```
**********************************************************
   27817 Wed Sep 26 12:51:33 2012
new/usr/src/common/iscsit/iscsit_common.c
inet_pton
**********************************************************
/*
 * 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) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2012, Nexenta Systems, Inc. All rights reserved.
 */
#endif /* ! codereview */

#include <sys/time.h>

#if defined(_KERNEL)
#include <sys/ddi.h>
#include <sys/types.h>
#include <sys/sunddi.h>
#include <sys/socket.h>
#include <inet/tcp.h>
#include <inet/ip.h>
#endif /* ! codereview */
#else
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#endif

#include <sys/iscsit/iscsit_common.h>
#include <sys/iscsi_protocol.h>
#include <sys/iscsit/isns_protocol.h>

void *
iscsit_zalloc(size_t size)
{
#if defined(_KERNEL)
        return (kmem_zalloc(size, KM_SLEEP));
#else
        return (calloc(1, size));
#endif
```

```
}

void
iscsit_free(void *buf, size_t size)     /* ARGSUSED */
{
#if defined(_KERNEL)
        kmem_free(buf, size);
#else
        free(buf);
#endif
}

/*
 * default_port should be the port to be used, if not specified
 * as part of the supplied string 'arg'.
 */

#define NI_MAXHOST      1025
#define NI_MAXSERV      32


struct sockaddr_storage *
it_common_convert_sa(char *arg, struct sockaddr_storage *buf,
    uint32_t default_port)
{
        /* Why does addrbuf need to be this big!??! XXX */
        char            addrbuf[NI_MAXHOST + NI_MAXSERV + 1];
        char            *addr_str;
        char            *port_str;
#ifndef _KERNEL
        char            *errchr;
#endif
        long            tmp_port = 0;
        sa_family_t     af;

        struct sockaddr_in      *sin;
        struct sockaddr_in6     *sin6;
        struct sockaddr_storage *sa = buf;

        if (!arg || !buf) {
                return (NULL);
        }

        bzero(buf, sizeof (struct sockaddr_storage));

        /* don't modify the passed-in string */
        (void) strlcpy(addrbuf, arg, sizeof (addrbuf));

        addr_str = addrbuf;

        if (*addr_str == '[') {
                /*
                 * An IPv6 address must be inside square brackets
                 */
                port_str = strchr(addr_str, ']');
                if (!port_str) {
                        /* No closing bracket */
                        return (NULL);
                }

                /* strip off the square brackets so we can convert */
                addr_str++;
                *port_str = '\0';
                port_str++;

                if (*port_str == ':') {
                        /* TCP port to follow */
                        port_str++;
                } else if (*port_str == '\0') {
                        /* No port specified */
                        port_str = NULL;
                } else {
                        /* malformed */
                        return (NULL);
                }
                af = AF_INET6;
        } else {
                port_str = strchr(addr_str, ':');
                if (port_str) {
                        *port_str = '\0';
                        port_str++;
                }
                af = AF_INET;
        }

        if (port_str) {
#if defined(_KERNEL)
                if (ddi_strtol(port_str, NULL, 10, &tmp_port) != 0) {
                        return (NULL);
                }
#else
                tmp_port = strtol(port_str, &errchr, 10);
#endif
                if (tmp_port < 0 || tmp_port > 65535) {
                        return (NULL);
                }
        } else {
                tmp_port = default_port;
        }

        sa->ss_family = af;

        sin = (struct sockaddr_in *)sa;
        if (af == AF_INET) {
                if (inet_pton(af, addr_str,
                    (void *)&(sin->sin_addr.s_addr)) != 1) {
                        return (NULL);
                }
                /*
                 * intet_pton does not seem to convert to network
                 * order in kernel. This is a workaround until the
                 * inet_pton works or we have our own inet_pton function.
