```
**********************************************************
  108019 Mon Jul 29 18:04:06 2013
new/usr/src/lib/libipadm/common/ipadm_addr.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

/*
 * This file contains functions for address management such as creating
 * an address, deleting an address, enabling an address, disabling an
 * address, bringing an address down or up, setting/getting properties
 * on an address object and listing address information
 * for all addresses in active as well as persistent configuration.
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <inet/ip.h>
#include <string.h>
#include <strings.h>
#include <assert.h>
#include <sys/sockio.h>
#include <errno.h>
#include <unistd.h>
#include <stropts.h>
#include <zone.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <ctype.h>
#include <dhcpagent_util.h>
#include <dhcpagent_ipc.h>
#include <ipadm_ndpd.h>
#include <libdladm.h>
#include <libdllink.h>
#include <libdliptun.h>
#include <ifaddrs.h>
#include "libipadm_impl.h"

#define SIN6(a)         ((struct sockaddr_in6 *)a)
```



```
#define SIN(a)          ((struct sockaddr_in *)a)

static ipadm_status_t   i_ipadm_create_addr(ipadm_handle_t, ipadm_addrobj_t,
                            uint32_t);
static ipadm_status_t   i_ipadm_create_dhcp(ipadm_handle_t, ipadm_addrobj_t,
                            uint32_t);
static ipadm_status_t   i_ipadm_delete_dhcp(ipadm_handle_t, ipadm_addrobj_t,
                            boolean_t);
static ipadm_status_t   i_ipadm_get_db_addr(ipadm_handle_t, const char *,
                            const char *, nvlist_t **);
static ipadm_status_t   i_ipadm_op_dhcp(ipadm_addrobj_t, dhcp_ipc_type_t,
                            int *);
static ipadm_status_t   i_ipadm_validate_create_addr(ipadm_handle_t,
                            ipadm_addrobj_t, uint32_t);
static ipadm_status_t   i_ipadm_addr_persist_nvl(ipadm_handle_t, nvlist_t *,
                            uint32_t);
static ipadm_status_t   i_ipadm_get_default_prefixlen(struct sockaddr_storage *,
                            uint32_t *);
static ipadm_status_t   i_ipadm_get_static_addr_db(ipadm_handle_t,
                            ipadm_addrobj_t);
static boolean_t        i_ipadm_is_user_aobjname_valid(const char *);

/*
 * Callback functions to retrieve property values from the kernel. These
 * functions, when required, translate the values from the kernel to a format
 * suitable for printing. They also retrieve DEFAULT, PERM and POSSIBLE values
 * for a given property.
 */
static ipadm_pd_getf_t  i_ipadm_get_prefixlen, i_ipadm_get_addr_flag,
                        i_ipadm_get_zone, i_ipadm_get_broadcast;

/*
 * Callback functions to set property values. These functions translate the
 * values to a format suitable for kernel consumption, allocate the necessary
 * ioctl buffers and then invoke ioctl().
 */
static ipadm_pd_setf_t  i_ipadm_set_prefixlen, i_ipadm_set_addr_flag,
                        i_ipadm_set_zone;

/* address properties description table */
ipadm_prop_desc_t ipadm_addrprop_table[] = {
        { "broadcast", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "broadcast", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            NULL, NULL, i_ipadm_get_broadcast },

        { "deprecated", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "deprecated", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            i_ipadm_set_addr_flag, i_ipadm_get_onoff,
            i_ipadm_get_addr_flag },

        { "prefixlen", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "prefixlen", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            i_ipadm_set_prefixlen, i_ipadm_get_prefixlen,
            i_ipadm_get_prefixlen },

        { "private", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "private", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            i_ipadm_set_addr_flag, i_ipadm_get_onoff, i_ipadm_get_addr_flag },

        { "transmit", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "transmit", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            i_ipadm_set_addr_flag, i_ipadm_get_onoff, i_ipadm_get_addr_flag },

        { "zone", NULL, IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
        { "zone", IPADMPROP_CLASS_ADDR, MOD_PROTO_NONE, 0,
            i_ipadm_set_zone, NULL, i_ipadm_get_zone },
```

```
120         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
119         { NULL, 0, 0, 0, NULL, NULL, NULL }
121 };

123 static ipadm_prop_desc_t up_addrprop = { "up", NULL, IPADMPROP_CLASS_ADDR,
122 static ipadm_prop_desc_t up_addrprop = { "up", IPADMPROP_CLASS_ADDR,
124                                         MOD_PROTO_NONE, 0, NULL, NULL, NULL };

126 /*
127  * Helper function that initializes the 'ipadm_ifname', 'ipadm_aobjname', and
128  * 'ipadm_atype' fields of the given 'ipaddr'.
129  */
130 void
131 i_ipadm_init_addr(ipadm_addrobj_t ipaddr, const char *ifname,
132     const char *aobjname, ipadm_addr_type_t atype)
133 {
134         bzero(ipaddr, sizeof (struct ipadm_addrobj_s));
135         (void) strlcpy(ipaddr->ipadm_ifname, ifname,
136             sizeof (ipaddr->ipadm_ifname));
137         (void) strlcpy(ipaddr->ipadm_aobjname, aobjname,
138             sizeof (ipaddr->ipadm_aobjname));
139         ipaddr->ipadm_atype = atype;
140 }
______unchanged_portion_omitted_

1374 static ipadm_prop_desc_t *
1375 i_ipadm_get_addrprop_desc(const char *pname)
1376 {
1377         int i;

1379         for (i = 0; ipadm_addrprop_table[i].ipd_name != NULL; i++) {
1380                 if (strcmp(pname, ipadm_addrprop_table[i].ipd_name) == 0 ||
1381                     (ipadm_addrprop_table[i].ipd_old_name != NULL &&
1382                     strcmp(pname, ipadm_addrprop_table[i].ipd_old_name) == 0))
1379                 if (strcmp(pname, ipadm_addrprop_table[i].ipd_name) == 0)
1383                         return (&ipadm_addrprop_table[i]);
1384         }
1385         return (NULL);
1386 }
______unchanged_portion_omitted_
```

```
**********************************************************
   54869 Mon Jul 29 18:04:08 2013
new/usr/src/lib/libipadm/common/ipadm_prop.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

/*
 * This file contains routines that are used to modify/retrieve protocol or
 * interface property values. It also holds all the supported properties for
 * both IP interface and protocols in 'ipadm_prop_desc_t'. Following protocols
 * are supported: IP, IPv4, IPv6, TCP, SCTP, UDP and ICMP.
 *
 * This file also contains walkers, which walks through the property table and
 * calls the callback function, of the form 'ipadm_prop_wfunc_t' , for every
 * property in the table.
 */

#include <unistd.h>
#include <errno.h>
#include <ctype.h>
#include <fcntl.h>
#include <strings.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/sockio.h>
#include <assert.h>
#include <libdllink.h>
#include <zone.h>
#include "libipadm_impl.h"
#include <inet/tunables.h>

#define IPADM_NONESTR           "none"
#define DEF_METRIC_VAL          0       /* default metric value */

#define A_CNT(arr)      (sizeof (arr) / sizeof (arr[0]))

static ipadm_status_t   i_ipadm_validate_if(ipadm_handle_t, const char *,
                            uint_t, uint_t);
```



```
/*
 * Callback functions to retrieve property values from the kernel. These
 * functions, when required, translate the values from the kernel to a format
 * suitable for printing. For example: boolean values will be translated
 * to on/off. They also retrieve DEFAULT, PERM and POSSIBLE values for
 * a given property.
 */
static ipadm_pd_getf_t  i_ipadm_get_prop, i_ipadm_get_ifprop_flags,
                        i_ipadm_get_mtu, i_ipadm_get_metric,
                        i_ipadm_get_usesrc, i_ipadm_get_forwarding,
                        i_ipadm_get_ecnsack, i_ipadm_get_hostmodel;

/*
 * Callback function to set property values. These functions translate the
 * values to a format suitable for kernel consumption, allocates the necessary
 * ioctl buffers and then invokes ioctl().
 */
static ipadm_pd_setf_t  i_ipadm_set_prop, i_ipadm_set_mtu,
                        i_ipadm_set_ifprop_flags,
                        i_ipadm_set_metric, i_ipadm_set_usesrc,
                        i_ipadm_set_forwarding, i_ipadm_set_eprivport,
                        i_ipadm_set_ecnsack, i_ipadm_set_hostmodel;

/* array of protocols we support */
static int protocols[] = { MOD_PROTO_IP, MOD_PROTO_RAWIP,
                           MOD_PROTO_TCP, MOD_PROTO_UDP,
                           MOD_PROTO_SCTP };

/*
 * Supported IP protocol properties.
 */
static ipadm_prop_desc_t ipadm_ip_prop_table[] = {
        { "arp", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
        { "arp", IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_ifprop_flags, i_ipadm_get_onoff,
            i_ipadm_get_ifprop_flags },

        { "forwarding", NULL, IPADMPROP_CLASS_MODIF, MOD_PROTO_IPV4, 0,
        { "forwarding", IPADMPROP_CLASS_MODIF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_forwarding, i_ipadm_get_onoff,
            i_ipadm_get_forwarding },

        { "metric", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
        { "metric", IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_metric, NULL, i_ipadm_get_metric },

        { "mtu", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
        { "mtu", IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_mtu, i_ipadm_get_mtu, i_ipadm_get_mtu },

        { "exchange_routes", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
        { "exchange_routes", IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_ifprop_flags, i_ipadm_get_onoff,
            i_ipadm_get_ifprop_flags },

        { "usesrc", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
        { "usesrc", IPADMPROP_CLASS_IF, MOD_PROTO_IPV4, 0,
            i_ipadm_set_usesrc, NULL, i_ipadm_get_usesrc },

        { "ttl", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV4, 0,
        { "ttl", IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV4, 0,
            i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

        { "forwarding", NULL, IPADMPROP_CLASS_MODIF, MOD_PROTO_IPV6, 0,
        { "forwarding", IPADMPROP_CLASS_MODIF, MOD_PROTO_IPV6, 0,
```

```
117             i_ipadm_set_forwarding, i_ipadm_get_onoff,
118             i_ipadm_get_forwarding },

120         { "hoplimit", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV6, 0,
119         { "hoplimit", IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV6, 0,
121             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

123         { "metric", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
122         { "metric", IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
124             i_ipadm_set_metric, NULL, i_ipadm_get_metric },

126         { "mtu", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
125         { "mtu", IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
127             i_ipadm_set_mtu, i_ipadm_get_mtu, i_ipadm_get_mtu },

129         { "nud", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
128         { "nud", IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
130             i_ipadm_set_ifprop_flags, i_ipadm_get_onoff,
131             i_ipadm_get_ifprop_flags },

133         { "exchange_routes", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
132         { "exchange_routes", IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
134             i_ipadm_set_ifprop_flags, i_ipadm_get_onoff,
135             i_ipadm_get_ifprop_flags },

137         { "usesrc", NULL, IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
136         { "usesrc", IPADMPROP_CLASS_IF, MOD_PROTO_IPV6, 0,
138             i_ipadm_set_usesrc, NULL, i_ipadm_get_usesrc },

140         { "hostmodel", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV6, 0,
139         { "hostmodel", IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV6, 0,
141             i_ipadm_set_hostmodel, i_ipadm_get_hostmodel,
142             i_ipadm_get_hostmodel },

144         { "hostmodel", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV4, 0,
143         { "hostmodel", IPADMPROP_CLASS_MODULE, MOD_PROTO_IPV4, 0,
145             i_ipadm_set_hostmodel, i_ipadm_get_hostmodel,
146             i_ipadm_get_hostmodel },

148         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
147         { NULL, 0, 0, 0, NULL, NULL, NULL }
149 };

151 /* possible values for TCP properties 'ecn' and 'sack' */
152 static const char *ecn_sack_vals[] = {"never", "passive", "active", NULL};

154 /* Supported TCP protocol properties */
155 static ipadm_prop_desc_t ipadm_tcp_prop_table[] = {
156         { "ecn", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
155         { "ecn", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
157             i_ipadm_set_ecnsack, i_ipadm_get_ecnsack, i_ipadm_get_ecnsack },

159         { "extra_priv_ports", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP,
158         { "extra_priv_ports", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP,
160             IPADMPROP_MULVAL, i_ipadm_set_eprivport, i_ipadm_get_prop,
161             i_ipadm_get_prop },

163         { "largest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
162         { "largest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
164             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

166         { "max_buf", "_max_buf", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
165         { "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
167             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

169         { "recv_buf", "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
```



```
170             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

172         { "sack", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
168         { "sack", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
173             i_ipadm_set_ecnsack, i_ipadm_get_ecnsack, i_ipadm_get_ecnsack },

175         { "send_buf", "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
171         { "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
176             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

178         { "smallest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
174         { "smallest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
179             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

181         { "smallest_nonpriv_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP,
182             0, i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },
177         { "smallest_nonpriv_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_TCP, 0,
178             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

184         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
180         { NULL, 0, 0, 0, NULL, NULL, NULL }
185 };

187 /* Supported UDP protocol properties */
188 static ipadm_prop_desc_t ipadm_udp_prop_table[] = {
189         { "extra_priv_ports", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP,
185         { "extra_priv_ports", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP,
190             IPADMPROP_MULVAL, i_ipadm_set_eprivport, i_ipadm_get_prop,
191             i_ipadm_get_prop },

193         { "largest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
189         { "largest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
194             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

196         { "max_buf", "_max_buf", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
192         { "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
197             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

199         { "recv_buf", "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
195         { "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
200             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

202         { "send_buf", "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
198         { "smallest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
203             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

205         { "smallest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
201         { "smallest_nonpriv_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP, 0,
206             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

208         { "smallest_nonpriv_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_UDP,
209             0, i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

211         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
204         { NULL, 0, 0, 0, NULL, NULL, NULL }
212 };

214 /* Supported SCTP protocol properties */
215 static ipadm_prop_desc_t ipadm_sctp_prop_table[] = {
216         { "extra_priv_ports", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP,
209         { "extra_priv_ports", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP,
217             IPADMPROP_MULVAL, i_ipadm_set_eprivport, i_ipadm_get_prop,
218             i_ipadm_get_prop },

220         { "largest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
213         { "largest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
```

```
221             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

223         { "max_buf", "_max_buf", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
216         { "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
224             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

226         { "recv_buf", "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
219         { "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
227             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

229         { "send_buf", "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
222         { "smallest_anon_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
230             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

232         { "smallest_anon_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
225         { "smallest_nonpriv_port", IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP, 0,
233             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

235         { "smallest_nonpriv_port", NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_SCTP,
236             0, i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

238         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
228         { NULL, 0, 0, 0, NULL, NULL, NULL }
239 };

241 /* Supported ICMP protocol properties */
242 static ipadm_prop_desc_t ipadm_icmp_prop_table[] = {
243         { "max_buf", "_max_buf", IPADMPROP_CLASS_MODULE, MOD_PROTO_RAWIP, 0,
233         { "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_RAWIP, 0,
244             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

246         { "recv_buf", "recv_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_RAWIP, 0,
236         { "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_RAWIP, 0,
247             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

249         { "send_buf", "send_maxbuf", IPADMPROP_CLASS_MODULE, MOD_PROTO_RAWIP, 0,
250             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop },

252         { NULL, NULL, 0, 0, 0, NULL, NULL, NULL }
239         { NULL, 0, 0, 0, NULL, NULL, NULL }
253 };

255 /*
256  * A dummy private property structure, used while handling private
257  * protocol properties (properties not yet supported by libipadm).
258  */
259 static ipadm_prop_desc_t ipadm_privprop =
260         { NULL, NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_NONE, 0,
246 static ipadm_prop_desc_t        ipadm_privprop =\
247         { NULL, IPADMPROP_CLASS_MODULE, MOD_PROTO_NONE, 0,
261             i_ipadm_set_prop, i_ipadm_get_prop, i_ipadm_get_prop };

263 /*
264  * Returns the property description table, for the given protocol
265  */
266 static ipadm_prop_desc_t *
267 i_ipadm_get_propdesc_table(uint_t proto)
268 {
269         switch (proto) {
270         case MOD_PROTO_IP:
271         case MOD_PROTO_IPV4:
272         case MOD_PROTO_IPV6:
273                 return (ipadm_ip_prop_table);
274         case MOD_PROTO_RAWIP:
275                 return (ipadm_icmp_prop_table);
276         case MOD_PROTO_TCP:
```



```
277                 return (ipadm_tcp_prop_table);
278         case MOD_PROTO_UDP:
279                 return (ipadm_udp_prop_table);
280         case MOD_PROTO_SCTP:
281                 return (ipadm_sctp_prop_table);
282         }

284         return (NULL);
285 }

287 static ipadm_prop_desc_t *
288 i_ipadm_get_prop_desc(const char *pname, uint_t proto, int *errp)
289 {
290         int             err = 0;
291         boolean_t       matched_name = B_FALSE;
292         ipadm_prop_desc_t *ipdp = NULL, *ipdtbl;

294         if ((ipdtbl = i_ipadm_get_propdesc_table(proto)) == NULL) {
295                 err = EINVAL;
296                 goto ret;
297         }

299         for (ipdp = ipdtbl; ipdp->ipd_name != NULL; ipdp++) {
300                 if (strcmp(pname, ipdp->ipd_name) == 0 ||
301                     (ipdp->ipd_old_name != NULL &&
302                     strcmp(pname, ipdp->ipd_old_name) == 0)) {
286                 if (strcmp(pname, ipdp->ipd_name) == 0) {
303                         matched_name = B_TRUE;
304                         if (ipdp->ipd_proto == proto)
305                                 break;
306                 }
307         }

309         if (ipdp->ipd_name == NULL) {
310                 err = ENOENT;
311                 /* if we matched name, but failed protocol check */
312                 if (matched_name)
313                         err = EPROTO;
314                 ipdp = NULL;
315         }
316 ret:
317         if (errp != NULL)
318                 *errp = err;
319         return (ipdp);
320 }
______unchanged_portion_omitted_
```

```
**********************************************************
    8986 Mon Jul 29 18:04:10 2013
new/usr/src/lib/libipadm/common/libipadm_impl.h
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _LIBIPADM_IMPL_H
#define _LIBIPADM_IMPL_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/socket.h>
#include <net/if.h>
#include <libipadm.h>
#include <libdladm.h>
#include <ipadm_ipmgmt.h>
#include <inet/tunables.h>
#include <netinet/in.h>
#include <pthread.h>
#include <libinetutil.h>
#include <libsocket_priv.h>

#define IPADM_STRSIZE           256
#define IPADM_ONSTR             "on"
#define IPADM_OFFSTR            "off"
#define ARP_MOD_NAME            "arp"
#define IPADM_LOGICAL_SEP       ':'
#define IPV6_MIN_MTU            1280    /* rfc2460 */

/* mask for flags accepted by libipadm functions */
#define IPADM_COMMON_OPT_MASK   (IPADM_OPT_ACTIVE | IPADM_OPT_PERSIST)

/* Opaque library handle */
struct ipadm_handle {
        int             iph_sock;       /* socket to interface */
        int             iph_sock6;      /* socket to interface */
        int             iph_door_fd;    /* door descriptor to ipmgmtd */
```



```
        int             iph_rtsock;     /* routing socket */
        dladm_handle_t  iph_dlh;        /* handle to libdladm library */
        uint32_t        iph_flags;      /* internal flags */
        pthread_mutex_t iph_lock;       /* lock to set door_fd */
        zoneid_t        iph_zoneid;     /* zoneid where handle was opened */
};
______unchanged_portion_omitted_

#define ipadm_static_addr       ipadm_addr_u.ipadm_static_addr_s.ipadm_addr
#define ipadm_static_aname      ipadm_addr_u.ipadm_static_addr_s.ipadm_ahname
#define ipadm_static_prefixlen  ipadm_addr_u.ipadm_static_addr_s.ipadm_prefixlen
#define ipadm_static_dst_addr   ipadm_addr_u.ipadm_static_addr_s.ipadm_dstaddr
#define ipadm_static_dname      ipadm_addr_u.ipadm_static_addr_s.ipadm_dhname
#define ipadm_intfid            ipadm_addr_u.ipadm_ipv6_intfid_s.ipadm_intfid
#define ipadm_intfidlen         ipadm_addr_u.ipadm_ipv6_intfid_s.ipadm_intfidlen
#define ipadm_stateless         ipadm_addr_u.ipadm_ipv6_intfid_s.ipadm_stateless
#define ipadm_stateful          ipadm_addr_u.ipadm_ipv6_intfid_s.ipadm_stateful
#define ipadm_primary           ipadm_addr_u.ipadm_dhcp_s.ipadm_primary
#define ipadm_wait              ipadm_addr_u.ipadm_dhcp_s.ipadm_wait

/*
 * Data structures and callback functions related to property management
 */
struct ipadm_prop_desc;
typedef struct ipadm_prop_desc ipadm_prop_desc_t;

/* property set() callback */
typedef ipadm_status_t  ipadm_pd_setf_t(ipadm_handle_t, const void *,
    ipadm_prop_desc_t *, const void *, uint_t, uint_t);

/* property get() callback */
typedef ipadm_status_t  ipadm_pd_getf_t(ipadm_handle_t, const void *,
    ipadm_prop_desc_t *, char *, uint_t *, uint_t, uint_t);

struct ipadm_prop_desc {
        char            *ipd_name;      /* property name */
        char            *ipd_old_name;  /* for backward compatibility */
        uint_t          ipd_class;      /* prop. class - global/perif/both */
        uint_t          ipd_proto;      /* protocol to which property belongs */
        uint_t          ipd_flags;      /* see below */
        ipadm_pd_setf_t *ipd_set;       /* set callback function */
        ipadm_pd_getf_t *ipd_get_range; /* get range callback function */
        ipadm_pd_getf_t *ipd_get;       /* get value callback function */
};
______unchanged_portion_omitted_
```

```
**********************************************************
   22256 Mon Jul 29 18:04:12 2013
new/usr/src/man/man1m/ipadm.1m
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@delphix.com>
**********************************************************
```

```
'\" te
.\" Copyright (c) 2012, Joyent, Inc. All Rights Reserved
.\" Copyright (c) 2013 by Delphix. All rights reserved.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH IPADM 1M "May 14, 2012"
.SH NAME
ipadm \- configure IP network interfaces and protocol properties.
