107     ipkp->inUnknownProtos.value.ui32 = ipmib.ipIfStatsInUnknownProtos;
14108     ipkp->inDiscards.value.ui32 = ipmib.ipIfStatsInDiscards;
14109     ipkp->inDelivers.value.ui64 = ipmib.ipIfStatsHCInDelivers;
14110     ipkp->outRequests.value.ui64 = ipmib.ipIfStatsOutRequests;
14111     ipkp->outDiscards.value.ui32 = ipmib.ipIfStatsOutDiscards;
14112     ipkp->outNoRoutes.value.ui32 = ipmib.ipIfStatsOutNoRoutes;
14113     ipkp->reasmTimeout.value.ui32 = ipst->ips_ip_reassembly_timeout;
14114     ipkp->reasmReqds.value.ui32 = ipmib.ipIfStatsReasmReqds;
14115     ipkp->reasmOKs.value.ui32 = ipmib.ipIfStatsReasmOKs;
14116     ipkp->reasmFails.value.ui32 = ipmib.ipIfStatsReasmFails;
14117     ipkp->fragOKs.value.ui32 = ipmib.ipIfStatsOutFragOKs;
14118     ipkp->fragFails.value.ui32 = ipmib.ipIfStatsOutFragFails;
14119     ipkp->fragCreates.value.ui32 = ipmib.ipIfStatsOutFragCreates;

14121     ipkp->routingDiscards.value.ui32 = 0;
14122     ipkp->inErrs.value.ui32 = ipmib.tcpIfStatsInErrs;
14123     ipkp->noPorts.value.ui32 = ipmib.udpIfStatsNoPorts;
14124     ipkp->inChecksumErrs.value.ui32 = ipmib.ipIfStatsInChecksumErrs;

```

```

14125     ipkp->reasmDuplicates.value.ui32 = ipmib.ipIfStatsReasmDuplicates;
14126     ipkp->reasmPartDups.value.ui32 = ipmib.ipIfStatsReasmPartDups;
14127     ipkp->forwProhibits.value.ui32 = ipmib.ipIfStatsForwProhibits;
14128     ipkp->udpInChecksumErrs.value.ui32 = ipmib.udpIfStatsInChecksumErrs;
14129     ipkp->udpInOverflows.value.ui32 = ipmib.udpIfStatsInOverflows;
14130     ipkp->rawipInOverflows.value.ui32 = ipmib.rawipIfStatsInOverflows;
14131     ipkp->ipsecInSucceeded.value.ui32 = ipmib.ipsecIfStatsInSucceeded;
14132     ipkp->ipsecInFailed.value.i32 = ipmib.ipsecIfStatsInFailed;

14134     ipkp->inIPv6.value.ui32 = ipmib.ipIfStatsInWrongIPVersion;
14135     ipkp->outIPv6.value.ui32 = ipmib.ipIfStatsOutWrongIPVersion;
14136     ipkp->outSwitchIPv6.value.ui32 = ipmib.ipIfStatsOutSwitchIPVersion;

14138     netstack_rele(ns);

14140     return (0);
14141 }
_____unchanged_portion_omitted_____

14207 static int
14208 icmp_kstat_update(kstat_t *kp, int rw)
14209 {
14210     icmp_named_kstat_t *icmpkp;
14211     netstackid_t stackid = (zoneid_t)(uintptr_t)kp->ks_private;
14212     netstack_t *ns;
14213     ip_stack_t *ipst;

14215     if (kp->ks_data == NULL)
14215     if ((kp == NULL) || (kp->ks_data == NULL))
14216         return (EIO);

14218     if (rw == KSTAT_WRITE)
14219         return (EACCES);

14221     ns = netstack_find_by_stackid(stackid);
14222     if (ns == NULL)
14223         return (-1);
14224     ipst = ns->netstack_ip;
14225     if (ipst == NULL) {
14226         netstack_rele(ns);
14227         return (-1);
14228     }
14229     icmpkp = (icmp_named_kstat_t *)kp->ks_data;