                 */
#ifdef _KERNEL
                sin->sin_addr.s_addr = ntohl((uint32_t)sin->sin_addr.s_addr);
#endif
                sin->sin_port = htons(tmp_port);
        } else {
                sin6 = (struct sockaddr_in6 *)sa;
                if (inet_pton(af, addr_str,
                    (void *)&(sin6->sin6_addr.s6_addr)) != 1) {
                        return (NULL);
                }
                sin6->sin6_port = htons(tmp_port);
        }

        /* successful */
        return (sa);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   49721 Wed Sep 26 12:51:35 2012
new/usr/src/uts/common/fs/nfs/nfs4_srv_deleg.c
inet_pton
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */
#endif /* ! codereview */

#include <sys/systm.h>
#include <rpc/auth.h>
#include <rpc/clnt.h>
#include <nfs/nfs4_kprot.h>
#include <nfs/nfs4.h>
#include <nfs/lm.h>
#include <sys/cmn_err.h>
#include <sys/disp.h>
#include <sys/sdt.h>

#include <sys/pathname.h>

#include <sys/strsubr.h>
#include <sys/ddi.h>

#include <sys/vnode.h>
#include <sys/sdt.h>
#include <inet/common.h>
#include <inet/ip.h>
#include <inet/ip6.h>

#define MAX_READ_DELEGATIONS 5

krwlock_t rfs4_deleg_policy_lock;
srv_deleg_policy_t rfs4_deleg_policy = SRV_NEVER_DELEGATE;
static int rfs4_deleg_wlp = 5;
kmutex_t rfs4_deleg_lock;
static int rfs4_deleg_disabled;
static int rfs4_max_setup_cb_tries = 5;

#ifdef DEBUG
```

```
static int rfs4_test_cbgetattr_fail = 0;
int rfs4_cb_null;
int rfs4_cb_debug;
int rfs4_deleg_debug;

#endif

static void rfs4_recall_file(rfs4_file_t *,
    void (*recall)(rfs4_deleg_state_t *, bool_t),
    bool_t, rfs4_client_t *);
static  void            rfs4_revoke_file(rfs4_file_t *);
static  void            rfs4_cb_chflush(rfs4_cbinfo_t *);
static  CLIENT          *rfs4_cb_getch(rfs4_cbinfo_t *);
static  void            rfs4_cb_freech(rfs4_cbinfo_t *, CLIENT *, bool_t);
static rfs4_deleg_state_t *rfs4_deleg_state(rfs4_state_t *,
    open_delegation_type4, int *);

/*
 * Convert a universal address to an transport specific
 * address using inet_pton.
 */
static int
uaddr2sockaddr(int af, char *ua, void *ap, in_port_t *pp)
{
        int dots = 0, i, j, len, k;
        unsigned char c;
        in_port_t port = 0;

        len = strlen(ua);

        for (i = len-1; i >= 0; i--) {

                if (ua[i] == '.')
                        dots++;

                if (dots == 2) {

                        ua[i] = '\0';
                        /*
                         * We use k to remember were to stick '.' back, since
                         * ua was kmem_allocateded from the pool len+1.
                         */
                        k = i;
                        if (inet_pton(af, ua, ap) == 1) {

                                c = 0;

                                for (j = i+1; j < len; j++) {
                                        if (ua[j] == '.') {
                                                port = c << 8;
                                                c = 0;
                                        } else if (ua[j] >= '0' &&
                                            ua[j] <= '9') {
                                                c *= 10;
                                                c += ua[j] - '0';
                                        } else {
                                                ua[k] = '.';
                                                return (EINVAL);
                                        }
                                }
                                port += c;


                                /* reset to network order */
                                if (af == AF_INET) {
                                        *(uint32_t *)ap =
                                                htonl(*(uint32_t *)ap);
                                *pp = htons(port);
                                } else {
                                        int ix;
                                        uint16_t *sap;

                                        for (sap = ap, ix = 0; ix <
                                            sizeof (struct in6_addr) /
                                            sizeof (uint16_t); ix++)
                                                sap[ix] = htons(sap[ix]);

                                        *pp = htons(port);
                                }

                                ua[k] = '.';
                                return (0);
                        } else {
                                ua[k] = '.';
                                return (EINVAL);
                        }
                }
        }

        return (EINVAL);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   24552 Wed Sep 26 12:51:38 2012
new/usr/src/uts/common/fs/sockfs/nl7c.c
inet_pton
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  */
  25 /*
  26  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  27  */
  28 #endif /* ! codereview */

  30 /*
  31  * NL7C (Network Layer 7 Cache) as part of SOCKFS provides an in-kernel
  32  * gateway cache for the request/response message based L7 protocol HTTP
  33  * (Hypertext Transfer Protocol, see HTTP/1.1 RFC2616) in a semantically
  34  * transparent manner.