.SH SYNOPSIS
.LP
.nf
\fBipadm\fR create-if [\fB-t\fR] \fIinterface\fR
.fi

.LP
.nf
\fBipadm\fR disable-if [\fB-t\fR] \fIinterface\fR
.fi

.LP
.nf
\fBipadm\fR enable-if [\fB-t\fR] \fIinterface\fR
.fi

.LP
.nf
\fBipadm\fR delete-if \fIinterface\fR
.fi

.LP
.nf
\fBipadm\fR show-if [[\fB-p\fR] \fB-o\fR \fIfield\fR[,...]] [\fIinterface\fR]
.fi

.LP
.nf
\fBipadm\fR set-ifprop [\fB-t\fR] \fB-p\fR \fIprop\fR=<\fIvalue\fR[,...]> \fB-m\
.fi

.LP
.nf
\fBipadm\fR reset-ifprop [\fB-t\fR] \fB-p\fR \fIprop\fR \fB-m\fR \fIprotocol\fR
.fi

.LP
.nf
\fBipadm\fR show-ifprop [[\fB-c\fR]\fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIprop
  [\fIinterface\fR]
.fi

.LP
.nf
\fBipadm\fR create-addr [\fB-t\fR] \fB-T\fR static [\fB-d\fR]
   \fB-a\fR {local|remote}=\fIaddr\fR[/\fIprefixlen\fR],... \fIaddrobj\fR
.fi

.LP
```



```
.nf
\fBipadm\fR create-addr [\fB-t\fR] \fB-T\fR dhcp [\fB-w\fR \fIseconds\fR | forev
.fi

.LP
.nf
\fBipadm\fR create-addr [\fB-t\fR] \fB-T\fR addrconf [\fB-i\fR \fIinterface-id\f
   [\fB-p\fR {stateful|stateless}={yes|no},..] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR down-addr [\fB-t\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR up-addr [\fB-t\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR disable-addr [\fB-t\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR enable-addr [\fB-t\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR refresh-addr [\fB-i\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR delete-addr [\fB-r\fR] \fIaddrobj\fR
.fi

.LP
.nf
\fBipadm\fR show-addr [[\fB-p\fR] \fB-o\fR \fIfield\fR[,...]] [\fIaddrobj\fR]
.fi

.LP
.nf
\fBipadm\fR set-addrprop [\fB-t\fR] \fB-p\fR \fIprop\fR=<\fIvalue\fR[,...]> \fIa
.fi

.LP
.nf
\fBipadm\fR reset-addrprop [\fB-t\fR] \fB-p\fR \fIprop\fR=<\fIvalue\fR[,...]> \f
.fi

.LP
.nf
\fBipadm\fR show-addrprop [[\fB-c\fR] \fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIp
.fi

.LP
.nf
\fBipadm\fR set-prop [\fB-t\fR] \fB-p\fR \fIprop\fR[+|-]=<\fIvalue\fR[,...]> \fI
.fi

.LP
```

```
.nf
\fBipadm\fR reset-prop [\fB-t\fR] \fB-p\fR \fIprop\fR \fIprotocol\fR
.fi

.LP
.nf
\fBipadm\fR show-prop [[\fB-c\fR] \fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIprop\
.fi

.SH DESCRIPTION
.sp
.LP

The \fBipadm\fR command is a stable replacement for the \fBifconfig\fR(1M) and
\fBndd\fR(1M) commands. It is used to create IP interfaces and to confgure IP
addresses on those interfaces. It is also used to get, set or reset properties
on interfaces, addresses and protocols.
.LP
For subcommands that take an \fIaddrobj\fR, the \fIaddrobj\fR specifies a
unique address on the system. It is made up of two parts, delimited by a '/'.
The first part is the name of the interface and the second part is a string up
to 32 characters long.  For example, "lo0/v4" is a loopback interface
addrobj name.
.LP
For subcommands that take a \fIprotocol\fR, this can be one of
the following values: ip, ipv4, ipv6, icmp, tcp, sctp or udp.

.SH SUBCOMMANDS
.sp
.LP
The following subcommands are supported:
.sp
.ne 2
.na
\fB\fBcreate-if\fR [\fB-t\fR] \fIinterface\fR\fR
.ad
.sp .6
.RS 4n
The \fBcreate-if\fR subcommand is used to create an IP interface that will
handle both IPv4 and IPv6 packets. The interface will be enabled as part of
the creation process.  The IPv4 interface will have the address 0.0.0.0.
The IPv6 interface will have the adress ::.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the creation is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fBdisable-if\fR [\fB-t\fR] \fIinterface\fR\fR
.ad
.sp .6
.RS 4n
The \fBdisable-if\fR subcommand is used to disable an IP interface.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the disable is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
```



```
.na
\fB\fBenable-if\fR [\fB-t\fR] \fIinterface\fR\fR
.ad
.sp .6
.RS 4n
The \fBenable-if\fR subcommand is used to enable an IP interface.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the enable is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fbdelete-if\fR \fIinterface\fR\fR
.ad
.sp .6
.RS 4n
The \fBdelete-if\fR subcommand is used to permanently delete an IP interface.
.sp

.RE

.sp
.ne 2
.na
\fB\fBshow-if\fR [[\fB-p\fR] \fB-o\fR \fIfield\fR[,...]] [\fIinterface\fR]\fR
.ad
.sp .6
.RS 4n
The \fBshow-if\fR subcommand is used to show the current IP interface
configuration.
.sp
The \fB-p\fR option (also \fB--parsable\fR) prints
the output in a parsable format.
.sp
The \fB-o\fR option (also \fB--output\fR) is used
to select which fields will be shown.  The field value can be one of the
following names:
.sp
.ne 2
.na
.RS 4n
\fBALL\fR
.ad
.RS 4n
Display all fields
.RE

.sp
.ne 2
.na
\fBIFNAME\fR
.ad
.RS 4n
The name of the interface
.RE

.sp
.ne 2
.na
\fBSTATE\fR
.ad
.RS 4n
```

```
The state can be one of the following values:
.sp
.ne 2
.na
.RS 4n
ok - resources for the interface have been allocated
.sp
offline - the interface is offline
.sp
failed - the interface’s datalink is down
.sp
down - the interface is down
.sp
disabled - the interface is disabled
.RE
.RE

.sp
.ne 2
.na
\fBCURRENT\fR
.ad
.RS 4n
A set of single character flags indicating the following:
.sp
.ne 2
.na
.RS 4n
b - broadcast (mutually exclusive with ’p’)
.br
m - multicast
.br
p - point-to-point (mutually exclusive with ’b’)
.br
v - virtual interface
.br
I - IPMP
.br
s - IPMP standby
.br
i - IPMP inactive
.br
V - VRRP
.br
a - VRRP accept mode
.br
4 - IPv4
.br
6 - IPv6
.RE
.RE

.sp
.ne 2
.na
\fBPERSISTENT\fR
.ad
.RS 4n
A set of single character flags showing what configuration will be used the
next time the interface is enabled:
.sp
.ne 2
.na
.RS 4n
s - IPMP standby
.br
```



```
4 - IPv4
.br
6 - IPv6
.RE
.RE
.RE

.RE

.sp
.ne 2
.na
\fB\fBset-ifprop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR=<\fIvalue\fR[,...]> \fB-m\fR
.ad
.sp .6
.RS 4n
The \fBset-ifprop\fR subcommand is used to set a property’s value(s) on the IP
interface.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the setting is temporary and will not be persistent across reboots.
.sp
The \fB-p\fR option (also \fB--prop\fR) specifies the property name and
value(s).  The property name can be one of the following:
.sp
.ne 2
.na

.RS 4n

\fBarp\fR
.ad
.RS 4n
Enables ("on") or disables ("off") ARP.
.RE

.sp
.ne 2
.na
\fBexchange_routes\fR
.ad
.RS 4n
Enables ("on") or disables ("off") the exchange of routing data.
.RE

.sp
.ne 2
.na
\fBforwarding\fR
.ad
.RS 4n
Enables ("on") or disables ("off") IP forwarding.
.RE

.sp
.ne 2
.na
\fBmetric\fR
.ad
.RS 4n
Set the routing metric to the numeric value. The value is treated as extra
hops to the destination.
.RE

.sp
.ne 2
```

```
.na
\fBmtu\fR
.ad
.RS 4n
Set the maximum transmission unit to the numeric value.
.RE

.sp
.ne 2
.na
\fBnud\fR
.ad
.RS 4n
Enables ("on") or disables ("off") neighbor unreachability detection.
.RE

.sp
.ne 2
.na
\fBusesrc\fR
.ad
.RS 4n
Indicates which interface to use for source address selection. A value
"none" may also be used.
.RE
.RE

.sp
The \fB-m\fR option (also \fB--module\fR) specifies which protocol
the setting applies to.
.sp

.RE
.RE

.sp
.ne 2
.na
\fB\fBreset-ifprop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR \fB-m\fR \fIprotocol\fR \fI
.ad
.sp .6
.RS 4n
The \fBreset-ifprop\fR subcommand is used to reset an IP interface’s property
value to the default.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the disable is temporary and will not be persistent across reboots.
.sp
The \fB-p\fR option (also \fB--prop\fR) specifies the property name.
See the \fBset-ifprop\fR subcommand for the list of property names.
.sp
The \fB-m\fR option (also \fB--module\fR) specifies which protocol
the setting applies to.
.sp

.RE

.sp
.ne 2
.na
\fB\fBshow-ifprop\fR [[\fB-c\fR]\fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIprop\fR
  [\fIinterface\fR]\fR
.ad
.sp .6
.RS 4n
The \fBshow-ifprop\fR subcommand is used to display the property values
```



```
for one or all of the IP interfaces.
.sp
The \fB-c\fR option (also \fB--parsable\fR) prints
the output in a parsable format.
.sp
The \fB-o\fR option (also \fB--output\fR) is used
to select which fields will be shown.  The field value can be one of the
following names:
.sp
.ne 2
.na
.RS 4n
\fBALL\fR
.ad
.RS 4n
Display all fields
.RE

.sp
.ne 2
.na
\fBIFNAME\fR
.ad
.RS 4n
The name of the interface
.RE

.sp
.ne 2
.na
\fBPROPERTY\fR
.ad
.RS 4n
The name of the property
.RE

.sp
.ne 2
.na
\fBPROTO\fR
.ad
.RS 4n
The name of the protocol
.RE

.sp
.ne 2
.na
\fBPERM\fR
.ad
.RS 4n
If the property is readable ("r") and/or writable ("w").
.RE

.sp
.ne 2
.na
\fBCURRENT\fR
.ad
.RS 4n
The value of the property
.RE

.sp
.ne 2
.na
```

```
\fBPERSISTENT\fR
.ad
.RS 4n
The persistent value of the property
.RE

.sp
.ne 2
.na
\fBDEFAULT\fR
.ad
.RS 4n
The default value of the property
.RE

.sp
.ne 2
.na
\fBPOSSIBLE\fR
.ad
.RS 4n
The possible values for the property
.RE
.RE

.sp
The \fB-p\fR option (also \fB--prop\fR) is used
to specify which properties to display.  See the \fBset-ifprop\fR
subcommand for the list of property names.
.sp
The \fB-m\fR option (also \fB--module\fR) specifies which protocol
to display.
.sp

.RE

.sp
.ne 2
.na
\fB\fBcreate-addr\fR [\fB-t\fR] \fB-T\fR static [\fB-d\fR] \\
    \fB-a\fR {local|remote}=\fIaddr\fR[/\fIprefixlen\fR],... \fIaddrobj\fR\fR
.br
\fB\fBcreate-addr\fR [\fB-t\fR] \fB-T\fR dhcp [\fB-w\fR \fIseconds\fR | forever
.br
\fB\fBcreate-addr\fR [\fB-t\fR] \fB-T\fR addrconf [\fB-i\fR \fIinterface-id\fR]
    [\fB-p\fR {stateful|stateless}={yes|no},..] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBcreate-addr\fR subcommand is used to set an address on an IP interface.
The address will be enabled but can disabled using the \fBdisable-addr\fR
subcommand. This subcommand has three different forms, depending on the
value of the \fB-T\fR option.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the address is temporary and will not be persistent across reboots.
.sp
The \fB-T\fR static option creates a static addrobj.  This takes the following
options:
.RS 4n

The \fB-d\fR option (also \fB--down\fR) means the address is down.
.sp
The \fB-a\fR option (also \fB--address\fR) specifies the address.
The "local" or "remote" prefix can be used for a point-to-point interface.
In this case, both addresses must be given.
```



```
Otherwise, the equal sign ("=") should be omitted and the address should be
provided by itself and with no second address.
.sp

.RE

The \fB-T\fR dhcp option causes the address to be obtained via DHCP.
This takes the following options:
.RS 4n

The \fB-w\fR option (also \fB--wait\fR) gives the time, in seconds,
that the command should wait to obtain an address.
.sp

.RE

The \fB-T\fR addrconf option creates an auto-configured address.
This takes the following options:
.RS 4n

The \fB-i\fR option (also \fB--interface-id\fR) gives the interface ID to
be used.
.sp
The \fB-p\fR option (also \fB--prop\fR) indicates which method of
auto-configuration should be used.
.sp

.RE
.RE

.sp
.ne 2
.na
\fB\fBdown-addr\fR [\fB-t\fR] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBdown-addr\fR subcommand is used to down the address. This will
stop packets from being sent or received.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the down is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fBup-addr\fR [\fB-t\fR] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBup-addr\fR subcommand is used to up the address. This will
enable packets to be sent and received.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the up is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fBdisable-addr\fR [\fB-t\fR] \fIaddrobj\fR\fR
```

```
.ad
.sp .6
.RS 4n
The \fBdisable-addr\fR subcommand is used to disable the address.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the disable is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fBenable-addr\fR [\fB-t\fR] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBenable-addr\fR subcommand is used to enable the address.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the enable is temporary and will not be persistent across reboots.
.sp

.RE

.sp
.ne 2
.na
\fB\fBrefresh-addr\fR [\fB-i\fR] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBrefresh-addr\fR subcommand is used to extend the lease for DHCP
addresses. It also restarts duplicate address detection for Static addresses.
.sp
The \fB-i\fR option (also \fB--inform\fR) means
that the network configuration will be obtained from DHCP without taking
a lease on the address.
.sp

.RE

.sp
.ne 2
.na
\fB\fBdelete-addr\fR [\fB-r\fR] \fIaddrobj\fR\fR
.ad
.sp .6
.RS 4n
The \fBdelete-addr\fR subcommand deletes the given address.
.sp
The \fB-r\fR option (also \fB--release\fR) is used for DHCP-assigned
addresses to indicate that the address should be released.
.sp

.RE

.sp
.ne 2
.na
\fB\fBshow-addr\fR [[\fB-p\fR] \fB-o\fR \fIfield\fR[,...]] [\fIaddrobj\fR]\fR
.ad
.sp .6
.RS 4n
The \fBshow-addr\fR subcommand is used to show the current address properties.
```



```
.sp
The \fB-p\fR option (also \fB--parsable\fR) prints
the output in a parsable format.
.sp
The \fB-o\fR option (also \fB--output\fR) is used
to select which fields will be shown.  The field value can be one of the
following names:
.sp
.ne 2
.na
.RS 4n
\fBALL\fR
.ad
.RS 4n
Display all fields
.RE

.sp
.ne 2
.na
\fBADDROBJ\fR
.ad
.RS 4n
The name of the address
.RE

.sp
.ne 2
.na
\fBTYPE\fR
.ad
.RS 4n
The type of the address.  It can be "static", "dhcp" or "addrconf".
.RE

.sp
.ne 2
.na
\fBSTATE\fR
.ad
.RS 4n
The state of the address.  It can be one of the following values:
.sp
.ne 2
.na
.RS 4n
disabled s see the \fBdisable-addr\fR subcommand
.sp
down - see the \fBdown-addr\fR subcommand
.sp
duplicate - the address is a duplicate
.sp
inaccessible - the interface for this address has failed
.sp
ok - the address is up
.sp
tentative - duplicate address detection in progress
.RE
.RE

.sp
.ne 2
.na
\fBCURRENT\fR
.ad
.RS 4n
```

```
785 A set of single character flags indicating the following:
786 .sp
787 .ne 2
788 .na
789 .RS 4n
790 U - up
791 .br
792 u - unnumbered (matches another local address)
793 .br
794 p - private, not advertised to routing
795 .br
796 t - temporary IPv6 address
797 .br
798 d - deprecated (not used for outgoing packets)
799 .RE
800 .RE

802 .sp
803 .ne 2
804 .na
805 \fBPERSISTENT\fR
806 .ad
807 .RS 4n
808 A set of single character flags showing the configuration which will be used
809 when the address is enabled.
810 .sp
811 .ne 2
812 .na
813 .RS 4n
814 U - up
815 .br
816 p - private, not advertised to routing
817 .br
818 d - deprecated (not used for outgoing packets)
819 .RE
820 .RE

822 .sp
823 .ne 2
824 .na
825 \fBADDR\fR
826 .ad
827 .RS 4n
828 The address
829 .RE
830 .RE

832 .RE

834 .sp
835 .ne 2
836 .na
837 \fB\fBset-addrprop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR=<\fIvalue\fR[,...]> \fIaddr
838 .ad
839 .sp .6
840 .RS 4n
841 The \fBset-addrprop\fR subcommand is used to set a property's value(s) on the
842 addrobj.
843 .sp
844 The \fB-t\fR option (also \fB--temporary\fR) means
845 that the setting is temporary and will not be persistent across reboots.
846 .sp
847 The \fB-p\fR option (also \fB--prop\fR) specifies the property name and
848 value(s).  The property name can be one of the following:
849 .sp
850 .ne 2
```



```
851 .na

853 .RS 4n

855 \fBbroadcast\fR
856 .ad
857 .RS 4n
858 The broadcast address (read-only)
859 .RE

861 .sp
862 .ne 2
863 .na
864 \fBdeprecated\fR
865 .ad
866 .RS 4n
867 The address should not be used to send packets but can still receive packets.
868 Can be "on" or "off".
869 .RE

871 .sp
872 .ne 2
873 .na
874 \fBprefixlen\fR
875 .ad
876 .RS 4n
877 The number of bits in the IPv4 netmask or IPv6 prefix.
878 .RE

880 .sp
881 .ne 2
882 .na
883 \fBprivate\fR
884 .ad
885 .RS 4n
886 The address is not advertised to routing.
887 Can be "on" or "off".
888 .RE

890 .sp
891 .ne 2
892 .na
893 \fBtransmit\fR
894 .ad
895 .RS 4n
896 Packets can be transmitted.
897 Can be "on" or "off".
898 .RE

900 .sp
901 .ne 2
902 .na
903 \fBzone\fR
904 .ad
905 .RS 4n
906 The zone the addrobj is in.
907 .RE

909 .RE
910 .RE

912 .sp
913 .ne 2
914 .na
915 \fB\fBreset-addrprop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR \fIaddrobj\fR\fR
916 .ad
```

```
 917 .sp .6
 918 .RS 4n
 919 The \fBreset-addrprop\fR subcommand is used to reset an addrobj’s property
 920 value to the default.
 921 .sp
 922 The \fB-t\fR option (also \fB--temporary\fR) means
 923 that the disable is temporary and will not be persistent across reboots.
 924 .sp
 925 The \fB-p\fR option (also \fB--prop\fR) specifies the property name.
 926 See the \fBset-addrprop\fR subcommand for the list of property names.
 927 .sp

 929 .RE

 931 .sp
 932 .ne 2
 933 .na
 934 \fB\fBshow-addrprop\fR [[\fB-c\fR]\fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIprop\
 935 .ad
 936 .sp .6
 937 .RS 4n
 938 The \fBshow-addrprop\fR subcommand is used to display the property values
 939 for one or all of the addrobjs.
 940 .sp
 941 The \fB-c\fR option (also \fB--parsable\fR) prints
 942 the output in a parsable format.