14231     icmpkp->inMsgs.value.ui32 = ipst->ips_icmp_mib.icmpInMsgs;
14232     icmpkp->inErrors.value.ui32 = ipst->ips_icmp_mib.icmpInErrors;
14233     icmpkp->inDestUnreachs.value.ui32 = ipst->ips_icmp_mib.icmpInDestUnreachs;
14234     ipst->ips_icmp_mib.icmpInDestUnreachs;
14235     icmpkp->inTimeExcds.value.ui32 = ipst->ips_icmp_mib.icmpInTimeExcds;
14236     icmpkp->inParmProbs.value.ui32 = ipst->ips_icmp_mib.icmpInParmProbs;
14237     icmpkp->inSrcQuenchs.value.ui32 = ipst->ips_icmp_mib.icmpInSrcQuenchs;
14238     icmpkp->inRedirects.value.ui32 = ipst->ips_icmp_mib.icmpInRedirects;
14239     icmpkp->inEchos.value.ui32 = ipst->ips_icmp_mib.icmpInEchos;
14240     icmpkp->inEchoReps.value.ui32 = ipst->ips_icmp_mib.icmpInEchoReps;
14241     icmpkp->inTimestamps.value.ui32 = ipst->ips_icmp_mib.icmpInTimestamps;
14242     icmpkp->inTimestampReps.value.ui32 = ipst->ips_icmp_mib.icmpInTimestampReps;
14243     ipst->ips_icmp_mib.icmpInTimestampReps;
14244     icmpkp->inAddrMasks.value.ui32 = ipst->ips_icmp_mib.icmpInAddrMasks;
14245     icmpkp->inAddrMaskReps.value.ui32 = ipst->ips_icmp_mib.icmpInAddrMaskReps;
14246     ipst->ips_icmp_mib.icmpInAddrMaskReps;
14247     icmpkp->outMsgs.value.ui32 = ipst->ips_icmp_mib.icmpOutMsgs;
14248     icmpkp->outErrors.value.ui32 = ipst->ips_icmp_mib.icmpOutErrors;
14249     icmpkp->outDestUnreachs.value.ui32 = ipst->ips_icmp_mib.icmpOutDestUnreachs;
14250     ipst->ips_icmp_mib.icmpOutDestUnreachs;
14251     icmpkp->outTimeExcds.value.ui32 = ipst->ips_icmp_mib.icmpOutTimeExcds;
14252     icmpkp->outParmProbs.value.ui32 = ipst->ips_icmp_mib.icmpOutParmProbs;

```

```
14253     icmpkp->outSrcQuenchs.value.ui32 =
14254         ipst->ips_icmp_mib.icmpOutSrcQuenchs;
14255     icmpkp->outRedirects.value.ui32 = ipst->ips_icmp_mib.icmpOutRedirects;
14256     icmpkp->outEchos.value.ui32 = ipst->ips_icmp_mib.icmpOutEchos;
14257     icmpkp->outEchoReps.value.ui32 = ipst->ips_icmp_mib.icmpOutEchoReps;
14258     icmpkp->outTimestamps.value.ui32 =
14259         ipst->ips_icmp_mib.icmpOutTimestamps;
14260     icmpkp->outTimestampReps.value.ui32 =
14261         ipst->ips_icmp_mib.icmpOutTimestampReps;
14262     icmpkp->outAddrMasks.value.ui32 =
14263         ipst->ips_icmp_mib.icmpOutAddrMasks;
14264     icmpkp->outAddrMaskReps.value.ui32 =
14265         ipst->ips_icmp_mib.icmpOutAddrMaskReps;
14266     icmpkp->inCksumErrs.value.ui32 = ipst->ips_icmp_mib.icmpInCksumErrs;
14267     icmpkp->inUnknowns.value.ui32 = ipst->ips_icmp_mib.icmpInUnknowns;
14268     icmpkp->inFragNeeded.value.ui32 = ipst->ips_icmp_mib.icmpInFragNeeded;
14269     icmpkp->outFragNeeded.value.ui32 =
14270         ipst->ips_icmp_mib.icmpOutFragNeeded;
14271     icmpkp->outDrops.value.ui32 = ipst->ips_icmp_mib.icmpOutDrops;
14272     icmpkp->inOverflows.value.ui32 = ipst->ips_icmp_mib.icmpInOverflows;
14273     icmpkp->inBadRedirects.value.ui32 =
14274         ipst->ips_icmp_mib.icmpInBadRedirects;