  35  *
  36  * Neither the requesting user agent (client, e.g. web browser) nor the
  37  * origin server (e.g. webserver) that provided the response cached by
  38  * NL7C are impacted in any way.
  39  *
  40  * Note, currently NL7C only processes HTTP messages via the embedded
  41  * URI of scheme http (not https nor any other), additional scheme are
  42  * intended to be supported as is practical such that much of the NL7C
  43  * framework may appear more general purpose then would be needed just
  44  * for an HTTP gateway cache.
  45  *
  46  * NL7C replaces NCA (Network Cache and Accelerator) and in the future
  47  * NCAS (NCA/SSL).
  48  *
  49  * Further, NL7C uses all NCA configuration files, see "/etc/nca/", the
  50  * NCA socket API, "AF_NCA", and "ndd /dev/nca" for backwards compatibility.
  51  */

  53 #include <sys/systm.h>
  54 #include <sys/strsun.h>
  55 #include <sys/strsubr.h>
  56 #include <inet/common.h>
  57 #include <inet/led.h>
  58 #include <inet/mi.h>
  59 #include <netinet/in.h>
  60 #include <fs/sockfs/nl7c.h>
  61 #include <fs/sockfs/nl7curi.h>
```

```
  62 #include <fs/sockfs/socktpi.h>

  64 #include <inet/nca/ncadoorhdr.h>
  65 #include <inet/nca/ncalogd.h>
  66 #include <inet/nca/ncandd.h>
  67 #include <inet/ip.h>
  68 #endif /* ! codereview */

  70 #include <sys/promif.h>

  72 /*
  73  * NL7C, NCA, NL7C logger enabled:
  74  */

  76 boolean_t       nl7c_enabled = B_FALSE;

  78 boolean_t       nl7c_logd_enabled = B_FALSE;
  79 boolean_t       nl7c_logd_started = B_FALSE;
  80 boolean_t       nl7c_logd_cycle = B_TRUE;

  82 /*
  83  * Some externs:
  84  */

  26 extern int      inet_pton(int, char *, void *);
  85 extern void     nl7c_uri_init(void);
  86 extern boolean_t nl7c_logd_init(int, caddr_t *);
  87 extern void     nl7c_nca_init(void);

  89 /*
  90  * nl7c_addr_t - a singly linked grounded list, pointed to by *nl7caddrs,
  91  * constructed at init time by parsing "/etc/nca/ncaport.conf".
  92  *
  93  * This list is searched at bind(3SOCKET) time when an application doesn't
  94  * explicitly set AF_NCA but instead uses AF_INET, if a match is found then
  95  * the underlying socket is marked sti_nl7c_flags NL7C_ENABLED.
  96  */

  98 typedef struct nl7c_addr_s {
  99         struct nl7c_addr_s *next;       /* next entry */
 100         sa_family_t     family;         /* addr type, only INET and INET6 */
 101         uint16_t        port;           /* port */
 102         union {
 103                 ipaddr_t        v4;     /* IPv4 address */
 104                 in6_addr_t      v6;     /* IPv6 address */
 105                 void            *align; /* foce alignment */
 106         }               addr;           /* address */

 108         struct sonode   *listener;      /* listen()er's sonode */
 109         boolean_t       temp;           /* temporary addr via add_addr() ? */
 110 } nl7c_addr_t;
______unchanged_portion_omitted_

 307 /*
 308  * Inet ASCII to binary.