 943 .sp
 944 The \fB-o\fR option (also \fB--output\fR) is used
 945 to select which fields will be shown.  The field value can be one of the
 946 following names:
 947 .sp
 948 .ne 2
 949 .na
 950 .RS 4n
 951 \fBALL\fR
 952 .ad
 953 .RS 4n
 954 Display all fields
 955 .RE

 957 .sp
 958 .ne 2
 959 .na
 960 \fBADDROBJ\fR
 961 .ad
 962 .RS 4n
 963 The name of the addrobj
 964 .RE

 966 .sp
 967 .ne 2
 968 .na
 969 \fBPROPERTY\fR
 970 .ad
 971 .RS 4n
 972 The name of the property
 973 .RE

 975 .sp
 976 .ne 2
 977 .na
 978 \fBPERM\fR
 979 .ad
 980 .RS 4n
 981 If the property is readable ("r") and/or writable ("w").
 982 .RE
```



```
 984 .sp
 985 .ne 2
 986 .na
 987 \fBCURRENT\fR
 988 .ad
 989 .RS 4n
 990 The value of the property
 991 .RE

 993 .sp
 994 .ne 2
 995 .na
 996 \fBPERSISTENT\fR
 997 .ad
 998 .RS 4n
 999 The persistent value of the property
1000 .RE

1002 .sp
1003 .ne 2
1004 .na
1005 \fBDEFAULT\fR
1006 .ad
1007 .RS 4n
1008 The default value of the property
1009 .RE

1011 .sp
1012 .ne 2
1013 .na
1014 \fBPOSSIBLE\fR
1015 .ad
1016 .RS 4n
1017 The possible values for the property
1018 .RE
1019 .RE

1021 .sp
1022 The \fB-p\fR option (also \fB--prop\fR) is used
1023 to specify which properties to display.  See the \fBset-addrprop\fR
1024 subcommand for the list of property names.
1025 .sp

1027 .RE

1029 .sp
1030 .ne 2
1031 .na
1032 \fB\fBset-prop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR[+|-]=<\fIvalue\fR[,...]> \fIpro
1033 .ad
1034 .sp .6
1035 .RS 4n
1036 The \fBset-prop\fR subcommand is used to set a property’s value(s) on the
1037 protocol.
1038 .sp
1039 The \fB-t\fR option (also \fB--temporary\fR) means
1040 that the setting is temporary and will not be persistent across reboots.
1041 .sp
1042 The \fB-p\fR option (also \fB--prop\fR) specifies the property name and
1043 value(s). The optional [+|-] syntax can be used to add/remove values from the
1044 current list of values on the property.
1045 The property name can be one of the following:
1046 .sp
1047 .ne 2
1048 .na
```

```
.RS 4n

\fBecn\fR
.ad
.RS 4n
Explicit congestion control (TCP-only)
Can be "never", "passive" or "active".
.RE

\fBextra_priv_ports\fR
.ad
.RS 4n
Additional privileged ports (SCTP, TCP or UDP)
.RE

\fBforwarding\fR
.ad
.RS 4n
Packet forwarding is enabled.
Can be "on" or "off".
.RE

\fBhoplimit\fR
.ad
.RS 4n
The IPv6 hoplimit.
.RE

\fBlargest_anon_port\fR
.ad
.RS 4n
Largest ephemeral port (SCTP, TCP or UDP)
.RE

\fBmax_buf\fR
\fIrecv_maxbuf\fR
.ad
.RS 4n
Maximum receive or send buffer size (ICMP, SCTP, TCP, or UDP).  This also
sets the upper limit for the \fBrecv_buf\fB and \fBsend_buf\fB properties.
Receive buffer size (ICMP, SCTP, TCP or UDP)
.RE

\fBrecv_buf\fR
.ad
.RS 4n
Default receive buffer size (ICMP, SCTP, TCP, or UDP).  The maximum value for
this property is controlled by the \fBmax_buf\fR property.
.RE

\fBsack\fR
.ad
.RS 4n
Selective acknowledgement (TCP).
Can be "active", "passive" or "never".
.RE

\fBsend_buf\fR
\fIsend_maxbuf\fR
.ad
.RS 4n
Default send buffer size (ICMP, SCTP, TCP, or UDP).  The maximum value for
this property is controlled by the \fBmax_buf\fR property.
Send buffer size (ICMP, SCTP, TCP or UDP)
.RE
```



```
\fBsmallest_anon_port\fR
.ad
.RS 4n
Smallest ephemeral port (SCTP, TCP or UDP)
.RE

\fBsmallest_nonpriv_port\fR
.ad
.RS 4n
Smallest non-privileged port (SCTP, TCP or UDP)
.RE

\fBttl\fR
.ad
.RS 4n
The IPv4 time-to-live.
.RE

.RE
.RE

.sp
.ne 2
.na
\fB\fBreset-prop\fR [\fB-t\fR] \fB-p\fR \fIprop\fR \fIprotocol\fR\fR
.ad
.sp .6
.RS 4n
The \fBreset-prop\fR subcommand is used to reset a protocol’s property
value to the default.
.sp
The \fB-t\fR option (also \fB--temporary\fR) means
that the disable is temporary and will not be persistent across reboots.
.sp
The \fB-p\fR option (also \fB--prop\fR) specifies the property name.
See the \fBset-prop\fR subcommand for the list of property names.
.sp

.RE

.sp
.ne 2
.na
\fB\fBshow-prop\fR [[\fB-c\fR]\fB-o\fR \fIfield\fR[,...]] [\fB-p\fR \fIprop\fR,.
.ad
.sp .6
.RS 4n
The \fBshow-prop\fR subcommand is used to display the property values
for one or all of the protocols.
.sp
The \fB-c\fR option (also \fB--parsable\fR) prints
the output in a parsable format.
.sp
The \fB-o\fR option (also \fB--output\fR) is used
to select which fields will be shown.  The field value can be one of the
following names:
.sp
.ne 2
.na
.RS 4n
\fBALL\fR
.ad
.RS 4n
Display all fields
.RE
```

```
.sp
.ne 2
.na
\fBPROTO\fR
.ad
.RS 4n
The name of the protocol
.RE

.sp
.ne 2
.na
\fBPROPERTY\fR
.ad
.RS 4n
The name of the property
.RE

.sp
.ne 2
.na
\fBPERM\fR
.ad
.RS 4n
If the property is readable ("r") and/or writable ("w").
.RE

.sp
.ne 2
.na
\fBCURRENT\fR
.ad
.RS 4n
The value of the property
.RE

.sp
.ne 2
.na
\fBPERSISTENT\fR
.ad
.RS 4n
The persistent value of the property
.RE

.sp
.ne 2
.na
\fBDEFAULT\fR
.ad
.RS 4n
The default value of the property
.RE

.sp
.ne 2
.na
\fBPOSSIBLE\fR
.ad
.RS 4n
The possible values for the property
.RE
.RE

.sp
```



```
The \fB-p\fR option (also \fB--prop\fR) is used
to specify which properties to display.  See the \fBset-prop\fR
subcommand for the list of property names.
.sp

.RE

.SH SEE ALSO
.sp
.LP
\fBifconfig\fR(1M), \fBdladm\fR(1M), \fBndd\fR(1M), \fBzonecfg\fR(1M),
\fBarp\fR(1M), \fBcfgadm\fR(1M), \fBif_mpadm\fR(1M), \fBnsswitch.conf\fR(4),
and \fBdhcp\fR(5).
```

```
**********************************************************
  153451 Mon Jul 29 18:04:14 2013
new/usr/src/uts/common/inet/ip/icmp.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2013 by Delphix. All rights reserved.
 */
/* 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);
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,
```

```
124                     uchar_t *invalp, uint_t *outlenp, uchar_t *outvalp,
125                     void *thisdg_attrs, cred_t *cr);
126 int             icmp_opt_get(conn_t *connp, int level, int name,
127                     uchar_t *ptr);
128 static int      icmp_output_newdst(conn_t *connp, mblk_t *data_mp, sin_t *sin,
129                     sin6_t *sin6, cred_t *cr, pid_t pid, ip_xmit_attr_t *ixa);
130 static mblk_t   *icmp_prepend_hdr(conn_t *, ip_xmit_attr_t *, const ip_pkt_t *,
131     const in6_addr_t *, const in6_addr_t *, uint32_t, mblk_t *, int *);
132 static mblk_t   *icmp_prepend_header_template(conn_t *, ip_xmit_attr_t *,
133     mblk_t *, const in6_addr_t *, uint32_t, int *);
134 static int      icmp_snmp_set(queue_t *q, t_scalar_t level, t_scalar_t name,
135                     uchar_t *ptr, int len);
136 static void     icmp_ud_err(queue_t *q, mblk_t *mp, t_scalar_t err);
137 static void     icmp_tpi_unbind(queue_t *q, mblk_t *mp);
138 static void     icmp_wput(queue_t *q, mblk_t *mp);
139 static void     icmp_wput_fallback(queue_t *q, mblk_t *mp);
140 static void     icmp_wput_other(queue_t *q, mblk_t *mp);
141 static void     icmp_wput_iocdata(queue_t *q, mblk_t *mp);
142 static void     icmp_wput_restricted(queue_t *q, mblk_t *mp);
143 static void     icmp_ulp_recv(conn_t *, mblk_t *, uint_t);

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

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

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

161 int             rawip_getsockname(sock_lower_handle_t, struct sockaddr *,
162                     socklen_t *, cred_t *);
163 int             rawip_getpeername(sock_lower_handle_t, struct sockaddr *,
164                     socklen_t *, cred_t *);

166 static struct module_info icmp_mod_info =  {
167         5707, "icmp", 1, INFPSZ, 512, 128
168 };
```

*unchanged_portion_omitted_*

```
216 static int
217 icmp_set_buf_prop(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
218     const char *ifname, const void *pval, uint_t flags)
219 {
220         return (mod_set_buf_prop(stack->netstack_icmp->is_propinfo_tbl,
221             stack, cr, pinfo, ifname, pval, flags));
222 }

224 static int
225 icmp_get_buf_prop(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
226     void *val, uint_t psize, uint_t flags)
227 {
228         return (mod_get_buf_prop(stack->netstack_icmp->is_propinfo_tbl, stack,
229             pinfo, ifname, val, psize, flags));
230 }

232 /*
233  * All of these are alterable, within the min/max values given, at run time.
234  *
```



```
235  * Note: All those tunables which do not start with "icmp_" are Committed and
236  * therefore are public. See PSARC 2010/080.
237  */
238 static mod_prop_info_t icmp_propinfo_tbl[] = {
239         /* tunable - 0 */
240         { "_wroff_extra", MOD_PROTO_RAWIP,
241             mod_set_uint32, mod_get_uint32,
242             {0, 128, 32}, {32} },

244         { "_ipv4_ttl", MOD_PROTO_RAWIP,
245             mod_set_uint32, mod_get_uint32,
246             {1, 255, 255}, {255} },

248         { "_ipv6_hoplimit", MOD_PROTO_RAWIP,
249             mod_set_uint32, mod_get_uint32,
250             {0, IPV6_MAX_HOPS, IPV6_DEFAULT_HOPS},
251             {IPV6_DEFAULT_HOPS} },

253         { "_bsd_compat", MOD_PROTO_RAWIP,
254             mod_set_boolean, mod_get_boolean,
255             {B_TRUE}, {B_TRUE} },

257         { "send_buf", MOD_PROTO_RAWIP,
258             icmp_set_buf_prop, icmp_get_buf_prop,
241         { "send_maxbuf", MOD_PROTO_RAWIP,
242             mod_set_uint32, mod_get_uint32,
259             {4096, 65536, 8192}, {8192} },

261         { "_xmit_lowat", MOD_PROTO_RAWIP,
262             mod_set_uint32, mod_get_uint32,
263             {0, 65536, 1024}, {1024} },

265         { "recv_buf", MOD_PROTO_RAWIP,
266             icmp_set_buf_prop, icmp_get_buf_prop,
249         { "recv_maxbuf", MOD_PROTO_RAWIP,
250             mod_set_uint32, mod_get_uint32,
267             {4096, 65536, 8192}, {8192} },

269         { "max_buf", MOD_PROTO_RAWIP,
253         { "_max_buf", MOD_PROTO_RAWIP,
270             mod_set_uint32, mod_get_uint32,
271             {65536, ULP_MAX_BUF, 256*1024}, {256*1024} },
255             {65536, 1024*1024*1024, 256*1024}, {256 * 1024} },

273         { "_pmtu_discovery", MOD_PROTO_RAWIP,
274             mod_set_boolean, mod_get_boolean,
275             {B_FALSE}, {B_FALSE} },

277         { "_sendto_ignerr", MOD_PROTO_RAWIP,
278             mod_set_boolean, mod_get_boolean,
279             {B_FALSE}, {B_FALSE} },

281         { "?", MOD_PROTO_RAWIP, NULL, mod_get_allprop, {0}, {0} },

283         { NULL, 0, NULL, NULL, {0}, {0} }
284 };
```

*unchanged_portion_omitted_*

```
**********************************************************
  533173 Mon Jul 29 18:04:15 2013
new/usr/src/uts/common/inet/ip/ip_if.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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 by Delphix. 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;
______unchanged_portion_omitted_

/*
 * process the SIOC{SET|GET}PROP ioctl's
 */
/* ARGSUSED */
```

```
8850 static void
8851 ip_sioctl_getsetprop(queue_t *q, mblk_t *mp)
8852 {
8853         struct iocblk   *iocp = (struct iocblk *)mp->b_rptr;
8854         mblk_t          *mp1 = mp->b_cont;
8855         mod_ioc_prop_t  *pioc;
8856         mod_prop_info_t *ptbl = NULL, *pinfo = NULL;
8857         ip_stack_t      *ipst;
8857         icmp_stack_t    *is;
8858         tcp_stack_t     *tcps;
8859         sctp_stack_t    *sctps;
8860         udp_stack_t     *us;
8858         netstack_t      *stack;
8862         void            *cbarg;
8859         cred_t          *cr;
8860         boolean_t       set;
8861         int             err;

8863         ASSERT(q->q_next == NULL);
8864         ASSERT(CONN_Q(q));

8866         if (!getset_ioctl_checks(mp)) {
8867                 miocnak(q, mp, 0, EINVAL);
8868                 return;
8869         }
8870         ipst = CONNQ_TO_IPST(q);
8871         stack = ipst->ips_netstack;
8872         pioc = (mod_ioc_prop_t *)mp1->b_rptr;

8874         switch (pioc->mpr_proto) {
8875         case MOD_PROTO_IP:
8876         case MOD_PROTO_IPV4:
8877         case MOD_PROTO_IPV6:
8878                 ptbl = ipst->ips_propinfo_tbl;
8883                 cbarg = ipst;
8879                 break;
8880         case MOD_PROTO_RAWIP:
8881                 ptbl = stack->netstack_icmp->is_propinfo_tbl;
8886                 is = stack->netstack_icmp;
8887                 ptbl = is->is_propinfo_tbl;
8888                 cbarg = is;
8882                 break;
8883         case MOD_PROTO_TCP:
8884                 ptbl = stack->netstack_tcp->tcps_propinfo_tbl;
8891                 tcps = stack->netstack_tcp;
8892                 ptbl = tcps->tcps_propinfo_tbl;
8893                 cbarg = tcps;
8885                 break;
8886         case MOD_PROTO_UDP:
8887                 ptbl = stack->netstack_udp->us_propinfo_tbl;
8896                 us = stack->netstack_udp;
8897                 ptbl = us->us_propinfo_tbl;
8898                 cbarg = us;
8888                 break;
8889         case MOD_PROTO_SCTP:
8890                 ptbl = stack->netstack_sctp->sctps_propinfo_tbl;
8901                 sctps = stack->netstack_sctp;
8902                 ptbl = sctps->sctps_propinfo_tbl;
8903                 cbarg = sctps;
8891                 break;
8892         default:
8893                 miocnak(q, mp, 0, EINVAL);
8894                 return;
8895         }

8897         pinfo = mod_prop_lookup(ptbl, pioc->mpr_name, pioc->mpr_proto);
```



```
8898         if (pinfo == NULL) {
8910         /* search for given property in respective protocol propinfo table */
8911         for (pinfo = ptbl; pinfo->mpi_name != NULL; pinfo++) {
8912                 if (strcmp(pinfo->mpi_name, pioc->mpr_name) == 0 &&
8913                     pinfo->mpi_proto == pioc->mpr_proto)
8914                         break;
8915         }
8916         if (pinfo->mpi_name == NULL) {
8899                 miocnak(q, mp, 0, ENOENT);
8900                 return;
8901         }

8903         set = (iocp->ioc_cmd == SIOCSETPROP) ? B_TRUE : B_FALSE;
8904         if (set && pinfo->mpi_setf != NULL) {
8905                 cr = msg_getcred(mp, NULL);
8906                 if (cr == NULL)
8907                         cr = iocp->ioc_cr;
8908                 err = pinfo->mpi_setf(stack, cr, pinfo, pioc->mpr_ifname,
8926                 err = pinfo->mpi_setf(cbarg, cr, pinfo, pioc->mpr_ifname,
8909                     pioc->mpr_val, pioc->mpr_flags);
8910         } else if (!set && pinfo->mpi_getf != NULL) {
8911                 err = pinfo->mpi_getf(stack, pinfo, pioc->mpr_ifname,
8929                 err = pinfo->mpi_getf(cbarg, pinfo, pioc->mpr_ifname,
8912                     pioc->mpr_val, pioc->mpr_valsize, pioc->mpr_flags);
8913         } else {
8914                 err = EPERM;
8915         }

8917         if (err != 0) {
8918                 miocnak(q, mp, 0, err);
8919         } else {
8920                 if (set)
8921                         miocack(q, mp, 0, 0);
8922                 else    /* For get, we need to return back the data */
8923                         miocack(q, mp, iocp->ioc_count, 0);
8924         }
8925 }

8927 /*
8928  * process the legacy ND_GET, ND_SET ioctl just for {ip|ip6}_forwarding
8929  * as several routing daemons have unfortunately used this 'unpublished'
8930  * but well-known ioctls.
8931  */
8932 /* ARGSUSED */
8933 static void
8934 ip_process_legacy_nddprop(queue_t *q, mblk_t *mp)
8935 {
8936         struct iocblk   *iocp = (struct iocblk *)mp->b_rptr;
8937         mblk_t          *mp1 = mp->b_cont;
8938         char            *pname, *pval, *buf;
8939         uint_t          bufsize, proto;
8940         mod_prop_info_t *pinfo = NULL;
8958         mod_prop_info_t *ptbl = NULL, *pinfo = NULL;
8941         ip_stack_t      *ipst;
8942         int             err = 0;

8944         ASSERT(CONN_Q(q));
8945         ipst = CONNQ_TO_IPST(q);

8947         if (iocp->ioc_count == 0 || mp1 == NULL) {
8948                 miocnak(q, mp, 0, EINVAL);
8949                 return;
8950         }

8952         mp1->b_datap->db_lim[-1] = '\0';        /* Force null termination */
8953         pval = buf = pname = (char *)mp1->b_rptr;
```

```
8954         bufsize = MBLKL(mp1);

8956         if (strcmp(pname, "ip_forwarding") == 0) {
8957                 pname = "forwarding";
8958                 proto = MOD_PROTO_IPV4;
8959         } else if (strcmp(pname, "ip6_forwarding") == 0) {
8960                 pname = "forwarding";
8961                 proto = MOD_PROTO_IPV6;
8962         } else {
8963                 miocnak(q, mp, 0, EINVAL);
8964                 return;
8965         }

8967         pinfo = mod_prop_lookup(ipst->ips_propinfo_tbl, pname, proto);
8985         ptbl = ipst->ips_propinfo_tbl;
8986         for (pinfo = ptbl; pinfo->mpi_name != NULL; pinfo++) {
8987                 if (strcmp(pinfo->mpi_name, pname) == 0 &&
8988                     pinfo->mpi_proto == proto)
8989                         break;
8990         }

8992         ASSERT(pinfo->mpi_name != NULL);

8969         switch (iocp->ioc_cmd) {
8970         case ND_GET:
8971                 if ((err = pinfo->mpi_getf(ipst->ips_netstack, pinfo, NULL, buf,
8972                     bufsize, 0)) == 0) {
8996                 if ((err = pinfo->mpi_getf(ipst, pinfo, NULL, buf, bufsize,
8997                     0)) == 0) {
8973                         miocack(q, mp, iocp->ioc_count, 0);
8974                         return;
8975                 }
8976                 break;
8977         case ND_SET:
8978                 /*
8979                  * buffer will have property name and value in the following
8980                  * format,
8981                  * <property name>'\0'<property value>'\0', extract them;
8982                  */
8983                 while (*pval++)
8984                         noop;

8986                 if (!*pval || pval >= (char *)mp1->b_wptr) {
8987                         err = EINVAL;
8988                 } else if ((err = pinfo->mpi_setf(ipst->ips_netstack, NULL,
8989                     pinfo, NULL, pval, 0)) == 0) {
9013                 } else if ((err = pinfo->mpi_setf(ipst, NULL, pinfo, NULL,
9014                     pval, 0)) == 0) {
8990                         miocack(q, mp, 0, 0);
8991                         return;
8992                 }
8993                 break;
8994         default:
8995                 err = EINVAL;
8996                 break;
8997         }
8998         miocnak(q, mp, 0, err);
8999 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   27327 Mon Jul 29 18:04:18 2013
new/usr/src/uts/common/inet/ip/ip_tunables.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2013 by Delphix. All rights reserved.