14276     netstack_rele(ns);
14277     return (0);
14278 }
```

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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /*
27  * Copyright (c) 2018, Joyent, Inc.
28 */

30 #include <sys/types.h>
31 #include <sys/stream.h>
32 #include <sys/cmn_err.h>
33 #define _SUN_TPI_VERSION 2
34 #include <sys/tihdr.h>
35 #include <sys/ddi.h>
36 #include <sys/sunddi.h>
37 #include <sys/tsol/tndb.h>

39 #include <netinet/in.h>

41 #include <inet/common.h>
42 #include <inet/ip.h>
43 #include <inet/mib2.h>
44 #include <inet/snmpcom.h>
45 #include <inet/kstatcom.h>
46 #include <inet/ipclassifier.h>
47 #include "sctp_impl.h"
48 #include "sctp_addr.h"

50 static void sctp_clr_kstats2(sctp_kstat_t *);
51 static void sctp_add_kstats2(sctp_kstat_counter_t *, sctp_kstat_t *);
52 static int sctp_snmp_state(sctp_t *);
53 static void sctp_sum_mib(sctp_stack_t *, mib2_sctp_t *);
54 static void sctp_add_mib(mib2_sctp_t *, mib2_sctp_t *);

56 static int
57 sctp_kstat_update(kstat_t *kp, int rw)
58 {
59     sctp_named_kstat_t    *sctpkp;
60     sctp_t                *sctp, *sctp_prev;
61     zoneid_t              myzoneid;

```

```

62     netstackid_t         stackid = (netstackid_t)(uintptr_t)kp->ks_private;
63     netstack_t           *ns;
64     sctp_stack_t         *sctps;
65     mib2_sctp_t          sctp_mib;

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

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

73     ns = netstack_find_by_stackid(stackid);
74     if (ns == NULL)
75         return (-1);
76     sctps = ns->netstack_sctp;
77     if (sctps == NULL) {
78         netstack_rele(ns);
79         return (-1);
80     }

82     /*
83      * For all exclusive netstacks, the zone ID is always GLOBAL_ZONEID.
84      */
85     if (stackid != GLOBAL_NETSTACKID)
86         myzoneid = GLOBAL_ZONEID;
87     else
88         myzoneid = curproc->p_zone->zone_id;

90     bzero(&sctp_mib, sizeof (sctp_mib));

92     /*
93      * Get the number of current associations and gather their
94      * individual set of statistics.
95      */
96     sctp_prev = NULL;
97     mutex_enter(&sctps->sctps_g_lock);
98     sctp = list_head(&sctps->sctps_g_list);
99     while (sctp != NULL) {
100         mutex_enter(&sctp->sctp_reflock);
101         if (sctp->sctp_condemned) {
102             mutex_exit(&sctp->sctp_reflock);
103             sctp = list_next(&sctps->sctps_g_list, sctp);
104             continue;
105         }
106         sctp->sctp_refcnt++;
107         mutex_exit(&sctp->sctp_reflock);
108         mutex_exit(&sctps->sctps_g_lock);
109         if (sctp_prev != NULL)
110             SCTP_REFRELE(sctp_prev);
111         if (sctp->sctp_connp->conn_zoneid != myzoneid)
112             goto next_sctp;
113         if (sctp->sctp_state == SCTPS_ESTABLISHED ||
114             sctp->sctp_state == SCTPS_SHUTDOWN_PENDING ||
115             sctp->sctp_state == SCTPS_SHUTDOWN_RECEIVED) {
116             /*
117              * Just bump the local sctp_mib. The number of
118              * existing associations is not kept in kernel.
119              */
120             BUMP_MIB(&sctp_mib, sctpCurrEstab);
121         }

123         if (sctp->sctp_opkts) {
124             SCTPS_UPDATE_MIB(sctps, sctpOutSCTPPkts,
125                 sctp->sctp_opkts);
126             sctp->sctp_opkts = 0;