 309  *
 310  * Note, it's assumed that *s is a valid zero byte terminated string, and
 311  * that *p is a zero initialized struct (this is important as the value of
 312  * INADDR_ANY and IN6ADDR_ANY is zero).
 313  */

 315 static int
 316 inet_atob(char *s, nl7c_addr_t *p)
 317 {
 318         if (strcmp(s, "*") == 0) {
```

```
                /* INADDR_ANY */
                p->family = AF_INET;
                return (0);
        }
        if (strcmp(s, "::") == 0) {
                /* IN6ADDR_ANY */
                p->family = AF_INET6;
                return (0);
        }
        /* IPv4 address ? */
        if (inet_pton(AF_INET, s, &p->addr.v4) != 1) {
                /* Nop, IPv6 address ? */
                if (inet_pton(AF_INET6, s, &p->addr.v6) != 1) {
                        /* Nop, return error */
                        return (1);
                }
                p->family = AF_INET6;
        } else {
                p->family = AF_INET;
                p->addr.v4 = ntohl(p->addr.v4);
        }

#endif /* ! codereview */
        return (0);
}

/*
 * Open and read each line from "/etc/nca/ncaport.conf", the syntax of a
 * ncaport.conf file line is:
 *
 *      ncaport=IPaddr/Port[/Proxy]
 *
 * Where:
 *
 * ncaport - the only token recognized.
 *
 *  IPaddr - an IPv4 numeric dot address (e.g. 192.168.84.71) or '*' for
 *           INADDR_ANY, or an IPv6 numeric address or "::" for IN6ADDR_ANY.
 *
 *       / - IPaddr/Port separator.
 *
 *    Port - a TCP decimal port number.
 *
 * Note, all other lines will be ignored.
 */

static void
ncaportconf_read(void)
{
        int     ret;
        struct vnode *vp;
        char    c;
        ssize_t resid;
        char    buf[1024];
        char    *ebp = &buf[sizeof (buf)];
        char    *bp = ebp;
        offset_t off = 0;
        enum parse_e {START, TOK, ADDR, PORT, EOL} parse = START;
        nl7c_addr_t *addrp = NULL;
        char    *ncaport = "ncaport";
        char    string[] = "XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX";
        char    *stringp;
        char    *tok;
        char    *portconf = "/etc/nca/ncaport.conf";

        ret = vn_open(portconf, UIO_SYSSPACE, FREAD, 0, &vp, 0, 0);
```

```
        if (ret == ENOENT) {
                /* No portconf file, nothing to do */
                return;
        }
        if (ret != 0) {
                /* Error of some sort, tell'm about it */
                cmn_err(CE_WARN, "%s: open error %d", portconf, ret);
                return;
        }
        /*
         * Read portconf one buf[] at a time, parse one char at a time.