 */
/* Copyright (c) 1990 Mentat Inc. */

#include <inet/ip.h>
#include <inet/ip6.h>
#include <inet/ip_if.h>
#include <inet/ip_ire.h>
#include <inet/ipclassifier.h>
#include <inet/ip_impl.h>
#include <inet/tunables.h>
#include <sys/sunddi.h>
#include <sys/policy.h>

/* How long, in seconds, we allow frags to hang around. */
#define IP_REASM_TIMEOUT        15
#define IPV6_REASM_TIMEOUT      60

/*
 * Set ip{,6}_forwarding values. If the value is being set on an ill,
 * find the ill and set the value on it. On the other hand if we are modifying
 * global property, modify the global value and set the value on all the ills.
 */
/* ARGSUSED */
static int
ip_set_forwarding(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
ip_set_forwarding(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pval, uint_t flags)
{
        char                    *end;
        unsigned long           new_value;
        boolean_t               per_ill, isv6;
        ill_walk_context_t      ctx;
        ill_t                   *ill;
        ip_stack_t              *ipst = stack->netstack_ip;
        ip_stack_t              *ipst = (ip_stack_t *)cbarg;
```



```
        if (flags & MOD_PROP_DEFAULT) {
                new_value = pinfo->prop_def_bval;
        } else {
                if (ddi_strtoul(pval, &end, 10, &new_value) != 0 ||
                    *end != '\0')
                        return (EINVAL);
                if (new_value != B_TRUE && new_value != B_FALSE)
                        return (EINVAL);
        }

        per_ill = (ifname != NULL && ifname[0] != '\0');
        /*
         * if it's not per ill then set the global property and bring all the
         * ills up to date with the new global value.
         */
        if (!per_ill)
                pinfo->prop_cur_bval = (new_value == 1 ? B_TRUE : B_FALSE);

        isv6 = (pinfo->mpi_proto == MOD_PROTO_IPV6 ? B_TRUE : B_FALSE);
        rw_enter(&ipst->ips_ill_g_lock, RW_READER);
        if (isv6)
                ill = ILL_START_WALK_V6(&ctx, ipst);
        else
                ill = ILL_START_WALK_V4(&ctx, ipst);

        for (; ill != NULL; ill = ill_next(&ctx, ill)) {
                /*
                 * if the property needs to be set on a particular
                 * interface, look for that interface.
                 */
                if (per_ill && strcmp(ifname, ill->ill_name) != 0)
                        continue;
                (void) ill_forward_set(ill, new_value != 0);
        }
        rw_exit(&ipst->ips_ill_g_lock);

        return (0);
}

static int
ip_get_forwarding(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
ip_get_forwarding(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
    void *pval, uint_t pr_size, uint_t flags)
{
        boolean_t               value;
        ill_walk_context_t      ctx;
        ill_t                   *ill;
        ip_stack_t              *ipst = stack->netstack_ip;
        ip_stack_t              *ipst =  (ip_stack_t *)cbarg;
        boolean_t               get_def = (flags & MOD_PROP_DEFAULT);
        boolean_t               get_perm = (flags & MOD_PROP_PERM);
        boolean_t               isv6;
        size_t                  nbytes = 0;

        if (get_perm) {
                nbytes = snprintf(pval, pr_size, "%d", MOD_PROP_PERM_RW);
                goto ret;
        } else if (get_def) {
                nbytes = snprintf(pval, pr_size, "%d", pinfo->prop_def_bval);
                goto ret;
        }

        /*
         * if per interface value is not asked for return the current
         * global value
```

```
121          */
122         if (ifname == NULL || ifname[0] == '\0') {
123                 nbytes = snprintf(pval, pr_size, "%d", pinfo->prop_cur_bval);
124                 goto ret;
125         }

127         isv6 = (pinfo->mpi_proto == MOD_PROTO_IPV6 ? B_TRUE : B_FALSE);
128         rw_enter(&ipst->ips_ill_g_lock, RW_READER);
129         if (isv6)
130                 ill = ILL_START_WALK_V6(&ctx, ipst);
131         else
132                 ill = ILL_START_WALK_V4(&ctx, ipst);
133         for (; ill != NULL; ill = ill_next(&ctx, ill)) {
134                 /*
135                  * if the property needs to be obtained on a particular
136                  * interface, look for that interface.
137                  */
138                 if (strcmp(ifname, ill->ill_name) == 0)
139                         break;
140         }
141         if (ill == NULL) {
142                 rw_exit(&ipst->ips_ill_g_lock);
143                 return (ENXIO);
144         }
145         value = ((ill->ill_flags & ILLF_ROUTER) ? B_TRUE : B_FALSE);
146         rw_exit(&ipst->ips_ill_g_lock);
147         nbytes = snprintf(pval, pr_size, "%d", value);
148 ret:
149         if (nbytes >= pr_size)
150                 return (ENOBUFS);
151         return (0);
152 }

154 /*
155  * 'ip_debug' is a global variable. So, we will be modifying the global
156  * variable here.
157  */
158 /* ARGSUSED */
159 int
160 ip_set_debug(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
159 ip_set_debug(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
161     const char *ifname, const void* pval, uint_t flags)
162 {
163         unsigned long   new_value;
164         int             err;

166         if (cr != NULL && secpolicy_net_config(cr, B_FALSE) != 0)
167                 return (EPERM);

169         if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
170                 return (err);
171         ip_debug = (uint32_t)new_value;
172         return (0);
173 }

175 /*
176  * ip_debug is a global property. For default, permission and value range
177  * we retrieve the value from 'pinfo'. However for the current value we
178  * retrieve the value from the global variable 'ip_debug'
179  */
180 /* ARGSUSED */
181 int
182 ip_get_debug(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
181 ip_get_debug(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
183     void *pval, uint_t psize, uint_t flags)
184 {
```



```
185         boolean_t       get_def = (flags & MOD_PROP_DEFAULT);
186         boolean_t       get_perm = (flags & MOD_PROP_PERM);
187         boolean_t       get_range = (flags & MOD_PROP_POSSIBLE);
188         size_t          nbytes;

190         bzero(pval, psize);
191         if (get_perm)
192                 nbytes = snprintf(pval, psize, "%u", MOD_PROP_PERM_RW);
193         else if (get_range)
194                 nbytes = snprintf(pval, psize, "%u-%u",
195                     pinfo->prop_min_uval, pinfo->prop_max_uval);
196         else if (get_def)
197                 nbytes = snprintf(pval, psize, "%u", pinfo->prop_def_uval);
198         else
199                 nbytes = snprintf(pval, psize, "%u", ip_debug);
200         if (nbytes >= psize)
201                 return (ENOBUFS);
202         return (0);
203 }

205 /*
206  * Set the CGTP (multirouting) filtering status. If the status is changed
207  * from active to transparent or from transparent to active, forward the
208  * new status to the filtering module (if loaded).
209  */
210 /* ARGSUSED */
211 static int
212 ip_set_cgtp_filter(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
211 ip_set_cgtp_filter(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
213     const char *ifname, const void* pval, uint_t flags)
214 {
215         unsigned long   new_value;
216         ip_stack_t      *ipst = stack->netstack_ip;
215         ip_stack_t      *ipst = (ip_stack_t *)cbarg;
217         char            *end;

219         if (flags & MOD_PROP_DEFAULT) {
220                 new_value = pinfo->prop_def_bval;
221         } else {
222                 if (ddi_strtoul(pval, &end, 10, &new_value) != 0 ||
223                     *end != '\0' || new_value > 1) {
224                         return (EINVAL);
225                 }
226         }
227         if (!pinfo->prop_cur_bval && new_value) {
228                 cmn_err(CE_NOTE, "IP: enabling CGTP filtering%s",
229                     ipst->ips_ip_cgtp_filter_ops == NULL ?
230                     " (module not loaded)" : "");
231         }
232         if (pinfo->prop_cur_bval && !new_value) {
233                 cmn_err(CE_NOTE, "IP: disabling CGTP filtering%s",
234                     ipst->ips_ip_cgtp_filter_ops == NULL ?
235                     " (module not loaded)" : "");
236         }
237         if (ipst->ips_ip_cgtp_filter_ops != NULL) {
238                 int     res;
239                 netstackid_t stackid = ipst->ips_netstack->netstack_stackid;

241                 res = ipst->ips_ip_cgtp_filter_ops->cfo_change_state(stackid,
242                     new_value);
243                 if (res)
244                         return (res);
245         }
246         pinfo->prop_cur_bval = (new_value == 1 ? B_TRUE : B_FALSE);
247         ill_set_inputfn_all(ipst);
248         return (0);
```

```
249 }

251 /*
252  * Retrieve the default MTU or min-max MTU range for a given interface.
253  *
254  *  -- ill_max_frag value tells us the maximum MTU that can be handled by the
255  *     datalink. This value is advertised by the driver via DLPI messages
256  *     (DL_NOTE_SDU_SIZE/DL_INFO_ACK).
257  *
258  *  -- ill_current_frag for the most link-types will be same as ill_max_frag
259  *     to begin with. However it is dynamically computed for some link-types
260  *     like tunnels, based on the tunnel PMTU.
261  *
262  *  -- ill_mtu is the user set MTU using SIOCSLIFMTU and must lie between
263  *     (IPV6_MIN_MTU/IP_MIN_MTU) and ill_max_frag.
264  *
265  *  -- ill_user_mtu is set by in.ndpd using SIOCSLIFLNKINFO and must lie between
266  *     (IPV6_MIN_MTU/IP_MIN_MTU) and ill_max_frag.
267  */
268 int
269 ip_get_mtu(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
268 ip_get_mtu(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
270     void *pval, uint_t psize, uint_t flags)
271 {
272         ill_walk_context_t      ctx;
273         ill_t                   *ill;
274         ip_stack_t              *ipst = stack->netstack_ip;
273         ip_stack_t              *ipst =  (ip_stack_t *)cbarg;
275         boolean_t               isv6;
276         uint32_t                max_mtu, def_mtu;
277         size_t                  nbytes = 0;

279         if (!(flags & (MOD_PROP_DEFAULT|MOD_PROP_POSSIBLE)))
280                 return (ENOTSUP);

282         if (ifname == NULL || ifname[0] == '\0')
283                 return (ENOTSUP);

285         isv6 = (pinfo->mpi_proto == MOD_PROTO_IPV6 ? B_TRUE : B_FALSE);
286         rw_enter(&ipst->ips_ill_g_lock, RW_READER);
287         if (isv6)
288                 ill = ILL_START_WALK_V6(&ctx, ipst);
289         else
290                 ill = ILL_START_WALK_V4(&ctx, ipst);
291         for (; ill != NULL; ill = ill_next(&ctx, ill)) {
292                 if (strcmp(ifname, ill->ill_name) == 0)
293                         break;
294         }
295         if (ill == NULL) {
296                 rw_exit(&ipst->ips_ill_g_lock);
297                 return (ENXIO);
298         }
299         max_mtu = ill->ill_max_frag;
300         def_mtu = ill->ill_current_frag;
301         rw_exit(&ipst->ips_ill_g_lock);

303         if (flags & MOD_PROP_DEFAULT) {
304                 nbytes = snprintf(pval, psize, "%u", def_mtu);
305         } else if (flags & MOD_PROP_POSSIBLE) {
306                 uint32_t        min_mtu;

308                 min_mtu = isv6 ? IPV6_MIN_MTU : IP_MIN_MTU;
309                 nbytes = snprintf(pval, psize, "%u-%u", min_mtu, max_mtu);
310         } else {
311                 return (ENOTSUP);
312         }
```



```
314         if (nbytes >= psize)
315                 return (ENOBUFS);
316         return (0);
317 }
______unchanged_portion_omitted_

354 /* ARGSUSED */
355 static int
356 ip_set_src_multihoming(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
355 ip_set_src_multihoming(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
357     const char *ifname, const void* pval, uint_t flags)
358 {
359         unsigned long   new_value, old_value;
360         boolean_t       isv6;
361         ip_stack_t      *ipst = stack->netstack_ip;
360         ip_stack_t      *ipst = (ip_stack_t *)cbarg;
362         int             err;

364         old_value = pinfo->prop_cur_uval;

366         if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
367                 return (err);
368         pinfo->prop_cur_uval = new_value;
369         isv6 = (strcmp(pinfo->mpi_name, "ip6_strict_src_multihoming") == 0);
370         ip_set_src_multihoming_common(new_value, old_value, isv6, ipst);
371         return (0);
372 }


375 /* ARGSUSED */
376 static int
377 ip_set_hostmodel(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
376 ip_set_hostmodel(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
378     const char *ifname, const void* pval, uint_t flags)
379 {
380         ip_hostmodel_t  new_value, old_value;
381         ip_stack_t      *ipst = stack->netstack_ip;
380         ip_stack_t      *ipst = (ip_stack_t *)cbarg;
382         uint32_t        old_src_multihoming;
383         int             err;
384         ulong_t         tmp;
385         boolean_t       isv6;

387         old_value = pinfo->prop_cur_uval;

389         if ((err = mod_uint32_value(pval, pinfo, flags, &tmp)) != 0)
390                 return (err);
391         new_value = tmp;
392         pinfo->prop_cur_uval = new_value;

394         switch (old_value) {
395         case IP_WEAK_ES:
396                 old_src_multihoming = 0;
397                 break;
398         case IP_SRC_PRI_ES:
399                 old_src_multihoming = 1;
400                 break;
401         case IP_STRONG_ES:
402                 old_src_multihoming = 2;
403                 break;
404         default:
405                 ASSERT(0);
406                 old_src_multihoming = IP_MAXVAL_ES;
407                 break;
408         }
```

```
409         /*
410          * Changes to src_multihoming may require ire's to be rebound/unbound,
411          * and also require generation number resets. Changes to dst_multihoming
412          * require a simple reset of the value.
413          */
414         isv6 = (pinfo->mpi_proto == MOD_PROTO_IPV6);
415         if (new_value != old_value) {
416                 switch (new_value) {
417                 case IP_WEAK_ES:
418                         ip_set_src_multihoming_common(0, old_src_multihoming,
419                             isv6, ipst);
420                         if (isv6)
421                                 ipst->ips_ipv6_strict_dst_multihoming = 0;
422                         else
423                                 ipst->ips_ip_strict_dst_multihoming = 0;
424                         break;
425                 case IP_SRC_PRI_ES:
426                         ip_set_src_multihoming_common(1, old_src_multihoming,
427                             isv6, ipst);
428                         if (isv6)
429                                 ipst->ips_ipv6_strict_dst_multihoming = 0;
430                         else
431                                 ipst->ips_ip_strict_dst_multihoming = 0;
432                         break;
433                 case IP_STRONG_ES:
434                         ip_set_src_multihoming_common(2, old_src_multihoming,
435                             isv6, ipst);
436                         if (isv6)
437                                 ipst->ips_ipv6_strict_dst_multihoming = 1;
438                         else
439                                 ipst->ips_ip_strict_dst_multihoming = 1;
440                         break;
441                 default:
442                         return (EINVAL);
443                 }
444         }
445         return (0);
446 }

448 /* ARGSUSED */
449 int
450 ip_get_hostmodel(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
449 ip_get_hostmodel(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
451     void *pval, uint_t psize, uint_t flags)
452 {
453         boolean_t       isv6 = (pinfo->mpi_proto == MOD_PROTO_IPV6);
454         ip_stack_t      *ipst = stack->netstack_ip;
453         ip_stack_t      *ipst = cbarg;
455         ip_hostmodel_t  hostmodel;

457         if (psize < sizeof (hostmodel))
458                 return (ENOBUFS);
459         bzero(pval, psize);
460         if (!isv6) {
461                 if (ipst->ips_ip_strict_src_multihoming == 0 &&
462                     ipst->ips_ip_strict_dst_multihoming == 0)
463                         hostmodel = IP_WEAK_ES;
464                 else if (ipst->ips_ip_strict_src_multihoming == 1 &&
465                     ipst->ips_ip_strict_dst_multihoming == 0)
466                         hostmodel = IP_SRC_PRI_ES;
467                 else if (ipst->ips_ip_strict_src_multihoming == 2 &&
468                     ipst->ips_ip_strict_dst_multihoming == 1)
469                         hostmodel = IP_STRONG_ES;
470                 else
471                         hostmodel = IP_MAXVAL_ES;
472         } else {
```



```
473                 if (ipst->ips_ipv6_strict_src_multihoming == 0 &&
474                     ipst->ips_ipv6_strict_dst_multihoming == 0)
475                         hostmodel = IP_WEAK_ES;
476                 else if (ipst->ips_ipv6_strict_src_multihoming == 1 &&
477                     ipst->ips_ipv6_strict_dst_multihoming == 0)
478                         hostmodel = IP_SRC_PRI_ES;
479                 else if (ipst->ips_ipv6_strict_src_multihoming == 2 &&
480                     ipst->ips_ipv6_strict_dst_multihoming == 1)
481                         hostmodel = IP_STRONG_ES;
482                 else
483                         hostmodel = IP_MAXVAL_ES;
484         }
485         bcopy(&hostmodel, pval, sizeof (hostmodel));
486         return (0);
487 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10676 Mon Jul 29 18:04:20 2013
new/usr/src/uts/common/inet/sctp/sctp_tunables.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <inet/ip.h>
#include <inet/ip6.h>
#include <inet/sctp/sctp_stack.h>
#include <inet/sctp/sctp_impl.h>
#include <sys/sunddi.h>

/* Max size IP datagram is 64k - 1 */
#define SCTP_MSS_MAX_IPV4 (IP_MAXPACKET - (sizeof (ipha_t) + \
                                           sizeof (sctp_hdr_t)))
#define SCTP_MSS_MAX_IPV6 (IP_MAXPACKET - (sizeof (ip6_t) + \
                                           sizeof (sctp_hdr_t)))
/* Max of the above */
#define SCTP_MSS_MAX    SCTP_MSS_MAX_IPV4

/*
 * returns the current list of listener limit configuration.
 */
/* ARGSUSED */
static int
sctp_listener_conf_get(netstack_t *stack, mod_prop_info_t *pinfo,
    const char *ifname, void *val, uint_t psize, uint_t flags)
sctp_listener_conf_get(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
    void *val, uint_t psize, uint_t flags)
{
        sctp_stack_t    *sctps = stack->netstack_sctp;
        sctp_stack_t    *sctps = (sctp_stack_t *)cbarg;
        sctp_listener_t *sl;
        char            *pval = val;
        size_t          nbytes = 0, tbytes = 0;
        uint_t          size;
        int             err = 0;
```



```
        bzero(pval, psize);
        size = psize;

        if (flags & (MOD_PROP_DEFAULT|MOD_PROP_PERM|MOD_PROP_POSSIBLE))
                return (0);

        mutex_enter(&sctps->sctps_listener_conf_lock);
        for (sl = list_head(&sctps->sctps_listener_conf); sl != NULL;
            sl = list_next(&sctps->sctps_listener_conf, sl)) {
                if (psize == size)
                        nbytes = snprintf(pval, size, "%d:%d",  sl->sl_port,
                            sl->sl_ratio);
                else
                        nbytes = snprintf(pval, size, ",%d:%d",  sl->sl_port,
                            sl->sl_ratio);
                size -= nbytes;
                pval += nbytes;
                tbytes += nbytes;
                if (tbytes >= psize) {
                        /* Buffer overflow, stop copying information */
                        err = ENOBUFS;
                        break;
                }
        }

        mutex_exit(&sctps->sctps_listener_conf_lock);
        return (err);
}

/*
 * add a new listener limit configuration.
 */
/* ARGSUSED */
static int
sctp_listener_conf_add(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
sctp_listener_conf_add(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pval, uint_t flags)
{
        sctp_listener_t *new_sl;
        sctp_listener_t *sl;
        long            lport;
        long            ratio;
        char            *colon;
        sctp_stack_t    *sctps = stack->netstack_sctp;
        sctp_stack_t    *sctps = (sctp_stack_t *)cbarg;

        if (flags & MOD_PROP_DEFAULT)
                return (ENOTSUP);

        if (ddi_strtol(pval, &colon, 10, &lport) != 0 || lport <= 0 ||
            lport > USHRT_MAX || *colon != ':') {
                return (EINVAL);
        }
        if (ddi_strtol(colon + 1, NULL, 10, &ratio) != 0 || ratio <= 0)
                return (EINVAL);

        mutex_enter(&sctps->sctps_listener_conf_lock);
        for (sl = list_head(&sctps->sctps_listener_conf); sl != NULL;
            sl = list_next(&sctps->sctps_listener_conf, sl)) {
                /* There is an existing entry, so update its ratio value. */
                if (sl->sl_port == lport) {
                        sl->sl_ratio = ratio;
                        mutex_exit(&sctps->sctps_listener_conf_lock);
                        return (0);
                }
        }
```

```
121         if ((new_sl = kmem_alloc(sizeof (sctp_listener_t), KM_NOSLEEP)) ==
122             NULL) {
123                 mutex_exit(&sctps->sctps_listener_conf_lock);
124                 return (ENOMEM);
125         }

127         new_sl->sl_port = lport;
128         new_sl->sl_ratio = ratio;
129         list_insert_tail(&sctps->sctps_listener_conf, new_sl);
130         mutex_exit(&sctps->sctps_listener_conf_lock);
131         return (0);
132 }

134 /*
135  * remove a listener limit configuration.