```

```

127     }
129     if (sctp->sctp_obchunks) {
130         SCTPS_UPDATE_MIB(sctps, sctpOutCtrlChunks,
131             sctp->sctp_obchunks);
132         UPDATE_LOCAL(sctp->sctp_cum_obchunks,
133             sctp->sctp_obchunks);
134         sctp->sctp_obchunks = 0;
135     }
137     if (sctp->sctp_odchunks) {
138         SCTPS_UPDATE_MIB(sctps, sctpOutOrderChunks,
139             sctp->sctp_odchunks);
140         UPDATE_LOCAL(sctp->sctp_cum_odchunks,
141             sctp->sctp_odchunks);
142         sctp->sctp_odchunks = 0;
143     }
145     if (sctp->sctp_oudchunks) {
146         SCTPS_UPDATE_MIB(sctps, sctpOutUnorderChunks,
147             sctp->sctp_oudchunks);
148         UPDATE_LOCAL(sctp->sctp_cum_oudchunks,
149             sctp->sctp_oudchunks);
150         sctp->sctp_oudchunks = 0;
151     }
153     if (sctp->sctp_rxtchunks) {
154         SCTPS_UPDATE_MIB(sctps, sctpRetransChunks,
155             sctp->sctp_rxtchunks);
156         UPDATE_LOCAL(sctp->sctp_cum_rxtchunks,
157             sctp->sctp_rxtchunks);
158         sctp->sctp_rxtchunks = 0;
159     }
161     if (sctp->sctp_ipkts) {
162         SCTPS_UPDATE_MIB(sctps, sctpInSCTPPkts,
163             sctp->sctp_ipkts);
164         sctp->sctp_ipkts = 0;
165     }
167     if (sctp->sctp_ibchunks) {
168         SCTPS_UPDATE_MIB(sctps, sctpInCtrlChunks,
169             sctp->sctp_ibchunks);
170         UPDATE_LOCAL(sctp->sctp_cum_ibchunks,
171             sctp->sctp_ibchunks);
172         sctp->sctp_ibchunks = 0;
173     }
175     if (sctp->sctp_idchunks) {
176         SCTPS_UPDATE_MIB(sctps, sctpInOrderChunks,
177             sctp->sctp_idchunks);
178         UPDATE_LOCAL(sctp->sctp_cum_idchunks,
179             sctp->sctp_idchunks);
180         sctp->sctp_idchunks = 0;
181     }
183     if (sctp->sctp_iudchunks) {
184         SCTPS_UPDATE_MIB(sctps, sctpInUnorderChunks,
185             sctp->sctp_iudchunks);
186         UPDATE_LOCAL(sctp->sctp_cum_iudchunks,
187             sctp->sctp_iudchunks);
188         sctp->sctp_iudchunks = 0;
189     }
191     if (sctp->sctp_fragdmsgs) {
192         SCTPS_UPDATE_MIB(sctps, sctpFragUsrMsgs,