         */
        for (;;) {
                if (bp == ebp) {
                        /* Nothing left in buf[], read another */
                        ret = vn_rdwr(UIO_READ, vp, buf, sizeof (buf), off,
                            UIO_SYSSPACE, 0, (rlim64_t)0, CRED(), &resid);
                        if (ret != 0) {
                                /* Error of some sort, tell'm about it */
                                cmn_err(CE_WARN, "%s: read error %d",
                                    portconf, ret);
                                break;
                        }
                        if (resid == sizeof (buf)) {
                                /* EOF, done */
                                break;
                        }
                        /* Initilize per buf[] state */
                        bp = buf;
                        ebp = &buf[sizeof (buf) - resid];
                        off += sizeof (buf) - resid;
                }
                c = *bp++;
                switch (parse) {
                case START:
                        /* Initilize all per file line state */
                        if (addrp == NULL) {
                                addrp = kmem_zalloc(sizeof (*addrp),
                                    KM_NOSLEEP);
                        }
                        tok = ncaport;
                        stringp = string;
                        parse = TOK;
                        /*FALLTHROUGH*/
                case TOK:
                        if (c == '#') {
                                /* Comment through end of line */
                                parse = EOL;
                                break;
                        }
                        if (isalpha(c)) {
                                if (c != *tok++) {
                                        /* Only know one token, skip */
                                        parse = EOL;
                                }
                        } else if (c == '=') {
                                if (*tok != NULL) {
                                        /* Only know one token, skip */
                                        parse = EOL;
                                        break;
                                }
                                parse = ADDR;
                        } else if (c == '\n') {
                                /* Found EOL, empty line, next line */
                                parse = START;
                        } else {
```

```
				/* Unexpected char, skip */
				parse = EOL;
			}
			break;

		case ADDR:
			if (c == '/') {
				/* addr/port separator, end of addr */
				*stringp = NULL;
				if (inet_atob(string, addrp)) {
					/* Bad addr, skip */
					parse = EOL;
				} else {
					stringp = string;
					parse = PORT;
				}
			} else {
				/* Save char to string */
				if (stringp ==
				    &string[sizeof (string) - 1]) {
					/* Would overflow, skip */
					parse = EOL;
				} else {
					/* Copy IP addr char */
					*stringp++ = c;
				}
			}
			break;

		case PORT:
			if (isdigit(c)) {
				/* Save char to string */
				if (stringp ==
				    &string[sizeof (string) - 1]) {
					/* Would overflow, skip */
					parse = EOL;
				} else {
					/* Copy port digit char */
					*stringp++ = c;
				}
				break;
			} else if (c == '#' || isspace(c)) {
				/* End of port number, convert */
				*stringp = NULL;
				addrp->port = ntohs(atou(string));

				/* End of parse, add entry */
				nl7c_addr_add(addrp);
				addrp = NULL;
				parse = EOL;
			} else {
				/* Unrecognized char, skip */
				parse = EOL;
				break;
			}
			if (c == '\n') {
				/* Found EOL, start on next line */
				parse = START;
			}
			break;

		case EOL:
			if (c == '\n') {
				/* Found EOL, start on next line */
				parse = START;
			}
```

```
				break;
			}

		}
		if (addrp != NULL) {
			kmem_free(addrp, sizeof (*addrp));
		}
		(void) VOP_CLOSE(vp, FREAD, 1, (offset_t)0, CRED(), NULL);
		VN_RELE(vp);
}

/*
 * Open and read each line from "/etc/nca/ncakmod.conf" and parse looking
 * for the NCA enabled, the syntax is: status=enabled, all other lines will
 * be ignored.
 */

static void
ncakmodconf_read(void)
{
	int	ret;
	struct vnode *vp;
	char	c;
	ssize_t resid;
	char	buf[1024];
	char	*ebp = &buf[sizeof (buf)];
	char	*bp = ebp;
	offset_t off = 0;
	enum parse_e {START, TOK, EOL} parse = START;
	char	*status = "status=enabled";
	char	*tok;
	char	*ncakmod = "/etc/nca/ncakmod.conf";

	ret = vn_open(ncakmod, UIO_SYSSPACE, FREAD, 0, &vp, 0, 0);
	if (ret == ENOENT) {
		/* No ncakmod file, nothing to do */
		return;
	}
	if (ret != 0) {
		/* Error of some sort, tell'm about it */
		cmn_err(CE_WARN, "%s: open error %d", status, ret);
		return;
	}
	/*
	 * Read ncakmod one buf[] at a time, parse one char at a time.