136  */
137 /* ARGSUSED */
138 static int
139 sctp_listener_conf_del(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
138 sctp_listener_conf_del(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
140     const char *ifname, const void* pval, uint_t flags)
141 {
142         sctp_listener_t *sl;
143         long            lport;
144         sctp_stack_t    *sctps = stack->netstack_sctp;
143         sctp_stack_t    *sctps = (sctp_stack_t *)cbarg;

146         if (flags & MOD_PROP_DEFAULT)
147                 return (ENOTSUP);

149         if (ddi_strtol(pval, NULL, 10, &lport) != 0 || lport <= 0 ||
150             lport > USHRT_MAX) {
151                 return (EINVAL);
152         }
153         mutex_enter(&sctps->sctps_listener_conf_lock);
154         for (sl = list_head(&sctps->sctps_listener_conf); sl != NULL;
155             sl = list_next(&sctps->sctps_listener_conf, sl)) {
156                 if (sl->sl_port == lport) {
157                         list_remove(&sctps->sctps_listener_conf, sl);
158                         mutex_exit(&sctps->sctps_listener_conf_lock);
159                         kmem_free(sl, sizeof (sctp_listener_t));
160                         return (0);
161                 }
162         }
163         mutex_exit(&sctps->sctps_listener_conf_lock);
164         return (ESRCH);
165 }

167 static int
168 sctp_set_buf_prop(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
169     const char *ifname, const void *pval, uint_t flags)
170 {
171         return (mod_set_buf_prop(stack->netstack_sctp->sctps_propinfo_tbl,
172             stack, cr, pinfo, ifname, pval, flags));
173 }

175 static int
176 sctp_get_buf_prop(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
177     void *val, uint_t psize, uint_t flags)
178 {
179         return (mod_get_buf_prop(stack->netstack_sctp->sctps_propinfo_tbl,
180             stack, pinfo, ifname, val, psize, flags));
181 }

183 /*
```



```
184  * All of these are alterable, within the min/max values given, at run time.
185  *
186  * Note: All those tunables which do not start with "_" are Committed and
187  * therefore are public. See PSARC 2010/080.
188  */
189 mod_prop_info_t sctp_propinfo_tbl[] = {
190         { "_max_init_retr", MOD_PROTO_SCTP,
191             mod_set_uint32, mod_get_uint32,
192             {0, 128, 8}, {8} },

194         { "_pa_max_retr", MOD_PROTO_SCTP,
195             mod_set_uint32, mod_get_uint32,
196             {1, 128, 10}, {10} },

198         { "_pp_max_retr", MOD_PROTO_SCTP,
199             mod_set_uint32, mod_get_uint32,
200             {1, 128, 5}, {5} },

202         { "_cwnd_max", MOD_PROTO_SCTP,
203             mod_set_uint32, mod_get_uint32,
204             {128, ULP_MAX_BUF, 1024*1024}, {1024*1024} },
187             {128, (1<<30), 1024*1024}, {1024*1024} },

206         { "smallest_nonpriv_port", MOD_PROTO_SCTP,
207             mod_set_uint32, mod_get_uint32,
208             {1024, (32*1024), 1024}, {1024} },

210         { "_ipv4_ttl", MOD_PROTO_SCTP,
211             mod_set_uint32, mod_get_uint32,
212             {1, 255, 64}, {64} },

214         { "_heartbeat_interval", MOD_PROTO_SCTP,
215             mod_set_uint32, mod_get_uint32,
216             {0, 1*DAYS, 30*SECONDS}, {30*SECONDS} },

218         { "_initial_mtu", MOD_PROTO_SCTP,
219             mod_set_uint32, mod_get_uint32,
220             {68, 65535, 1500}, {1500} },

222         { "_mtu_probe_interval", MOD_PROTO_SCTP,
223             mod_set_uint32, mod_get_uint32,
224             {0, 1*DAYS, 10*MINUTES}, {10*MINUTES} },

226         { "_new_secret_interval", MOD_PROTO_SCTP,
227             mod_set_uint32, mod_get_uint32,
228             {0, 1*DAYS, 2*MINUTES}, {2*MINUTES} },

230         /* tunable - 10 */
231         { "_deferred_ack_interval", MOD_PROTO_SCTP,
232             mod_set_uint32, mod_get_uint32,
233             {10*MS, 1*MINUTES, 100*MS}, {100*MS} },

235         { "_snd_lowat_fraction", MOD_PROTO_SCTP,
236             mod_set_uint32, mod_get_uint32,
237             {0, 16, 0}, {0} },

239         { "_ignore_path_mtu", MOD_PROTO_SCTP,
240             mod_set_boolean, mod_get_boolean,
241             {B_FALSE}, {B_FALSE} },

243         { "_initial_ssthresh", MOD_PROTO_SCTP,
244             mod_set_uint32, mod_get_uint32,
245             {1024, UINT32_MAX, SCTP_RECV_HIWATER}, { SCTP_RECV_HIWATER} },

247         { "smallest_anon_port", MOD_PROTO_SCTP,
248             mod_set_uint32, mod_get_uint32,
```

```
249             {1024, ULP_MAX_PORT, 32*1024}, {32*1024} },

251         { "largest_anon_port", MOD_PROTO_SCTP,
252             mod_set_uint32, mod_get_uint32,
253             {1024, ULP_MAX_PORT, ULP_MAX_PORT}, {ULP_MAX_PORT} },

255         { "send_buf", MOD_PROTO_SCTP,
256             sctp_set_buf_prop, sctp_get_buf_prop,
257             {SCTP_XMIT_LOWATER,  ULP_MAX_BUF,  SCTP_XMIT_HIWATER},
238         { "send_maxbuf", MOD_PROTO_SCTP,
239             mod_set_uint32, mod_get_uint32,
240             {SCTP_XMIT_LOWATER,  (1<<30),  SCTP_XMIT_HIWATER},
258             {SCTP_XMIT_HIWATER} },

260         { "_xmit_lowat", MOD_PROTO_SCTP,
261             mod_set_uint32, mod_get_uint32,
262             {SCTP_XMIT_LOWATER,  ULP_MAX_BUF,  SCTP_XMIT_LOWATER},
245             {SCTP_XMIT_LOWATER,  (1<<30),  SCTP_XMIT_LOWATER},
263             {SCTP_XMIT_LOWATER} },

265         { "recv_buf", MOD_PROTO_SCTP,
266             sctp_set_buf_prop, sctp_get_buf_prop,
267             {SCTP_RECV_LOWATER,  ULP_MAX_BUF,  SCTP_RECV_HIWATER},
248         { "recv_maxbuf", MOD_PROTO_SCTP,
249             mod_set_uint32, mod_get_uint32,
250             {SCTP_RECV_LOWATER,  (1<<30),  SCTP_RECV_HIWATER},
268             {SCTP_RECV_HIWATER} },

270         { "max_buf", MOD_PROTO_SCTP,
253         { "_max_buf", MOD_PROTO_SCTP,
271             mod_set_uint32, mod_get_uint32,
272             {8192, ULP_MAX_BUF, 1024*1024}, {1024*1024} },
255             {8192, (1<<30), 1024*1024}, {1024*1024} },

274         /* tunable - 20 */
275         { "_rtt_updates", MOD_PROTO_SCTP,
276             mod_set_uint32, mod_get_uint32,
277             {0, 65536, 20}, {20} },

279         { "_ipv6_hoplimit", MOD_PROTO_SCTP,
280             mod_set_uint32, mod_get_uint32,
281             {0, IPV6_MAX_HOPS, IPV6_DEFAULT_HOPS}, {IPV6_DEFAULT_HOPS} },

283         { "_rto_min", MOD_PROTO_SCTP,
284             mod_set_uint32, mod_get_uint32,
285             {500*MS, 60*SECONDS, 1*SECONDS}, {1*SECONDS} },

287         { "_rto_max", MOD_PROTO_SCTP,
288             mod_set_uint32, mod_get_uint32,
289             {1*SECONDS, 60000*SECONDS, 60*SECONDS}, {60*SECONDS} },

291         { "_rto_initial", MOD_PROTO_SCTP,
292             mod_set_uint32, mod_get_uint32,
293             {1*SECONDS, 60000*SECONDS, 3*SECONDS}, {3*SECONDS} },

295         { "_cookie_life", MOD_PROTO_SCTP,
296             mod_set_uint32, mod_get_uint32,
297             {10*MS, 60000*SECONDS, 60*SECONDS}, {60*SECONDS} },

299         { "_max_in_streams", MOD_PROTO_SCTP,
300             mod_set_uint32, mod_get_uint32,
301             {1, UINT16_MAX, 32}, {32} },

303         { "_initial_out_streams", MOD_PROTO_SCTP,
304             mod_set_uint32, mod_get_uint32,
305             {1, UINT16_MAX, 32}, {32} },
```



```
307         { "_shutack_wait_bound", MOD_PROTO_SCTP,
308             mod_set_uint32, mod_get_uint32,
309             {0, 300*SECONDS, 60*SECONDS}, {60*SECONDS} },

311         { "_maxburst", MOD_PROTO_SCTP,
312             mod_set_uint32, mod_get_uint32,
313             {2, 8, 4}, {4} },

315         /* tunable - 30 */
316         { "_addip_enabled", MOD_PROTO_SCTP,
317             mod_set_boolean, mod_get_boolean,
318             {B_FALSE}, {B_FALSE} },

320         { "_recv_hiwat_minmss", MOD_PROTO_SCTP,
321             mod_set_uint32, mod_get_uint32,
322             {1, 65536, 4}, {4} },

324         { "_slow_start_initial", MOD_PROTO_SCTP,
325             mod_set_uint32, mod_get_uint32,
326             {1, 16, 4}, {4} },

328         { "_slow_start_after_idle", MOD_PROTO_SCTP,
329             mod_set_uint32, mod_get_uint32,
330             {1, 16384, 4}, {4} },

332         { "_prsctp_enabled", MOD_PROTO_SCTP,
333             mod_set_boolean, mod_get_boolean,
334             {B_TRUE}, {B_TRUE} },

336         { "_fast_rxt_thresh", MOD_PROTO_SCTP,
337             mod_set_uint32, mod_get_uint32,
338             {1, 10000, 3}, {3} },

340         { "_deferred_acks_max", MOD_PROTO_SCTP,
341             mod_set_uint32, mod_get_uint32,
342             { 1, 16, 2}, {2} },

344         /*
345          * sctp_wroff_xtra is the extra space in front of SCTP/IP header
346          * for link layer header.  It has to be a multiple of 8.
347          */
348         { "_wroff_xtra", MOD_PROTO_SCTP,
349             mod_set_aligned, mod_get_uint32,
350             {0, 256, 32}, {32} },

352         { "extra_priv_ports", MOD_PROTO_SCTP,
353             mod_set_extra_privports, mod_get_extra_privports,
354             {1, ULP_MAX_PORT, 0}, {0} },

356         { "_listener_limit_conf", MOD_PROTO_SCTP,
357             NULL, sctp_listener_conf_get, {0}, {0} },

359         { "_listener_limit_conf_add", MOD_PROTO_SCTP,
360             sctp_listener_conf_add, NULL, {0}, {0} },

362         { "_listener_limit_conf_del", MOD_PROTO_SCTP,
363             sctp_listener_conf_del, NULL, {0}, {0} },

365         { "?", MOD_PROTO_SCTP, NULL, mod_get_allprop, {0}, {0} },

367         { NULL, 0, NULL, NULL, {0}, {0} }
368 };
```

*unchanged_portion_omitted_*

```
**********************************************************
  129719 Mon Jul 29 18:04:21 2013
new/usr/src/uts/common/inet/tcp/tcp.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2011, Joyent Inc. All rights reserved.
 * Copyright (c) 2011 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */
/* Copyright (c) 1990 Mentat Inc. */

#include <sys/types.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strsubr.h>
#include <sys/stropts.h>
#include <sys/strlog.h>
#define _SUN_TPI_VERSION 2
#include <sys/tihdr.h>
#include <sys/timod.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/suntpi.h>
#include <sys/xti_inet.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/sdt.h>
#include <sys/vtrace.h>
#include <sys/kmem.h>
#include <sys/ethernet.h>
#include <sys/cpuvar.h>
#include <sys/dlpi.h>
#include <sys/pattr.h>
#include <sys/policy.h>
#include <sys/priv.h>
#include <sys/zone.h>
#include <sys/sunldi.h>

#include <sys/errno.h>
#include <sys/signal.h>
```



```
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/sockio.h>
#include <sys/isa_defs.h>
#include <sys/md5.h>
#include <sys/random.h>
#include <sys/uio.h>
#include <sys/systm.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netinet/ip6.h>
#include <netinet/icmp6.h>
#include <net/if.h>
#include <net/route.h>
#include <inet/ipsec_impl.h>

#include <inet/common.h>
#include <inet/ip.h>
#include <inet/ip_impl.h>
#include <inet/ip6.h>
#include <inet/ip_ndp.h>
#include <inet/proto_set.h>
#include <inet/mib2.h>
#include <inet/optcom.h>
#include <inet/snmpcom.h>
#include <inet/kstatcom.h>
#include <inet/tcp.h>
#include <inet/tcp_impl.h>
#include <inet/tcp_cluster.h>
#include <inet/udp_impl.h>
#include <net/pfkeyv2.h>
#include <inet/ipdrop.h>

#include <inet/ipclassifier.h>
#include <inet/ip_ire.h>
#include <inet/ip_ftable.h>
#include <inet/ip_if.h>
#include <inet/ipp_common.h>
#include <inet/ip_rts.h>
#include <inet/ip_netinfo.h>
#include <sys/squeue_impl.h>
#include <sys/squeue.h>
#include <sys/tsol/label.h>
#include <sys/tsol/tnet.h>
#include <rpc/pmap_prot.h>
#include <sys/callo.h>

/*
 * TCP Notes: aka FireEngine Phase I (PSARC 2002/433)
 *
 * (Read the detailed design doc in PSARC case directory)
 *
 * The entire tcp state is contained in tcp_t and conn_t structure
 * which are allocated in tandem using ipcl_conn_create() and passing
 * IPCL_TCPCONN as a flag. We use 'conn_ref' and 'conn_lock' to protect
 * the references on the tcp_t. The tcp_t structure is never compressed
 * and packets always land on the correct TCP perimeter from the time
 * eager is created till the time tcp_t dies (as such the old mentat
 * TCP global queue is not used for detached state and no IPSEC checking
 * is required). The global queue is still allocated to send out resets
 * for connection which have no listeners and IP directly calls
 * tcp_xmit_listeners_reset() which does any policy check.
 *
 * Protection and Synchronisation mechanism:
 *
 * The tcp data structure does not use any kind of lock for protecting
```

```
125  * its state but instead uses 'squeues' for mutual exclusion from various
126  * read and write side threads. To access a tcp member, the thread should
127  * always be behind squeue (via squeue_enter with flags as SQ_FILL, SQ_PROCESS,
128  * or SQ_NODRAIN). Since the squeues allow a direct function call, caller
129  * can pass any tcp function having prototype of edesc_t as argument
130  * (different from traditional STREAMs model where packets come in only
131  * designated entry points). The list of functions that can be directly
132  * called via squeue are listed before the usual function prototype.
133  *
134  * Referencing:
135  *
136  * TCP is MT-Hot and we use a reference based scheme to make sure that the
137  * tcp structure doesn't disappear when its needed. When the application
138  * creates an outgoing connection or accepts an incoming connection, we
139  * start out with 2 references on 'conn_ref'. One for TCP and one for IP.
140  * The IP reference is just a symbolic reference since ip_tcpclose()
141  * looks at tcp structure after tcp_close_output() returns which could
142  * have dropped the last TCP reference. So as long as the connection is
143  * in attached state i.e. !TCP_IS_DETACHED, we have 2 references on the
144  * conn_t. The classifier puts its own reference when the connection is
145  * inserted in listen or connected hash. Anytime a thread needs to enter
146  * the tcp connection perimeter, it retrieves the conn/tcp from q->ptr
147  * on write side or by doing a classify on read side and then puts a
148  * reference on the conn before doing squeue_enter/tryenter/fill. For
149  * read side, the classifier itself puts the reference under fanout lock
150  * to make sure that tcp can't disappear before it gets processed. The
151  * squeue will drop this reference automatically so the called function
152  * doesn't have to do a DEC_REF.
153  *
154  * Opening a new connection:
155  *
156  * The outgoing connection open is pretty simple. tcp_open() does the
157  * work in creating the conn/tcp structure and initializing it. The
158  * squeue assignment is done based on the CPU the application
159  * is running on. So for outbound connections, processing is always done
160  * on application CPU which might be different from the incoming CPU
161  * being interrupted by the NIC. An optimal way would be to figure out
162  * the NIC <-> CPU binding at listen time, and assign the outgoing
163  * connection to the squeue attached to the CPU that will be interrupted
164  * for incoming packets (we know the NIC based on the bind IP address).
165  * This might seem like a problem if more data is going out but the
166  * fact is that in most cases the transmit is ACK driven transmit where
167  * the outgoing data normally sits on TCP's xmit queue waiting to be
168  * transmitted.
169  *
170  * Accepting a connection:
171  *
172  * This is a more interesting case because of various races involved in
173  * establishing a eager in its own perimeter. Read the meta comment on
174  * top of tcp_input_listener(). But briefly, the squeue is picked by
175  * ip_fanout based on the ring or the sender (if loopback).
176  *
177  * Closing a connection:
178  *
179  * The close is fairly straight forward. tcp_close() calls tcp_close_output()
180  * via squeue to do the close and mark the tcp as detached if the connection
181  * was in state TCPS_ESTABLISHED or greater. In the later case, TCP keep its
182  * reference but tcp_close() drop IP's reference always. So if tcp was
183  * not killed, it is sitting in time_wait list with 2 reference - 1 for TCP
184  * and 1 because it is in classifier's connected hash. This is the condition
185  * we use to determine that its OK to clean up the tcp outside of squeue
186  * when time wait expires (check the ref under fanout and conn_lock and
187  * if it is 2, remove it from fanout hash and kill it).
188  *
189  * Although close just drops the necessary references and marks the
190  * tcp_detached state, tcp_close needs to know the tcp_detached has been
```



```
191  * set (under squeue) before letting the STREAM go away (because a
192  * inbound packet might attempt to go up the STREAM while the close
193  * has happened and tcp_detached is not set). So a special lock and
194  * flag is used along with a condition variable (tcp_closelock, tcp_closed,
195  * and tcp_closecv) to signal tcp_close that tcp_close_out() has marked
196  * tcp_detached.
197  *
198  * Special provisions and fast paths:
199  *
200  * We make special provisions for sockfs by marking tcp_issocket
201  * whenever we have only sockfs on top of TCP. This allows us to skip
202  * putting the tcp in acceptor hash since a sockfs listener can never
203  * become acceptor and also avoid allocating a tcp_t for acceptor STREAM
204  * since eager has already been allocated and the accept now happens
205  * on acceptor STREAM. There is a big blob of comment on top of
206  * tcp_input_listener explaining the new accept. When socket is POP'd,
207  * sockfs sends us an ioctl to mark the fact and we go back to old
208  * behaviour. Once tcp_issocket is unset, its never set for the
209  * life of that connection.
210  *
211  * IPsec notes :
212  *
213  * Since a packet is always executed on the correct TCP perimeter
214  * all IPsec processing is defered to IP including checking new
215  * connections and setting IPSEC policies for new connection. The
216  * only exception is tcp_xmit_listeners_reset() which is called
217  * directly from IP and needs to policy check to see if TH_RST
218  * can be sent out.
219  */

221 /*
222  * Values for squeue switch:
223  * 1: SQ_NODRAIN
224  * 2: SQ_PROCESS
225  * 3: SQ_FILL
226  */
227 int tcp_squeue_wput = 2;        /* /etc/systems */
228 int tcp_squeue_flag;

230 /*
231  * To prevent memory hog, limit the number of entries in tcp_free_list
232  * to 1% of available memory / number of cpus
233  */
234 uint_t tcp_free_list_max_cnt = 0;

235 #define TCP_XMIT_LOWATER        4096
236 #define TCP_XMIT_HIWATER        49152
237 #define TCP_RECV_LOWATER        2048
238 #define TCP_RECV_HIWATER        128000

236 #define TIDUSZ  4096    /* transport interface data unit size */

238 /*
239  * Size of acceptor hash list.  It has to be a power of 2 for hashing.
240  */
241 #define TCP_ACCEPTOR_FANOUT_SIZE                512

243 #ifdef  _ILP32
244 #define TCP_ACCEPTOR_HASH(accid)                                        \
245                 (((uint_t)(accid) >> 8) & (TCP_ACCEPTOR_FANOUT_SIZE - 1))
246 #else
247 #define TCP_ACCEPTOR_HASH(accid)                                        \
248                 ((uint_t)(accid) & (TCP_ACCEPTOR_FANOUT_SIZE - 1))
249 #endif  /* _ILP32 */

251 /*
```

```
 252  * Minimum number of connections which can be created per listener.  Used
 253  * when the listener connection count is in effect.