```

```

193         sctp->sctp_fragdmsgs);
194         sctp->sctp_fragdmsgs = 0;
195     }
197     if (sctp->sctp_reassmsgs) {
198         SCTPS_UPDATE_MIB(sctps, sctpReasmUsrMsgs,
199             sctp->sctp_reassmsgs);
200         sctp->sctp_reassmsgs = 0;
201     }
203 next_sctp:
204     sctp_prev = sctp;
205     mutex_enter(&sctps->sctps_g_lock);
206     sctp = list_next(&sctps->sctps_g_list, sctp);
207 }
208 mutex_exit(&sctps->sctps_g_lock);
209 if (sctp_prev != NULL)
210     Sctp_Release(sctp_prev);
212     sctp_sum_mib(sctps, &sctp_mib);
214     /* Copy data from the Sctp MIB */
215     sctpkp = (sctp_named_kstat_t *)kp->ks_data;
217     /* These are from global ndd params. */
218     sctpkp->sctpRtoMin.value.ui32 = sctps->sctps_rto_ming;
219     sctpkp->sctpRtoMax.value.ui32 = sctps->sctps_rto_maxg;
220     sctpkp->sctpRtoInitial.value.ui32 = sctps->sctps_rto_initialg;
221     sctpkp->sctpValCookieLife.value.ui32 = sctps->sctps_cookie_life;
222     sctpkp->sctpMaxInitRetr.value.ui32 = sctps->sctps_max_init_retr;
224     /* Copy data from the local sctp_mib to the provided kstat. */
225     sctpkp->sctpCurrEstab.value.i32 = sctp_mib.sctpCurrEstab;
226     sctpkp->sctpActiveEstab.value.i32 = sctp_mib.sctpActiveEstab;
227     sctpkp->sctpPassiveEstab.value.i32 = sctp_mib.sctpPassiveEstab;
228     sctpkp->sctpAborted.value.i32 = sctp_mib.sctpAborted;
229     sctpkp->sctpShutdowns.value.i32 = sctp_mib.sctpShutdowns;
230     sctpkp->sctpOutOfBlue.value.i32 = sctp_mib.sctpOutOfBlue;
231     sctpkp->sctpChecksumError.value.i32 = sctp_mib.sctpChecksumError;
232     sctpkp->sctpOutCtrlChunks.value.i64 = sctp_mib.sctpOutCtrlChunks;
233     sctpkp->sctpOutOrderChunks.value.i64 = sctp_mib.sctpOutOrderChunks;
234     sctpkp->sctpOutUnorderChunks.value.i64 = sctp_mib.sctpOutUnorderChunks;
235     sctpkp->sctpRetransChunks.value.i64 = sctp_mib.sctpRetransChunks;
236     sctpkp->sctpOutAck.value.i32 = sctp_mib.sctpOutAck;
237     sctpkp->sctpOutAckDelayed.value.i32 = sctp_mib.sctpOutAckDelayed;
238     sctpkp->sctpOutWinUpdate.value.i32 = sctp_mib.sctpOutWinUpdate;
239     sctpkp->sctpOutFastRetrans.value.i32 = sctp_mib.sctpOutFastRetrans;
240     sctpkp->sctpOutWinProbe.value.i32 = sctp_mib.sctpOutWinProbe;
241     sctpkp->sctpInCtrlChunks.value.i64 = sctp_mib.sctpInCtrlChunks;
242     sctpkp->sctpInOrderChunks.value.i64 = sctp_mib.sctpInOrderChunks;
243     sctpkp->sctpInUnorderChunks.value.i64 = sctp_mib.sctpInUnorderChunks;
244     sctpkp->sctpInAck.value.i32 = sctp_mib.sctpInAck;
245     sctpkp->sctpInDupAck.value.i32 = sctp_mib.sctpInDupAck;
246     sctpkp->sctpInAckUnsent.value.i32 = sctp_mib.sctpInAckUnsent;
247     sctpkp->sctpFragUsrMsgs.value.i64 = sctp_mib.sctpFragUsrMsgs;
248     sctpkp->sctpReasmUsrMsgs.value.i64 = sctp_mib.sctpReasmUsrMsgs;
249     sctpkp->sctpOutSCTPPkts.value.i64 = sctp_mib.sctpOutSCTPPkts;
250     sctpkp->sctpInSCTPPkts.value.i64 = sctp_mib.sctpInSCTPPkts;
251     sctpkp->sctpInInvalidCookie.value.i32 = sctp_mib.sctpInInvalidCookie;
252     sctpkp->sctpTimRetrans.value.i32 = sctp_mib.sctpTimRetrans;
253     sctpkp->sctpTimRetransDrop.value.i32 = sctp_mib.sctpTimRetransDrop;
254     sctpkp->sctpTimHeartBeatProbe.value.i32 =
255         sctp_mib.sctpTimHeartBeatProbe;
256     sctpkp->sctpTimHeartBeatDrop.value.i32 = sctp_mib.sctpTimHeartBeatDrop;
257     sctpkp->sctpListenDrop.value.i32 = sctp_mib.sctpListenDrop;
258     sctpkp->sctpInClosed.value.i32 = sctp_mib.sctpInClosed;

```

`new/usr/src/uts/common/inet/sctp/sctp_snmp.c`

5

```
260         netstack_rele(ns);  
261         return (0);  
262     }  
_____unchanged_portion_omitted_____
```