	 */
	for (;;) {
		if (bp == ebp) {
			/* Nothing left in buf[], read another */
			ret = vn_rdwr(UIO_READ, vp, buf, sizeof (buf), off,
			    UIO_SYSSPACE, 0, (rlim64_t)0, CRED(), &resid);
			if (ret != 0) {
				/* Error of some sort, tell'm about it */
				cmn_err(CE_WARN, "%s: read error %d",
				    status, ret);
				break;
			}
			if (resid == sizeof (buf)) {
				/* EOF, done */
				break;
			}
			/* Initilize per buf[] state */
			bp = buf;
			ebp = &buf[sizeof (buf) - resid];
			off += sizeof (buf) - resid;
		}
```

```
                c = *bp++;
                switch (parse) {
                case START:
                        /* Initilize all per file line state */
                        tok = status;
                        parse = TOK;
                        /*FALLTHROUGH*/
                case TOK:
                        if (c == '#') {
                                /* Comment through end of line */
                                parse = EOL;
                                break;
                        }
                        if (isalpha(c) || c == '=') {
                                if (c != *tok++) {
                                        /* Only know one token, skip */
                                        parse = EOL;
                                }
                        } else if (c == '\n') {
                                /*
                                 * Found EOL, if tok found done,
                                 * else start on next-line.
                                 */
                                if (*tok == NULL) {
                                        nl7c_enabled = B_TRUE;
                                        goto done;
                                }
                                parse = START;
                        } else {
                                /* Unexpected char, skip */
                                parse = EOL;
                        }
                        break;

                case EOL:
                        if (c == '\n') {
                                /* Found EOL, start on next line */
                                parse = START;
                        }
                        break;
                }

        }
done:
        (void) VOP_CLOSE(vp, FREAD, 1, (offset_t)0, CRED(), NULL);
        VN_RELE(vp);
}

/*
 * Open and read each line from "/etc/nca/ncalogd.conf" and parse for
 * the tokens and token text (i.e. key and value ncalogd.conf(4)):
 *
 *      status=enabled
 *
 *      logd_file_size=[0-9]+
 *
 *      logd_file_name=["]filename( filename)*["]
 */

static int      file_size = 1000000;
static caddr_t  fnv[NCA_FIOV_SZ];

static void
ncalogdconf_read(void)
{
        int     ret;
```

```
        struct vnode *vp;
        char    c;
        int     sz;
        ssize_t resid;
        char    buf[1024];
        char    *ebp = &buf[sizeof (buf)];
        char    *bp = ebp;
        offset_t off = 0;
        enum parse_e {START, TOK, TEXT, EOL} parse = START;
        char    *tokstatus = "status\0enabled";
        char    *toksize = "logd_file_size";
        char    *tokfile = "logd_path_name";
        char    *tokstatusp;
        char    *toksizep;
        char    *tokfilep;
        char    *tok;
        int     tokdelim = 0;
        char    *ncalogd = "/etc/nca/ncalogd.conf";
        char    *ncadeflog = "/var/nca/log";
        char    file[TYPICALMAXPATHLEN] = {0};
        char    *fp = file;
        caddr_t *fnvp = fnv;

        ret = vn_open(ncalogd, UIO_SYSSPACE, FREAD, 0, &vp, 0, 0);
        if (ret == ENOENT) {
                /* No ncalogd file, nothing to do */
                return;
        }
        if (ret != 0) {
                /* Error of some sort, tell'm about it */
                cmn_err(CE_WARN, "ncalogdconf_read: %s: open error(%d).",
                    ncalogd, ret);
                return;
        }
        /*
         * Read ncalogd.conf one buf[] at a time, parse one char at a time.
         */
        for (;;) {
                if (bp == ebp) {
                        /* Nothing left in buf[], read another */
                        ret = vn_rdwr(UIO_READ, vp, buf, sizeof (buf), off,
                            UIO_SYSSPACE, 0, (rlim64_t)0, CRED(), &resid);
                        if (ret != 0) {
                                /* Error of some sort, tell'm about it */
                                cmn_err(CE_WARN, "%s: read error %d",
                                    ncalogd, ret);
                                break;
                        }
                        if (resid == sizeof (buf)) {
                                /* EOF, done */
                                break;
                        }
                        /* Initilize per buf[] state */
                        bp = buf;
                        ebp = &buf[sizeof (buf) - resid];
                        off += sizeof (buf) - resid;
                }
                c = *bp++;
                switch (parse) {
                case START:
                        /* Initilize all per file line state */
                        tokstatusp = tokstatus;
                        toksizep = toksize;
                        tokfilep = tokfile;
                        tok = NULL;
                        parse = TOK;
                        sz = 0;
                        /*FALLTHROUGH*/
                case TOK:
                        if (isalpha(c) || c == ’_’) {
                                /*
                                 * Found a valid tok char, if matches
                                 * any of the tokens continue else NULL
                                 * then string pointer.