 254  */
 255 static uint32_t tcp_min_conn_listener = 2;

 257 uint32_t tcp_early_abort = 30;

 259 /* TCP Timer control structure */
 260 typedef struct tcpt_s {
 261         pfv_t   tcpt_pfv;       /* The routine we are to call */
 262         tcp_t   *tcpt_tcp;      /* The parameter we are to pass in */
 263 } tcpt_t;
______unchanged_portion_omitted_

2590 conn_t *
2591 tcp_create_common(cred_t *credp, boolean_t isv6, boolean_t issocket,
2592     int *errorp)
2593 {
2594         tcp_t           *tcp = NULL;
2595         conn_t          *connp;
2596         zoneid_t        zoneid;
2597         tcp_stack_t     *tcps;
2598         squeue_t        *sqp;

2600         ASSERT(errorp != NULL);
2601         /*
2602          * Find the proper zoneid and netstack.
2603          */
2604         /*
2605          * Special case for install: miniroot needs to be able to
2606          * access files via NFS as though it were always in the
2607          * global zone.
2608          */
2609         if (credp == kcred && nfs_global_client_only != 0) {
2610                 zoneid = GLOBAL_ZONEID;
2611                 tcps = netstack_find_by_stackid(GLOBAL_NETSTACKID)->
2612                     netstack_tcp;
2613                 ASSERT(tcps != NULL);
2614         } else {
2615                 netstack_t *ns;
2616                 int err;

2618                 if ((err = secpolicy_basic_net_access(credp)) != 0) {
2619                         *errorp = err;
2620                         return (NULL);
2621                 }

2623                 ns = netstack_find_by_cred(credp);
2624                 ASSERT(ns != NULL);
2625                 tcps = ns->netstack_tcp;
2626                 ASSERT(tcps != NULL);

2628                 /*
2629                  * For exclusive stacks we set the zoneid to zero
2630                  * to make TCP operate as if in the global zone.
2631                  */
2632                 if (tcps->tcps_netstack->netstack_stackid !=
2633                     GLOBAL_NETSTACKID)
2634                         zoneid = GLOBAL_ZONEID;
2635                 else
2636                         zoneid = crgetzoneid(credp);
2637         }

2639         sqp = IP_SQUEUE_GET((uint_t)gethrtime());
2640         connp = (conn_t *)tcp_get_conn(sqp, tcps);
2641         /*
```



```
2642          * Both tcp_get_conn and netstack_find_by_cred incremented refcnt,
2643          * so we drop it by one.
2644          */
2645         netstack_rele(tcps->tcps_netstack);
2646         if (connp == NULL) {
2647                 *errorp = ENOSR;
2648                 return (NULL);
2649         }
2650         ASSERT(connp->conn_ixa->ixa_protocol == connp->conn_proto);

2652         connp->conn_sqp = sqp;
2653         connp->conn_initial_sqp = connp->conn_sqp;
2654         connp->conn_ixa->ixa_sqp = connp->conn_sqp;
2655         tcp = connp->conn_tcp;

2657         /*
2658          * Besides asking IP to set the checksum for us, have conn_ip_output
2659          * to do the following checks when necessary:
2660          *
2661          * IXAF_VERIFY_SOURCE: drop packets when our outer source goes invalid
2662          * IXAF_VERIFY_PMTU: verify PMTU changes
2663          * IXAF_VERIFY_LSO: verify LSO capability changes
2664          */
2665         connp->conn_ixa->ixa_flags |= IXAF_SET_ULP_CKSUM | IXAF_VERIFY_SOURCE |
2666             IXAF_VERIFY_PMTU | IXAF_VERIFY_LSO;

2668         if (!tcps->tcps_dev_flow_ctl)
2669                 connp->conn_ixa->ixa_flags |= IXAF_NO_DEV_FLOW_CTL;

2671         if (isv6) {
2672                 connp->conn_ixa->ixa_src_preferences = IPV6_PREFER_SRC_DEFAULT;
2673                 connp->conn_ipversion = IPV6_VERSION;
2674                 connp->conn_family = AF_INET6;
2675                 tcp->tcp_mss = tcps->tcps_mss_def_ipv6;
2676                 connp->conn_default_ttl = tcps->tcps_ipv6_hoplimit;
2677         } else {
2678                 connp->conn_ipversion = IPV4_VERSION;
2679                 connp->conn_family = AF_INET;
2680                 tcp->tcp_mss = tcps->tcps_mss_def_ipv4;
2681                 connp->conn_default_ttl = tcps->tcps_ipv4_ttl;
2682         }
2683         connp->conn_xmit_ipp.ipp_unicast_hops = connp->conn_default_ttl;

2685         crhold(credp);
2686         connp->conn_cred = credp;
2687         connp->conn_cpid = curproc->p_pid;
2688         connp->conn_open_time = ddi_get_lbolt64();

2690         /* Cache things in the ixa without any refhold */
2691         ASSERT(!(connp->conn_ixa->ixa_free_flags & IXA_FREE_CRED));
2692         connp->conn_ixa->ixa_cred = credp;
2693         connp->conn_ixa->ixa_cpid = connp->conn_cpid;

2695         connp->conn_zoneid = zoneid;
2696         /* conn_allzones can not be set this early, hence no IPCL_ZONEID */
2697         connp->conn_ixa->ixa_zoneid = zoneid;
2698         connp->conn_mlp_type = mlptSingle;
2699         ASSERT(connp->conn_netstack == tcps->tcps_netstack);
2700         ASSERT(tcp->tcp_tcps == tcps);

2702         /*
2703          * If the caller has the process-wide flag set, then default to MAC
2704          * exempt mode.  This allows read-down to unlabeled hosts.
2705          */
2706         if (getpflags(NET_MAC_AWARE, credp) != 0)
2707                 connp->conn_mac_mode = CONN_MAC_AWARE;
```

```
2709         connp->conn_zone_is_global = (crgetzoneid(credp) == GLOBAL_ZONEID);

2711         if (issocket) {
2712                 tcp->tcp_issocket = 1;
2713         }

2715         connp->conn_rcvbuf = tcps->tcps_recv_hiwat;
2716         connp->conn_sndbuf = tcps->tcps_xmit_hiwat;
2717         if (tcps->tcps_snd_lowat_fraction != 0) {
2718                 connp->conn_sndlowat = connp->conn_sndbuf /
2719                     tcps->tcps_snd_lowat_fraction;
2720         } else {
2721                 connp->conn_sndlowat = tcps->tcps_xmit_lowat;
2722         }
2723         connp->conn_so_type = SOCK_STREAM;
2724         connp->conn_wroff = connp->conn_ht_iphc_allocated +
2725             tcps->tcps_wroff_xtra;

2727         SOCK_CONNID_INIT(tcp->tcp_connid);
2728         /* DTrace ignores this - it isn't a tcp:::state-change */
2729         tcp->tcp_state = TCPS_IDLE;
2730         tcp_init_values(tcp, NULL);
2731         return (connp);
2732 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   15246 Mon Jul 29 18:04:23 2013
new/usr/src/uts/common/inet/tcp/tcp_tunables.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2011, Joyent Inc. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */
/* Copyright (c) 1990 Mentat Inc. */

#include <inet/ip.h>
#include <inet/tcp_impl.h>
#include <sys/multidata.h>
#include <sys/sunddi.h>

/* Max size IP datagram is 64k - 1 */
#define TCP_MSS_MAX_IPV4 (IP_MAXPACKET - (sizeof (ipha_t) + sizeof (tcpha_t)))
#define TCP_MSS_MAX_IPV6 (IP_MAXPACKET - (sizeof (ip6_t) + sizeof (tcpha_t)))

/* Max of the above */
#define TCP_MSS_MAX             TCP_MSS_MAX_IPV4

#define TCP_XMIT_LOWATER        4096
#define TCP_XMIT_HIWATER        49152
#define TCP_RECV_LOWATER        2048
#define TCP_RECV_HIWATER        128000

/*
 * Set the RFC 1948 pass phrase
 */
/* ARGSUSED */
static int
tcp_set_1948phrase(netstack_t *stack,  cred_t *cr, mod_prop_info_t *pinfo,
tcp_set_1948phrase(void *cbarg,  cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pr_val, uint_t flags)
{
        tcp_stack_t     *tcps = (tcp_stack_t *)cbarg;

        if (flags & MOD_PROP_DEFAULT)
                return (ENOTSUP);
```



```
        /*
         * Basically, value contains a new pass phrase.  Pass it along!
         */
        tcp_iss_key_init((uint8_t *)pr_val, strlen(pr_val),
            stack->netstack_tcp);
        tcp_iss_key_init((uint8_t *)pr_val, strlen(pr_val), tcps);
        return (0);
}

/*
 * returns the current list of listener limit configuration.
 */
/* ARGSUSED */
static int
tcp_listener_conf_get(netstack_t *stack, mod_prop_info_t *pinfo,
tcp_listener_conf_get(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
    void *val, uint_t psize, uint_t flags)
{
        tcp_stack_t     *tcps = stack->netstack_tcp;
        tcp_stack_t     *tcps = (tcp_stack_t *)cbarg;
        tcp_listener_t  *tl;
        char            *pval = val;
        size_t          nbytes = 0, tbytes = 0;
        uint_t          size;
        int             err = 0;

        bzero(pval, psize);
        size = psize;

        if (flags & (MOD_PROP_DEFAULT|MOD_PROP_PERM|MOD_PROP_POSSIBLE))
                return (0);

        mutex_enter(&tcps->tcps_listener_conf_lock);
        for (tl = list_head(&tcps->tcps_listener_conf); tl != NULL;
            tl = list_next(&tcps->tcps_listener_conf, tl)) {
                if (psize == size)
                        nbytes = snprintf(pval, size, "%d:%d",  tl->tl_port,
                            tl->tl_ratio);
                else
                        nbytes = snprintf(pval, size, ",%d:%d",  tl->tl_port,
                            tl->tl_ratio);
                size -= nbytes;
                pval += nbytes;
                tbytes += nbytes;
                if (tbytes >= psize) {
                        /* Buffer overflow, stop copying information */
                        err = ENOBUFS;
                        break;
                }
        }

        mutex_exit(&tcps->tcps_listener_conf_lock);
        return (err);
}

/*
 * add a new listener limit configuration.
 */
/* ARGSUSED */
static int
tcp_listener_conf_add(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
tcp_listener_conf_add(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pval, uint_t flags)
{
```

```
        tcp_listener_t  *new_tl;
        tcp_listener_t  *tl;
        long            lport;
        long            ratio;
        char            *colon;
        tcp_stack_t     *tcps = stack->netstack_tcp;
        tcp_stack_t     *tcps = (tcp_stack_t *)cbarg;

        if (flags & MOD_PROP_DEFAULT)
                return (ENOTSUP);

        if (ddi_strtol(pval, &colon, 10, &lport) != 0 || lport <= 0 ||
            lport > USHRT_MAX || *colon != ':') {
                return (EINVAL);
        }
        if (ddi_strtol(colon + 1, NULL, 10, &ratio) != 0 || ratio <= 0)
                return (EINVAL);

        mutex_enter(&tcps->tcps_listener_conf_lock);
        for (tl = list_head(&tcps->tcps_listener_conf); tl != NULL;
            tl = list_next(&tcps->tcps_listener_conf, tl)) {
                /* There is an existing entry, so update its ratio value. */
                if (tl->tl_port == lport) {
                        tl->tl_ratio = ratio;
                        mutex_exit(&tcps->tcps_listener_conf_lock);
                        return (0);
                }
        }

        if ((new_tl = kmem_alloc(sizeof (tcp_listener_t), KM_NOSLEEP)) ==
            NULL) {
                mutex_exit(&tcps->tcps_listener_conf_lock);
                return (ENOMEM);
        }

        new_tl->tl_port = lport;
        new_tl->tl_ratio = ratio;
        list_insert_tail(&tcps->tcps_listener_conf, new_tl);
        mutex_exit(&tcps->tcps_listener_conf_lock);
        return (0);
}

/*
 * remove a listener limit configuration.
 */
/* ARGSUSED */
static int
tcp_listener_conf_del(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
tcp_listener_conf_del(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pval, uint_t flags)
{
        tcp_listener_t  *tl;
        long            lport;
        tcp_stack_t     *tcps = stack->netstack_tcp;
        tcp_stack_t     *tcps = (tcp_stack_t *)cbarg;

        if (flags & MOD_PROP_DEFAULT)
                return (ENOTSUP);

        if (ddi_strtol(pval, NULL, 10, &lport) != 0 || lport <= 0 ||
            lport > USHRT_MAX) {
                return (EINVAL);
        }
        mutex_enter(&tcps->tcps_listener_conf_lock);
        for (tl = list_head(&tcps->tcps_listener_conf); tl != NULL;
            tl = list_next(&tcps->tcps_listener_conf, tl)) {
```



```
                if (tl->tl_port == lport) {
                        list_remove(&tcps->tcps_listener_conf, tl);
                        mutex_exit(&tcps->tcps_listener_conf_lock);
                        kmem_free(tl, sizeof (tcp_listener_t));
                        return (0);
                }
        }
        mutex_exit(&tcps->tcps_listener_conf_lock);
        return (ESRCH);
}

static int
tcp_set_buf_prop(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
        return (mod_set_buf_prop(stack->netstack_tcp->tcps_propinfo_tbl, stack,
            cr, pinfo, ifname, pval, flags));
}

static int
tcp_get_buf_prop(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
    void *val, uint_t psize, uint_t flags)
{
        return (mod_get_buf_prop(stack->netstack_tcp->tcps_propinfo_tbl, stack,
            pinfo, ifname, val, psize, flags));
}

/*
 * Special checkers for smallest/largest anonymous port so they don't
 * ever happen to be (largest < smallest).
 */
/* ARGSUSED */
static int
tcp_smallest_anon_set(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
tcp_smallest_anon_set(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
        unsigned long new_value;
        tcp_stack_t *tcps = stack->netstack_tcp;
        tcp_stack_t *tcps = (tcp_stack_t *)cbarg;
        int err;

        if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
                return (err);
        /* mod_uint32_value() + pinfo guarantees we're in TCP port range. */
        if ((uint32_t)new_value > tcps->tcps_largest_anon_port)
                return (ERANGE);
        pinfo->prop_cur_uval = (uint32_t)new_value;
        return (0);
}

/* ARGSUSED */
static int
tcp_largest_anon_set(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
tcp_largest_anon_set(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
        unsigned long new_value;
        tcp_stack_t *tcps = stack->netstack_tcp;
        tcp_stack_t *tcps = (tcp_stack_t *)cbarg;
        int err;

        if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
                return (err);
        /* mod_uint32_value() + pinfo guarantees we're in TCP port range. */
        if ((uint32_t)new_value < tcps->tcps_smallest_anon_port)
```

```
237                 return (ERANGE);
238         pinfo->prop_cur_uval = (uint32_t)new_value;
239         return (0);
240 }

242 /*
243  * All of these are alterable, within the min/max values given, at run time.
244  *
245  * Note: All those tunables which do not start with "_" are Committed and
246  * therefore are public. See PSARC 2010/080.
247  */
248 mod_prop_info_t tcp_propinfo_tbl[] = {
249         /* tunable - 0 */
250         { "_time_wait_interval", MOD_PROTO_TCP,
251             mod_set_uint32, mod_get_uint32,
252             {1*SECONDS, 10*MINUTES, 1*MINUTES}, {1*MINUTES} },

254         { "_conn_req_max_q", MOD_PROTO_TCP,
255             mod_set_uint32, mod_get_uint32,
256             {1, UINT32_MAX, 128}, {128} },

258         { "_conn_req_max_q0", MOD_PROTO_TCP,
259             mod_set_uint32, mod_get_uint32,
260             {0, UINT32_MAX, 1024}, {1024} },

262         { "_conn_req_min", MOD_PROTO_TCP,
263             mod_set_uint32, mod_get_uint32,
264             {1, 1024, 1}, {1} },

266         { "_conn_grace_period", MOD_PROTO_TCP,
267             mod_set_uint32, mod_get_uint32,
268             {0*MS, 20*SECONDS, 0*MS}, {0*MS} },

270         { "_cwnd_max", MOD_PROTO_TCP,
271             mod_set_uint32, mod_get_uint32,
272             {128, ULP_MAX_BUF, 1024*1024}, {1024*1024} },
261             {128, (1<<30), 1024*1024}, {1024*1024} },

274         { "_debug", MOD_PROTO_TCP,
275             mod_set_uint32, mod_get_uint32,
276             {0, 10, 0}, {0} },

278         { "smallest_nonpriv_port", MOD_PROTO_TCP,
279             mod_set_uint32, mod_get_uint32,
280             {1024, (32*1024), 1024}, {1024} },

282         { "_ip_abort_cinterval", MOD_PROTO_TCP,
283             mod_set_uint32, mod_get_uint32,
284             {1*SECONDS, UINT32_MAX, 3*MINUTES}, {3*MINUTES} },

286         { "_ip_abort_linterval", MOD_PROTO_TCP,
287             mod_set_uint32, mod_get_uint32,
288             {1*SECONDS, UINT32_MAX, 3*MINUTES}, {3*MINUTES} },

290         /* tunable - 10 */
291         { "_ip_abort_interval", MOD_PROTO_TCP,
292             mod_set_uint32, mod_get_uint32,
293             {500*MS, UINT32_MAX, 5*MINUTES}, {5*MINUTES} },

295         { "_ip_notify_cinterval", MOD_PROTO_TCP,
296             mod_set_uint32, mod_get_uint32,
297             {1*SECONDS, UINT32_MAX, 10*SECONDS},
298             {10*SECONDS} },

300         { "_ip_notify_interval", MOD_PROTO_TCP,
301             mod_set_uint32, mod_get_uint32,
```



```
302             {500*MS, UINT32_MAX, 10*SECONDS}, {10*SECONDS} },

304         { "_ipv4_ttl", MOD_PROTO_TCP,
305             mod_set_uint32, mod_get_uint32,
306             {1, 255, 64}, {64} },

308         { "_keepalive_interval", MOD_PROTO_TCP,
309             mod_set_uint32, mod_get_uint32,
310             {10*SECONDS, 10*DAYS, 2*HOURS}, {2*HOURS} },

312         { "_maxpsz_multiplier", MOD_PROTO_TCP,
313             mod_set_uint32, mod_get_uint32,
314             {0, 100, 10}, {10} },

316         { "_mss_def_ipv4", MOD_PROTO_TCP,
317             mod_set_uint32, mod_get_uint32,
318             {1, TCP_MSS_MAX_IPV4, 536}, {536} },

320         { "_mss_max_ipv4", MOD_PROTO_TCP,
321             mod_set_uint32, mod_get_uint32,
322             {1, TCP_MSS_MAX_IPV4, TCP_MSS_MAX_IPV4},
323             {TCP_MSS_MAX_IPV4} },

325         { "_mss_min", MOD_PROTO_TCP,
326             mod_set_uint32, mod_get_uint32,
327             {1, TCP_MSS_MAX, 108}, {108} },

329         { "_naglim_def", MOD_PROTO_TCP,
330             mod_set_uint32, mod_get_uint32,
331             {1, (64*1024)-1, (4*1024)-1}, {(4*1024)-1} },

333         /* tunable - 20 */
334         { "_rexmit_interval_initial", MOD_PROTO_TCP,
335             mod_set_uint32, mod_get_uint32,
336             {1*MS, 20*SECONDS, 1*SECONDS}, {1*SECONDS} },

338         { "_rexmit_interval_max", MOD_PROTO_TCP,
339             mod_set_uint32, mod_get_uint32,
340             {1*MS, 2*HOURS, 60*SECONDS}, {60*SECONDS} },

342         { "_rexmit_interval_min", MOD_PROTO_TCP,
343             mod_set_uint32, mod_get_uint32,
344             {1*MS, 2*HOURS, 400*MS}, {400*MS} },

346         { "_deferred_ack_interval", MOD_PROTO_TCP,
347             mod_set_uint32, mod_get_uint32,
348             {1*MS, 1*MINUTES, 100*MS}, {100*MS} },

350         { "_snd_lowat_fraction", MOD_PROTO_TCP,
351             mod_set_uint32, mod_get_uint32,
352             {0, 16, 10}, {10} },
341             {0, 16, 0}, {0} },

354         { "_dupack_fast_retransmit", MOD_PROTO_TCP,
355             mod_set_uint32, mod_get_uint32,
356             {1, 10000, 3}, {3} },

358         { "_ignore_path_mtu", MOD_PROTO_TCP,
359             mod_set_boolean, mod_get_boolean,
360             {B_FALSE}, {B_FALSE} },

362         { "smallest_anon_port", MOD_PROTO_TCP,
363             tcp_smallest_anon_set, mod_get_uint32,
364             {1024, ULP_MAX_PORT, 32*1024}, {32*1024} },

366         { "largest_anon_port", MOD_PROTO_TCP,
```

```
367             tcp_largest_anon_set, mod_get_uint32,
368             {1024, ULP_MAX_PORT, ULP_MAX_PORT},
369             {ULP_MAX_PORT} },

371         { "send_buf", MOD_PROTO_TCP,
372             tcp_set_buf_prop, tcp_get_buf_prop,
373             {TCP_XMIT_LOWATER, ULP_MAX_BUF, TCP_XMIT_HIWATER},
360         { "send_maxbuf", MOD_PROTO_TCP,
361             mod_set_uint32, mod_get_uint32,
362             {TCP_XMIT_LOWATER, (1<<30), TCP_XMIT_HIWATER},
374             {TCP_XMIT_HIWATER} },

376         /* tunable - 30 */
377         { "_xmit_lowat", MOD_PROTO_TCP,
378             mod_set_uint32, mod_get_uint32,
379             {TCP_XMIT_LOWATER, ULP_MAX_BUF, TCP_XMIT_LOWATER},
368             {TCP_XMIT_LOWATER, (1<<30), TCP_XMIT_LOWATER},
380             {TCP_XMIT_LOWATER} },

382         { "recv_buf", MOD_PROTO_TCP,
383             tcp_set_buf_prop, tcp_get_buf_prop,
384             {TCP_RECV_LOWATER, ULP_MAX_BUF, TCP_RECV_HIWATER},
371         { "recv_maxbuf", MOD_PROTO_TCP,
372             mod_set_uint32, mod_get_uint32,
373             {TCP_RECV_LOWATER, (1<<30), TCP_RECV_HIWATER},
385             {TCP_RECV_HIWATER} },

387         { "_recv_hiwat_minmss", MOD_PROTO_TCP,
388             mod_set_uint32, mod_get_uint32,
389             {1, 65536, 4}, {4} },

391         { "_fin_wait_2_flush_interval", MOD_PROTO_TCP,
392             mod_set_uint32, mod_get_uint32,
393             {1*SECONDS, 2*HOURS, 60*SECONDS},
394             {60*SECONDS} },

396         { "max_buf", MOD_PROTO_TCP,
385         { "_max_buf", MOD_PROTO_TCP,
397             mod_set_uint32, mod_get_uint32,
398             {8192, ULP_MAX_BUF, 1024*1024}, {1024*1024} },
387             {8192, (1<<30), 1024*1024}, {1024*1024} },

400         /*
401          * Question:  What default value should I set for tcp_strong_iss?