                                 */
                                if (tokstatusp != NULL && c != *tokstatusp++)
                                        tokstatusp = NULL;
                                if (toksizep != NULL && c != *toksizep++)
                                        toksizep = NULL;
                                if (tokfilep != NULL && c != *tokfilep++)
                                        tokfilep = NULL;

                                if (tokstatusp == NULL &&
                                    toksizep == NULL &&
                                    tokfilep == NULL) {
                                        /*
                                         * All tok string pointers are NULL
                                         * so skip rest of line.
                                         */
                                        parse = EOL;
                                }
                        } else if (c == ’=’) {
                                /*
                                 * Found tok separator, if tok found get
                                 * tok text, else skip rest of line.
                                 */
                                if (tokstatusp != NULL && *tokstatusp == NULL)
                                        tok = tokstatus;
                                else if (toksizep != NULL && *toksizep == NULL)
                                        tok = toksize;
                                else if (tokfilep != NULL && *tokfilep == NULL)
                                        tok = tokfile;
                                if (tok != NULL)
                                        parse = TEXT;
                                else
                                        parse = EOL;
                        } else if (c == ’\n’) {
                                /* Found EOL, start on next line */
                                parse = START;
                        } else {
                                /* Comment or unknown char, skip rest of line */
                                parse = EOL;
                        }
                        break;
                case TEXT:
                        if (c == ’\n’) {
                                /*
                                 * Found EOL, finish up tok text processing
                                 * (if any) and start on next line.
                                 */
                                if (tok == tokstatus) {
                                        if (*++tokstatusp == NULL)
                                                nl7c_logd_enabled = B_TRUE;
                                } else if (tok == toksize) {
                                        file_size = sz;
                                } else if (tok == tokfile) {
                                        if (tokdelim == 0) {
                                                /* Non delimited path name */
                                                *fnvp++ = strdup(file);
                                        } else if (fp != file) {
                                                /* No closing delimiter */
                                                /*EMPTY*/;
```

```
                                        }
                                }
                                parse = START;
                        } else if (tok == tokstatus) {
                                if (! isalpha(c) || *++tokstatusp == NULL ||
                                    c != *tokstatusp) {
                                        /* Not enabled, skip line */
                                        parse = EOL;
                                }
                        } else if (tok == toksize) {
                                if (isdigit(c)) {
                                        sz *= 10;
                                        sz += c - ’0’;
                                } else {
                                        /* Not a decimal digit, skip line */
                                        parse = EOL;
                                }
                        } else {
                                /* File name */
                                if (c == ’"’ && tokdelim++ == 0) {
                                        /* Opening delimiter, skip */
                                        /*EMPTY*/;
                                } else if (c == ’"’ || c == ’ ’) {
                                        /* List delim or filename separator */
                                        *fnvp++ = strdup(file);
                                        fp = file;
                                } else if (fp < &file[sizeof (file) - 1]) {
                                        /* Filename char */
                                        *fp++ = c;
                                } else {
                                        /* Filename to long, skip line */
                                        parse = EOL;
                                }
                        }
                        break;

                case EOL:
                        if (c == ’\n’) {
                                /* Found EOL, start on next line */
                                parse = START;
                        }
                        break;
                }

        }
done:
        (void) VOP_CLOSE(vp, FREAD, 1, (offset_t)0, CRED(), NULL);
        VN_RELE(vp);

        if (nl7c_logd_enabled) {
                if (fnvp == fnv) {
                        /*
                         * No logfile was specified and found so
                         * so use defualt NCA log file path.