402          */
403         { "_strong_iss", MOD_PROTO_TCP,
404             mod_set_uint32, mod_get_uint32,
405             {0, 2, 1}, {1} },

407         { "_rtt_updates", MOD_PROTO_TCP,
408             mod_set_uint32, mod_get_uint32,
409             {0, 65536, 20}, {20} },

411         { "_wscale_always", MOD_PROTO_TCP,
412             mod_set_boolean, mod_get_boolean,
413             {B_TRUE}, {B_TRUE} },

415         { "_tstamp_always", MOD_PROTO_TCP,
416             mod_set_boolean, mod_get_boolean,
417             {B_FALSE}, {B_FALSE} },

419         { "_tstamp_if_wscale", MOD_PROTO_TCP,
420             mod_set_boolean, mod_get_boolean,
421             {B_TRUE}, {B_TRUE} },

423         /* tunable - 40 */
```



```
424         { "_rexmit_interval_extra", MOD_PROTO_TCP,
425             mod_set_uint32, mod_get_uint32,
426             {0*MS, 2*HOURS, 0*MS}, {0*MS} },

428         { "_deferred_acks_max", MOD_PROTO_TCP,
429             mod_set_uint32, mod_get_uint32,
430             {0, 16, 2}, {2} },

432         { "_slow_start_after_idle", MOD_PROTO_TCP,
433             mod_set_uint32, mod_get_uint32,
434             {0, 16384, 0}, {0} },

436         { "_slow_start_initial", MOD_PROTO_TCP,
437             mod_set_uint32, mod_get_uint32,
438             {0, 16, 0}, {0} },

440         { "sack", MOD_PROTO_TCP,
441             mod_set_uint32, mod_get_uint32,
442             {0, 2, 2}, {2} },

444         { "_ipv6_hoplimit", MOD_PROTO_TCP,
445             mod_set_uint32, mod_get_uint32,
446             {0, IPV6_MAX_HOPS, IPV6_DEFAULT_HOPS},
447             {IPV6_DEFAULT_HOPS} },

449         { "_mss_def_ipv6", MOD_PROTO_TCP,
450             mod_set_uint32, mod_get_uint32,
451             {1, TCP_MSS_MAX_IPV6, 1220}, {1220} },

453         { "_mss_max_ipv6", MOD_PROTO_TCP,
454             mod_set_uint32, mod_get_uint32,
455             {1, TCP_MSS_MAX_IPV6, TCP_MSS_MAX_IPV6},
456             {TCP_MSS_MAX_IPV6} },

458         { "_rev_src_routes", MOD_PROTO_TCP,
459             mod_set_boolean, mod_get_boolean,
460             {B_FALSE}, {B_FALSE} },

462         { "_local_dack_interval", MOD_PROTO_TCP,
463             mod_set_uint32, mod_get_uint32,
464             {10*MS, 500*MS, 50*MS}, {50*MS} },

466         /* tunable - 50 */
467         { "_local_dacks_max", MOD_PROTO_TCP,
468             mod_set_uint32, mod_get_uint32,
469             {0, 16, 8}, {8} },

471         { "ecn", MOD_PROTO_TCP,
472             mod_set_uint32, mod_get_uint32,
473             {0, 2, 1}, {1} },

475         { "_rst_sent_rate_enabled", MOD_PROTO_TCP,
476             mod_set_boolean, mod_get_boolean,
477             {B_TRUE}, {B_TRUE} },

479         { "_rst_sent_rate", MOD_PROTO_TCP,
480             mod_set_uint32, mod_get_uint32,
481             {0, UINT32_MAX, 40}, {40} },

483         { "_push_timer_interval", MOD_PROTO_TCP,
484             mod_set_uint32, mod_get_uint32,
485             {0, 100*MS, 50*MS}, {50*MS} },

487         { "_use_smss_as_mss_opt", MOD_PROTO_TCP,
488             mod_set_boolean, mod_get_boolean,
489             {B_FALSE}, {B_FALSE} },
```

```
491         { "_keepalive_abort_interval", MOD_PROTO_TCP,
492             mod_set_uint32, mod_get_uint32,
493             {0, UINT32_MAX, 8*MINUTES}, {8*MINUTES} },

495         /*
496          * tcp_wroff_xtra is the extra space in front of TCP/IP header for link
497          * layer header.  It has to be a multiple of 8.
498          */
499         { "_wroff_xtra", MOD_PROTO_TCP,
500             mod_set_aligned, mod_get_uint32,
501             {0, 256, 32}, {32} },

503         { "_dev_flow_ctl", MOD_PROTO_TCP,
504             mod_set_boolean, mod_get_boolean,
505             {B_FALSE}, {B_FALSE} },

507         { "_reass_timeout", MOD_PROTO_TCP,
508             mod_set_uint32, mod_get_uint32,
509             {0, UINT32_MAX, 100*SECONDS}, {100*SECONDS} },

511         /* tunable - 60 */
512         { "extra_priv_ports", MOD_PROTO_TCP,
513             mod_set_extra_privports, mod_get_extra_privports,
514             {1, ULP_MAX_PORT, 0}, {0} },

516         { "_1948_phrase", MOD_PROTO_TCP,
517             tcp_set_1948phrase, NULL, {0}, {0} },

519         { "_listener_limit_conf", MOD_PROTO_TCP,
520             NULL, tcp_listener_conf_get, {0}, {0} },

522         { "_listener_limit_conf_add", MOD_PROTO_TCP,
523             tcp_listener_conf_add, NULL, {0}, {0} },

525         { "_listener_limit_conf_del", MOD_PROTO_TCP,
526             tcp_listener_conf_del, NULL, {0}, {0} },

528         { "_iss_incr", MOD_PROTO_TCP,
529             mod_set_uint32, mod_get_uint32,
530             {1, ISS_INCR, ISS_INCR},
531             {ISS_INCR} },

533         { "?", MOD_PROTO_TCP, NULL, mod_get_allprop, {0}, {0} },

535         { NULL, 0, NULL, NULL, {0}, {0} }
536 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   30072 Mon Jul 29 18:04:24 2013
new/usr/src/uts/common/inet/tcp_impl.h
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011, Joyent Inc. All rights reserved.
 * Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _INET_TCP_IMPL_H
#define _INET_TCP_IMPL_H

/*
 * TCP implementation private declarations.  These interfaces are
 * used to build the IP module and are not meant to be accessed
 * by any modules except IP itself.  They are undocumented and are
 * subject to change without notice.
 */

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef _KERNEL

#include <sys/cpuvar.h>
#include <sys/clock_impl.h>     /* For LBOLT_FASTPATH{,64} */
#include <inet/optcom.h>
#include <inet/tcp.h>
#include <inet/tunables.h>

#define TCP_MOD_ID      5105

extern struct qinit     tcp_sock_winit;
extern struct qinit     tcp_winit;

extern sock_downcalls_t sock_tcp_downcalls;

/*
 * Note that by default, the _snd_lowat_fraction tunable controls the value of
```



```
 * the transmit low water mark.  TCP_XMIT_LOWATER (and thus the _xmit_lowat
 * property) is only used if the administrator has disabled _snd_lowat_fraction
 * by setting it to 0.
 */
#define TCP_XMIT_LOWATER        4096
#define TCP_XMIT_HIWATER        49152
#define TCP_RECV_LOWATER        2048
#define TCP_RECV_HIWATER        128000

/*
 * Bind hash list size and has function.  It has to be a power of 2 for
 * hashing.
 */
#define TCP_BIND_FANOUT_SIZE    1024
#define TCP_BIND_HASH(lport) (ntohs(lport) & (TCP_BIND_FANOUT_SIZE - 1))

/*
 * This implementation follows the 4.3BSD interpretation of the urgent
 * pointer and not RFC 1122. Switching to RFC 1122 behavior would cause
 * incompatible changes in protocols like telnet and rlogin.
 */
#define TCP_OLD_URP_INTERPRETATION      1

/* TCP option length */
#define TCPOPT_NOP_LEN          1
#define TCPOPT_MAXSEG_LEN       4
#define TCPOPT_WS_LEN           3
#define TCPOPT_REAL_WS_LEN      (TCPOPT_WS_LEN+1)
#define TCPOPT_TSTAMP_LEN       10
#define TCPOPT_REAL_TS_LEN      (TCPOPT_TSTAMP_LEN+2)
#define TCPOPT_SACK_OK_LEN      2
#define TCPOPT_REAL_SACK_OK_LEN (TCPOPT_SACK_OK_LEN+2)
#define TCPOPT_REAL_SACK_LEN    4
#define TCPOPT_MAX_SACK_LEN     36
#define TCPOPT_HEADER_LEN       2

/* Round up the value to the nearest mss. */
#define MSS_ROUNDUP(value, mss)         ((((value) - 1) / (mss) + 1) * (mss))

/*
 * Was this tcp created via socket() interface?
 */
#define TCP_IS_SOCKET(tcp)      ((tcp)->tcp_issocket)

/*
 * Is this tcp not attached to any upper client?
 */
#define TCP_IS_DETACHED(tcp)    ((tcp)->tcp_detached)

/* TCP timers related data strucutres.  Refer to tcp_timers.c. */
typedef struct tcp_timer_s {
        conn_t  *connp;
        void    (*tcpt_proc)(void *);
        callout_id_t   tcpt_tid;
} tcp_timer_t;
_____unchanged_portion_omitted_
```

```
**********************************************************
   13690 Mon Jul 29 18:04:25 2013
new/usr/src/uts/common/inet/tunables.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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 by Delphix. All rights reserved.
 */

#include <inet/tunables.h>
#include <sys/md5.h>
#include <inet/common.h>
#include <inet/ip.h>
#include <inet/ip6.h>
#include <netinet/icmp6.h>
#include <inet/ip_stack.h>
#include <inet/rawip_impl.h>
#include <inet/tcp_stack.h>
#include <inet/tcp_impl.h>
#include <inet/udp_impl.h>
#include <inet/sctp/sctp_stack.h>
#include <inet/sctp/sctp_impl.h>
#include <inet/tunables.h>

mod_prop_info_t *
mod_prop_lookup(mod_prop_info_t ptbl[], const char *prop_name, uint_t proto)
{
        mod_prop_info_t *pinfo;

        /*
         * Walk the ptbl array looking for a property that has the requested
         * name and protocol number.  Note that we assume that all protocol
         * tables are terminated by an entry with a NULL property name.
         */
        for (pinfo = ptbl; pinfo->mpi_name != NULL; pinfo++) {
                if (strcmp(pinfo->mpi_name, prop_name) == 0 &&
                    pinfo->mpi_proto == proto)
                        return (pinfo);
        }
        return (NULL);
}
```



```
static int
prop_perm2const(mod_prop_info_t *pinfo)
{
        if (pinfo->mpi_setf == NULL)
                return (MOD_PROP_PERM_READ);
        if (pinfo->mpi_getf == NULL)
                return (MOD_PROP_PERM_WRITE);
        return (MOD_PROP_PERM_RW);
}

/*
 * Modifies the value of the property to default value or to the ‘pval’
 * specified by the user.
 */
/* ARGSUSED */
int
mod_set_boolean(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
mod_set_boolean(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void* pval, uint_t flags)
{
        char            *end;
        unsigned long   new_value;

        if (flags & MOD_PROP_DEFAULT) {
                pinfo->prop_cur_bval = pinfo->prop_def_bval;
                return (0);
        }

        if (ddi_strtoul(pval, &end, 10, &new_value) != 0 || *end != ’\0’)
                return (EINVAL);
        if (new_value != B_TRUE && new_value != B_FALSE)
                return (EINVAL);
        pinfo->prop_cur_bval = new_value;
        return (0);
}

/*
 * Retrieves property permission, default value, current value or possible
 * values for those properties whose value type is boolean_t.
 */
/* ARGSUSED */
int
mod_get_boolean(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
mod_get_boolean(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
    void *pval, uint_t psize, uint_t flags)
{
        boolean_t       get_def = (flags & MOD_PROP_DEFAULT);
        boolean_t       get_perm = (flags & MOD_PROP_PERM);
        boolean_t       get_range = (flags & MOD_PROP_POSSIBLE);
        size_t          nbytes;

        bzero(pval, psize);
        if (get_perm)
                nbytes = snprintf(pval, psize, "%u", prop_perm2const(pinfo));
        else if (get_range)
                nbytes = snprintf(pval, psize, "%u,%u", B_FALSE, B_TRUE);
        else if (get_def)
                nbytes = snprintf(pval, psize, "%u", pinfo->prop_def_bval);
        else
                nbytes = snprintf(pval, psize, "%u", pinfo->prop_cur_bval);
        if (nbytes >= psize)
                return (ENOBUFS);
        return (0);
}
```

*_____unchanged_portion_omitted_*

```
144 /*
145  * Modifies the value of the property to default value or to the ‘pval’
146  * specified by the user.
147  */
148 /* ARGSUSED */
149 int
150 mod_set_uint32(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
151 mod_set_uint32(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
152     const char *ifname, const void *pval, uint_t flags)
153 {
154         unsigned long   new_value;
155         int             err;

156         if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
157                 return (err);
158         pinfo->prop_cur_uval = (uint32_t)new_value;
159         return (0);
160 }

162 /*
163  * Rounds up the value to make it multiple of 8.
164  */
165 /* ARGSUSED */
166 int
167 mod_set_aligned(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
168 mod_set_aligned(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
169     const char *ifname, const void* pval, uint_t flags)
170 {
171         int     err;

172         if ((err = mod_set_uint32(stack, cr, pinfo, ifname, pval, flags)) != 0)
173         if ((err = mod_set_uint32(cbarg, cr, pinfo, ifname, pval, flags)) != 0)
174                 return (err);

175         /* if required, align the value to multiple of 8 */
176         if (pinfo->prop_cur_uval & 0x7) {
177                 pinfo->prop_cur_uval &= ~0x7;
178                 pinfo->prop_cur_uval += 0x8;
179         }

180         return (0);
181 }

183 /*
184  * Retrieves property permission, default value, current value or possible
185  * values for those properties whose value type is uint32_t.
186  */
187 /* ARGSUSED */
188 int
189 mod_get_uint32(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
190 mod_get_uint32(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
191     void *pval, uint_t psize, uint_t flags)
192 {
193         boolean_t       get_def = (flags & MOD_PROP_DEFAULT);
194         boolean_t       get_perm = (flags & MOD_PROP_PERM);
195         boolean_t       get_range = (flags & MOD_PROP_POSSIBLE);
196         size_t          nbytes;

197         bzero(pval, psize);
198         if (get_perm)
199                 nbytes = snprintf(pval, psize, "%u", prop_perm2const(pinfo));
200         else if (get_range)
201                 nbytes = snprintf(pval, psize, "%u-%u",
202                     pinfo->prop_min_uval, pinfo->prop_max_uval);
203         else if (get_def)
```



```
204                 nbytes = snprintf(pval, psize, "%u", pinfo->prop_def_uval);
205         else
206                 nbytes = snprintf(pval, psize, "%u", pinfo->prop_cur_uval);
207         if (nbytes >= psize)
208                 return (ENOBUFS);
209         return (0);
210 }

212 /*
213  * The range of the buffer size properties has a static lower bound configured
214  * in the property info structure of the property itself, and a dynamic upper
215  * bound.  The upper bound is the current value of the "max_buf" property
216  * in the appropriate protocol property table.
217  */
218 static void
219 mod_get_buf_prop_range(mod_prop_info_t ptbl[], mod_prop_info_t *pinfo,
220     uint32_t *min, uint32_t *max)
221 {
222         mod_prop_info_t *maxbuf_pinfo = mod_prop_lookup(ptbl, "max_buf",
223             pinfo->mpi_proto);

224         *min = pinfo->prop_min_uval;
225         *max = maxbuf_pinfo->prop_cur_uval;
226 }

228 /*
229  * Modifies the value of the buffer size property to its default value or to
230  * the value specified by the user.  This is similar to mod_set_uint32() except
231  * that the value has a dynamically bounded range (see mod_get_buf_prop_range()
232  * for details).
233  */
234 /* ARGSUSED */
235 int
236 mod_set_buf_prop(mod_prop_info_t ptbl[], netstack_t *stack, cred_t *cr,
237     mod_prop_info_t *pinfo, const char *ifname, const void *pval, uint_t flags)
238 {
239         unsigned long   new_value;
240         char            *end;
241         uint32_t        min, max;

242         if (flags & MOD_PROP_DEFAULT) {
243                 pinfo->prop_cur_uval = pinfo->prop_def_uval;
244                 return (0);
245         }

246         if (ddi_strtoul(pval, &end, 10, &new_value) != 0 || *end != ’\0’)
247                 return (EINVAL);

248         mod_get_buf_prop_range(ptbl, pinfo, &min, &max);
249         if (new_value < min || new_value > max)
250                 return (ERANGE);

251         pinfo->prop_cur_uval = new_value;
252         return (0);
253 }

255 /*
256  * Retrieves property permissions, default value, current value, or possible
257  * values for buffer size properties.  While these properties have integer
258  * values, they have a dynamic range (see mod_get_buf_prop_range() for
259  * details).  As such, they need to be handled differently.
260  */
261 int
262 mod_get_buf_prop(mod_prop_info_t ptbl[], netstack_t *stack,
263     mod_prop_info_t *pinfo, const char *ifname, void *pval, uint_t psize,
264     uint_t flags)
```

```
271 {
272         size_t nbytes;
273         uint32_t min, max;

275         if (flags & MOD_PROP_POSSIBLE) {
276                 mod_get_buf_prop_range(ptbl, pinfo, &min, &max);
277                 nbytes = snprintf(pval, psize, "%u-%u", min, max);
278                 return (nbytes < psize ? 0 : ENOBUFS);
279         }
280         return (mod_get_uint32(stack, pinfo, ifname, pval, psize, flags));
281 }

283 /*
284  * Implements /sbin/ndd -get /dev/ip ?, for all the modules. Needed for
285  * backward compatibility with /sbin/ndd.
286  */
287 /* ARGSUSED */
288 int
289 mod_get_allprop(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
200 mod_get_allprop(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
290     void *val, uint_t psize, uint_t flags)
291 {
292         char            *pval = val;
293         mod_prop_info_t *ptbl, *prop;
205         ip_stack_t      *ipst;
206         tcp_stack_t     *tcps;
207         sctp_stack_t    *sctps;
208         udp_stack_t     *us;
209         icmp_stack_t    *is;
294         uint_t          size;
295         size_t          nbytes = 0, tbytes = 0;

297         bzero(pval, psize);
298         size = psize;

300         switch (pinfo->mpi_proto) {
301         case MOD_PROTO_IP:
302         case MOD_PROTO_IPV4:
303         case MOD_PROTO_IPV6:
304                 ptbl = stack->netstack_ip->ips_propinfo_tbl;
220                 ipst = (ip_stack_t *)cbarg;
221                 ptbl = ipst->ips_propinfo_tbl;
305                 break;
306         case MOD_PROTO_RAWIP:
307                 ptbl = stack->netstack_icmp->is_propinfo_tbl;
224                 is = (icmp_stack_t *)cbarg;
225                 ptbl = is->is_propinfo_tbl;
308                 break;
309         case MOD_PROTO_TCP:
310                 ptbl = stack->netstack_tcp->tcps_propinfo_tbl;
228                 tcps = (tcp_stack_t *)cbarg;
229                 ptbl = tcps->tcps_propinfo_tbl;
311                 break;
312         case MOD_PROTO_UDP:
313                 ptbl = stack->netstack_udp->us_propinfo_tbl;
232                 us = (udp_stack_t *)cbarg;
233                 ptbl = us->us_propinfo_tbl;
314                 break;
315         case MOD_PROTO_SCTP:
316                 ptbl = stack->netstack_sctp->sctps_propinfo_tbl;
236                 sctps = (sctp_stack_t *)cbarg;
237                 ptbl = sctps->sctps_propinfo_tbl;
317                 break;
318         default:
319                 return (EINVAL);
320         }
```



```
322         for (prop = ptbl; prop->mpi_name != NULL; prop++) {
323                 if (prop->mpi_name[0] == '\0' ||
324                     strcmp(prop->mpi_name, "?") == 0) {
325                         continue;
326                 }
327                 nbytes = snprintf(pval, size, "%s %d %d", prop->mpi_name,
328                     prop->mpi_proto, prop_perm2const(prop));
329                 size -= nbytes + 1;
330                 pval += nbytes + 1;
331                 tbytes += nbytes + 1;
332                 if (tbytes >= psize) {
333                         /* Buffer overflow, stop copying information */
334                         return (ENOBUFS);
335                 }
336         }
337         return (0);
338 }

340 /*
341  * Hold a lock while changing *_epriv_ports to prevent multiple
342  * threads from changing it at the same time.
343  */
344 /* ARGSUSED */
345 int
346 mod_set_extra_privports(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
267 mod_set_extra_privports(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
347     const char *ifname, const void* val, uint_t flags)
348 {
349         uint_t          proto = pinfo->mpi_proto;
350         tcp_stack_t     *tcps;
351         sctp_stack_t    *sctps;
352         udp_stack_t     *us;
353         unsigned long   new_value;
354         char            *end;
355         kmutex_t        *lock;
356         uint_t          i, nports;
357         in_port_t       *ports;
358         boolean_t       def = (flags & MOD_PROP_DEFAULT);
359         const char      *pval = val;

361         if (!def) {
362                 if (ddi_strtoul(pval, &end, 10, &new_value) != 0 ||
363                     *end != '\0') {
364                         return (EINVAL);
365                 }

367                 if (new_value < pinfo->prop_min_uval ||
368                     new_value > pinfo->prop_max_uval) {
369                         return (ERANGE);
370                 }
371         }

373         switch (proto) {
374         case MOD_PROTO_TCP:
375                 tcps = stack->netstack_tcp;
296                 tcps = (tcp_stack_t *)cbarg;
376                 lock = &tcps->tcps_epriv_port_lock;
377                 ports = tcps->tcps_g_epriv_ports;
378                 nports = tcps->tcps_g_num_epriv_ports;
379                 break;
380         case MOD_PROTO_UDP:
381                 us = stack->netstack_udp;
302                 us = (udp_stack_t *)cbarg;
382                 lock = &us->us_epriv_port_lock;
383                 ports = us->us_epriv_ports;
```

```
384                 nports = us->us_num_epriv_ports;
385                 break;
386         case MOD_PROTO_SCTP:
387                 sctps = stack->netstack_sctp;
388                 sctps = (sctp_stack_t *)cbarg;
389                 lock = &sctps->sctps_epriv_port_lock;
390                 ports = sctps->sctps_g_epriv_ports;
391                 nports = sctps->sctps_g_num_epriv_ports;
392                 break;
393         default:
394                 return (ENOTSUP);
395         }
396
397         mutex_enter(lock);
398
399         /* if MOD_PROP_DEFAULT is set then reset the ports list to default */
400         if (def) {
401                 for (i = 0; i < nports; i++)
402                         ports[i] = 0;
403                 ports[0] = ULP_DEF_EPRIV_PORT1;
404                 ports[1] = ULP_DEF_EPRIV_PORT2;
405                 mutex_exit(lock);
406                 return (0);
407         }
408
409         /* Check if the value is already in the list */
410         for (i = 0; i < nports; i++) {
411                 if (new_value == ports[i])
412                         break;
413         }
414
415         if (flags & MOD_PROP_REMOVE) {
416                 if (i == nports) {
417                         mutex_exit(lock);
418                         return (ESRCH);
419                 }
420                 /* Clear the value */
421                 ports[i] = 0;
422         } else if (flags & MOD_PROP_APPEND) {
423                 if (i != nports) {
424                         mutex_exit(lock);
425                         return (EEXIST);
426                 }
427
428                 /* Find an empty slot */
429                 for (i = 0; i < nports; i++) {
430                         if (ports[i] == 0)
431                                 break;
432                 }
433                 if (i == nports) {
434                         mutex_exit(lock);
435                         return (EOVERFLOW);
436                 }
437                 /* Set the new value */
438                 ports[i] = (in_port_t)new_value;
439         } else {
440                 /*
441                  * If the user used 'assignment' modifier.
442                  * For eg:
443                  *      # ipadm set-prop -p extra_priv_ports=3001 tcp
444                  *
445                  * We clear all the ports and then just add 3001.
446                  */
447                 ASSERT(flags == MOD_PROP_ACTIVE);
448                 for (i = 0; i < nports; i++)
449                         ports[i] = 0;
450                 ports[0] = (in_port_t)new_value;
451         }
452
453         mutex_exit(lock);
454         return (0);
455 }
456
457 /*
458  * Note: No locks are held when inspecting *_epriv_ports
459  * but instead the code relies on:
460  * - the fact that the address of the array and its size never changes
461  * - the atomic assignment of the elements of the array
462  */
463 /* ARGSUSED */
464 int
465 mod_get_extra_privports(netstack_t *stack, mod_prop_info_t *pinfo,
466     const char *ifname, void *val, uint_t psize, uint_t flags)
385 mod_get_extra_privports(void *cbarg, mod_prop_info_t *pinfo, const char *ifname,
386     void *val, uint_t psize, uint_t flags)
467 {
468         uint_t          proto = pinfo->mpi_proto;
469         tcp_stack_t     *tcps;
470         sctp_stack_t    *sctps;
471         udp_stack_t     *us;
472         uint_t          i, nports, size;
473         in_port_t       *ports;
474         char            *pval = val;
475         size_t          nbytes = 0, tbytes = 0;
476         boolean_t       get_def = (flags & MOD_PROP_DEFAULT);
477         boolean_t       get_perm = (flags & MOD_PROP_PERM);
478         boolean_t       get_range = (flags & MOD_PROP_POSSIBLE);
479
480         bzero(pval, psize);
481         size = psize;
482
483         if (get_def) {
484                 tbytes = snprintf(pval, psize, "%u,%u", ULP_DEF_EPRIV_PORT1,
485                     ULP_DEF_EPRIV_PORT2);
486                 goto ret;
487         } else if (get_perm) {
488                 tbytes = snprintf(pval, psize, "%u", MOD_PROP_PERM_RW);
489                 goto ret;
490         }
491
492         switch (proto) {
493         case MOD_PROTO_TCP:
414                 tcps = stack->netstack_tcp;
494                 tcps = (tcp_stack_t *)cbarg;
495                 ports = tcps->tcps_g_epriv_ports;
496                 nports = tcps->tcps_g_num_epriv_ports;
497                 break;
498         case MOD_PROTO_UDP:
419                 us = stack->netstack_udp;
499                 us = (udp_stack_t *)cbarg;
500                 ports = us->us_epriv_ports;
501                 nports = us->us_num_epriv_ports;
502                 break;
503         case MOD_PROTO_SCTP:
424                 sctps = stack->netstack_sctp;
504                 sctps = (sctp_stack_t *)cbarg;
505                 ports = sctps->sctps_g_epriv_ports;
506                 nports = sctps->sctps_g_num_epriv_ports;
507                 break;
508         default:
509                 return (ENOTSUP);
```

```
511         if (get_range) {
512                 tbytes = snprintf(pval, psize, "%u-%u", pinfo->prop_min_uval,
513                     pinfo->prop_max_uval);
514                 goto ret;
515         }

517         for (i = 0; i < nports; i++) {
518                 if (ports[i] != 0) {
519                         if (psize == size)
520                                 nbytes = snprintf(pval, size, "%u", ports[i]);
521                         else
522                                 nbytes = snprintf(pval, size, ",%u", ports[i]);
523                         size -= nbytes;
524                         pval += nbytes;
525                         tbytes += nbytes;
526                         if (tbytes >= psize)
527                                 return (ENOBUFS);
528                 }
529         }
530         return (0);
531 ret:
532         if (tbytes >= psize)
533                 return (ENOBUFS);
534         return (0);
535 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6596 Mon Jul 29 18:04:27 2013
new/usr/src/uts/common/inet/tunables.h
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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 by Delphix. All rights reserved.
 */

#ifndef _INET_TUNABLES_H
#define _INET_TUNABLES_H

#include <sys/types.h>
#include <net/if.h>
#ifdef _KERNEL
#include <sys/netstack.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

#define MAXPROPNAMELEN  64

/*
 * The 'mod_ioc_prop_s' datastructure is used as an IOCTL argument for
 * SIOCSETPROP and SIOCGETPROP ioctls. This datastructure identifies the
 * protocol ('mpr_proto') property ('mpr_name'), which needs to be modified
 * or retrieved ('mpr_valsize' and 'mpr_val'). If the property applies to an
 * interface then 'mpr_ifname' contains the name of the interface.
 */
typedef struct mod_ioc_prop_s {
        uint_t          mpr_version;
        uint_t          mpr_flags;                      /* see below */
        /* name of the interface (ill) for which property will be applied */
        char            mpr_ifname[LIFNAMSIZ];
        uint_t          mpr_proto;                      /* see below */
        char            mpr_name[MAXPROPNAMELEN];       /* property name */
        uint_t          mpr_valsize;                    /* size of mpr_val */
        char            mpr_val[1];
} mod_ioc_prop_t;
```



```
#define MOD_PROP_VERSION        1

/* permission flags for properties */
#define MOD_PROP_PERM_READ      0x1
#define MOD_PROP_PERM_WRITE     0x2
#define MOD_PROP_PERM_RW        (MOD_PROP_PERM_READ|MOD_PROP_PERM_WRITE)

/* mpr_flags values */
#define MOD_PROP_ACTIVE         0x01    /* current value of the property */
#define MOD_PROP_DEFAULT        0x02    /* default value of the property */
#define MOD_PROP_POSSIBLE       0x04    /* possible values for the property */
#define MOD_PROP_PERM           0x08    /* read/write permission for property */
#define MOD_PROP_APPEND         0x10    /* append to multi-valued property */
#define MOD_PROP_REMOVE         0x20    /* remove from multi-valued property */

/* mpr_proto values */
#define MOD_PROTO_NONE          0x00
#define MOD_PROTO_IPV4          0x01    /* property is applicable to IPV4 */
#define MOD_PROTO_IPV6          0x02    /* property is applicable to IPV6 */
#define MOD_PROTO_RAWIP         0x04    /* property is applicable to ICMP */
#define MOD_PROTO_TCP           0x08    /* property is applicable to TCP */
#define MOD_PROTO_UDP           0x10    /* property is applicable to UDP */
#define MOD_PROTO_SCTP          0x20    /* property is applicable to SCTP */

/* property is applicable to both IPV[4|6] */
#define MOD_PROTO_IP            (MOD_PROTO_IPV4|MOD_PROTO_IPV6)

#ifdef  _KERNEL

typedef struct mod_prop_info_s mod_prop_info_t;

/* set/get property callback functions */
typedef int     mod_prop_setf_t(netstack_t *, cred_t *, mod_prop_info_t *,
typedef int     mod_prop_setf_t(void *, cred_t *, mod_prop_info_t *,
                    const char *, const void *, uint_t);
typedef int     mod_prop_getf_t(netstack_t *, mod_prop_info_t *, const char *,
                    void *, uint_t, uint_t);
typedef int     mod_prop_getf_t(void *, mod_prop_info_t *, const char *,
                    void *val, uint_t, uint_t);

typedef struct mod_propval_uint32_s {
        uint32_t        mod_propval_umin;
        uint32_t        mod_propval_umax;
        uint32_t        mod_propval_ucur;
} mod_propval_uint32_t;
_____unchanged_portion_omitted_

/* shortcuts to access current/default values */
#define prop_min_uval   u.mpi_uval.mod_propval_umin
#define prop_max_uval   u.mpi_uval.mod_propval_umax
#define prop_cur_uval   u.mpi_uval.mod_propval_ucur
#define prop_cur_bval   u.mpi_bval
#define prop_def_uval   u_def.mpi_def_uval
#define prop_def_bval   u_def.mpi_def_bval

#define MS              1L
#define SECONDS         (1000 * MS)
#define MINUTES         (60 * SECONDS)
#define HOURS           (60 * MINUTES)
#define DAYS            (24 * HOURS)

#define MB              (1024 * 1024)

/* Largest TCP/UDP/SCTP port number */
#define ULP_MAX_PORT    (64 * 1024 - 1)
```

```
149 /* extra privilege ports for upper layer protocols, tcp, sctp and udp */
150 #define ULP_DEF_EPRIV_PORT1     2049
151 #define ULP_DEF_EPRIV_PORT2     4045

153 #define ULP_MAX_BUF     (1<<30) /* Largest possible send/receive buffer */

155 /* generic function to set/get global module properties */
156 extern mod_prop_setf_t  mod_set_boolean, mod_set_uint32,
157                         mod_set_aligned, mod_set_extra_privports;

159 extern mod_prop_getf_t  mod_get_boolean, mod_get_uint32,
160                         mod_get_allprop, mod_get_extra_privports;

162 extern int              mod_uint32_value(const void *, mod_prop_info_t *,
163     uint_t, unsigned long *);
164 extern mod_prop_info_t  *mod_prop_lookup(mod_prop_info_t[], const char *,
165     uint_t);
166 extern int              mod_set_buf_prop(mod_prop_info_t[], netstack_t *,
167     cred_t *cr, mod_prop_info_t *, const char *, const void *, uint_t);
168 extern int              mod_get_buf_prop(mod_prop_info_t[], netstack_t *,
169     mod_prop_info_t *, const char *, void *, uint_t, uint_t);
159 extern int mod_uint32_value(const void *, mod_prop_info_t *, uint_t,
160     unsigned long *);

171 #endif  /* _KERNEL */

173 /*
174  * End-system model definitions that include the weak/strong end-system
175  * definitions in RFC 1122, Section 3.3.4.5. IP_WEAK_ES and IP_STRONG_ES
176  * conform to the corresponding  RFC 1122 definitions. The IP_SRC_PRI_ES
177  * hostmodel is similar to IP_WEAK_ES with one additional enhancement: for
178  * a packet with source S2, destination D2, the route selection algorithm
179  * will first attempt to find a route for the destination that goes out
180  * through an interface where S2 is configured and marked UP.  If such
181  * a route cannot be found, then the best-matching route for D2 will be
182  * selected, ignoring any mismatches between S2 and the interface addresses
183  * on the outgoing interface implied by the route.
184  */
185 typedef enum {
186         IP_WEAK_ES = 0,
187         IP_SRC_PRI_ES,
188         IP_STRONG_ES,
189         IP_MAXVAL_ES
190 } ip_hostmodel_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
    5002 Mon Jul 29 18:04:28 2013
new/usr/src/uts/common/inet/udp/udp_tunables.c
3942 inject sanity into ipadm tcp buffer size properties
3943 _snd_lowat_fraction tcp tunable has no effect
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Peng Dai <peng.dai@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 2013 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */
/* Copyright (c) 1990 Mentat Inc. */

#include <inet/ip.h>
#include <inet/ip6.h>
#include <inet/udp_impl.h>
#include <sys/sunddi.h>

static int
udp_set_buf_prop(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
        return (mod_set_buf_prop(stack->netstack_udp->us_propinfo_tbl, stack,
            cr, pinfo, ifname, pval, flags));
}

static int
udp_get_buf_prop(netstack_t *stack, mod_prop_info_t *pinfo, const char *ifname,
    void *val, uint_t psize, uint_t flags)
{
        return (mod_get_buf_prop(stack->netstack_udp->us_propinfo_tbl, stack,
            pinfo, ifname, val, psize, flags));
}

/*
 * Special checkers for smallest/largest anonymous port so they don't
 * ever happen to be (largest < smallest).
 */
/* ARGSUSED */
static int
udp_smallest_anon_set(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
udp_smallest_anon_set(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
```



```
        unsigned long new_value;
        udp_stack_t *us = stack->netstack_udp;
        udp_stack_t *us = (udp_stack_t *)cbarg;
        int err;

        if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
                return (err);
        /* mod_uint32_value() + pinfo guarantees we're in UDP port range. */
        if (new_value > us->us_largest_anon_port)
                return (ERANGE);
        pinfo->prop_cur_uval = (uint32_t)new_value;
        return (0);
}

/* ARGSUSED */
static int
udp_largest_anon_set(netstack_t *stack, cred_t *cr, mod_prop_info_t *pinfo,
udp_largest_anon_set(void *cbarg, cred_t *cr, mod_prop_info_t *pinfo,
    const char *ifname, const void *pval, uint_t flags)
{
        unsigned long new_value;
        udp_stack_t *us = stack->netstack_udp;
        udp_stack_t *us = (udp_stack_t *)cbarg;
        int err;

        if ((err = mod_uint32_value(pval, pinfo, flags, &new_value)) != 0)
                return (err);
        /* mod_uint32_value() + pinfo guarantees we're in UDP port range. */
        if (new_value < us->us_smallest_anon_port)
                return (ERANGE);
        pinfo->prop_cur_uval = (uint32_t)new_value;
        return (0);
}

/*
 * All of these are alterable, within the min/max values given, at run time.
 *
 * Note: All those tunables which do not start with "_" are Committed and
 * therefore are public. See PSARC 2010/080.
 */
mod_prop_info_t udp_propinfo_tbl[] = {
        /* tunable - 0 */
        { "_wroff_extra", MOD_PROTO_UDP,
            mod_set_uint32, mod_get_uint32,
            {0, 256, 32}, {32} },

        { "_ipv4_ttl", MOD_PROTO_UDP,
            mod_set_uint32, mod_get_uint32,
            {1, 255, 255}, {255} },

        { "_ipv6_hoplimit", MOD_PROTO_UDP,
            mod_set_uint32, mod_get_uint32,
            {0, IPV6_MAX_HOPS, IPV6_DEFAULT_HOPS}, {IPV6_DEFAULT_HOPS} },

        { "smallest_nonpriv_port", MOD_PROTO_UDP,
            mod_set_uint32, mod_get_uint32,
            {1024, (32 * 1024), 1024}, {1024} },

        { "_do_checksum", MOD_PROTO_UDP,
            mod_set_boolean, mod_get_boolean,
            {B_TRUE}, {B_TRUE} },

        { "smallest_anon_port", MOD_PROTO_UDP,
            udp_smallest_anon_set, mod_get_uint32,
            {1024, ULP_MAX_PORT, (32 * 1024)}, {(32 * 1024)} },
```

```
121         { "largest_anon_port", MOD_PROTO_UDP,
122             udp_largest_anon_set, mod_get_uint32,
123             {1024, ULP_MAX_PORT, ULP_MAX_PORT}, {ULP_MAX_PORT} },

125         { "send_buf", MOD_PROTO_UDP,
126             udp_set_buf_prop, udp_get_buf_prop,
127             {UDP_XMIT_LOWATER, ULP_MAX_BUF, UDP_XMIT_HIWATER},
108         { "send_maxbuf", MOD_PROTO_UDP,
109             mod_set_uint32, mod_get_uint32,
110             {UDP_XMIT_LOWATER, (1<<30), UDP_XMIT_HIWATER},
128             {UDP_XMIT_HIWATER} },

130         { "_xmit_lowat", MOD_PROTO_UDP,
131             mod_set_uint32, mod_get_uint32,
132             {0, ULP_MAX_BUF, UDP_XMIT_LOWATER},
115             {0, (1<<30), UDP_XMIT_LOWATER},
133             {UDP_XMIT_LOWATER} },

135         { "recv_buf", MOD_PROTO_UDP,
136             udp_set_buf_prop, udp_get_buf_prop,
137             {UDP_RECV_LOWATER, ULP_MAX_BUF, UDP_RECV_HIWATER},
118         { "recv_maxbuf", MOD_PROTO_UDP,
119             mod_set_uint32, mod_get_uint32,
120             {UDP_RECV_LOWATER, (1<<30), UDP_RECV_HIWATER},
138             {UDP_RECV_HIWATER} },

140         /* tunable - 10 */
141         { "max_buf", MOD_PROTO_UDP,
124         { "_max_buf", MOD_PROTO_UDP,
142             mod_set_uint32, mod_get_uint32,
143             {65536, ULP_MAX_BUF, 2*1024*1024}, {2*1024*1024} },
126             {65536, (1<<30), 2*1024*1024}, {2*1024*1024} },

145         { "_pmtu_discovery", MOD_PROTO_UDP,
146             mod_set_boolean, mod_get_boolean,
147             {B_FALSE}, {B_FALSE} },

149         { "_sendto_ignerr", MOD_PROTO_UDP,
150             mod_set_boolean, mod_get_boolean,
151             {B_FALSE}, {B_FALSE} },

153         { "extra_priv_ports", MOD_PROTO_UDP,
154             mod_set_extra_privports, mod_get_extra_privports,
155             {1, ULP_MAX_PORT, 0}, {0} },

157         { "?", MOD_PROTO_UDP, NULL, mod_get_allprop, {0}, {0} },

159         { NULL, 0, NULL, NULL, {0}, {0} }
160 };
```

*unchanged_portion_omitted_*