                         */
                        *fnvp++ = strdup(ncadeflog);
                }
                if (fnvp < &fnv[NCA_FIOV_SZ]) {
                        /* NULL terminate list */
                        *fnvp = NULL;
                }
        }
}

void
nl7clogd_startup(void)
```

```
{
        static kmutex_t startup;

        /*
         * Called on the first log() attempt, have to wait until then to
         * initialize logd as at logdconf_read() the root fs is read-only.
         */
        mutex_enter(&startup);
        if (nl7c_logd_started) {
                /* Lost the race, nothing todo */
                mutex_exit(&startup);
                return;
        }
        nl7c_logd_started = B_TRUE;
        if (! nl7c_logd_init(file_size, fnv)) {
                /* Failure, disable logging */
                nl7c_logd_enabled = B_FALSE;
                cmn_err(CE_WARN, "nl7clogd_startup: failed, disabling loggin");
                mutex_exit(&startup);
                return;
        }
        mutex_exit(&startup);
}


void
nl7c_startup()
{
        /*
         * Open, read, and parse the NCA logd configuration file,
         * then initialize URI processing and NCA compat.
         */
        ncalogdconf_read();
        nl7c_uri_init();
        nl7c_nca_init();
}

void
nl7c_init()
{
        /* Open, read, and parse the NCA kmod configuration file */
        ncakmodconf_read();

        if (nl7c_enabled) {
                /*
                 * NL7C is enabled so open, read, and parse
                 * the NCA address/port configuration file
                 * and call startup() to finish config/init.
                 */
                ncaportconf_read();
                nl7c_startup();
        }
}

/*
 * The main processing function called by accept() on a newly created
 * socket prior to returning it to the caller of accept().
 *
 * Here data is read from the socket until a completed L7 request parse
 * is completed. Data will be read in the context of the user thread
 * which called accept(), when parse has been completed either B_TRUE
 * or B_FALSE will be returned.
 *
 * If NL7C successfully process the L7 protocol request, i.e. generates
 * a response, B_TRUE will be returned.
 *
```

```
 * Else, B_FALSE will be returned if NL7C can't process the request:
 *
 * 1) Couldn't locate a URI within the request.
 *
 * 2) URI scheme not reqcognized.
 *
 * 3) A request which can't be processed.
 *
 * 4) A request which could be processed but NL7C dosen't currently have
 *    the response data. In which case NL7C will parse the returned response
 *    from the application for possible caching for subsequent request(s).
 */

volatile uint64_t nl7c_proc_cnt = 0;
volatile uint64_t nl7c_proc_error = 0;
volatile uint64_t nl7c_proc_ETIME = 0;
volatile uint64_t nl7c_proc_again = 0;
volatile uint64_t nl7c_proc_next = 0;
volatile uint64_t nl7c_proc_rcv = 0;
volatile uint64_t nl7c_proc_noLRI = 0;
volatile uint64_t nl7c_proc_nodata = 0;
volatile uint64_t nl7c_proc_parse = 0;

boolean_t
nl7c_process(struct sonode *so, boolean_t nonblocking)
{
        vnode_t *vp = SOTOV(so);
        sotpi_info_t *sti = SOTOTPI(so);
        mblk_t  *rmp = sti->sti_nl7c_rcv_mp;
        clock_t timout;
        rval_t  rval;
        uchar_t pri;
        int     pflag;
        int     error;
        boolean_t more;
        boolean_t ret = B_FALSE;
        boolean_t first = B_TRUE;
        boolean_t pollin = (sti->sti_nl7c_flags & NL7C_POLLIN);

        nl7c_proc_cnt++;

        /* Caller has so_lock enter()ed */
        error = so_lock_read_intr(so, nonblocking ? FNDELAY|FNONBLOCK : 0);
        if (error) {
                /* Couldn't read lock, pass on this socket */
                sti->sti_nl7c_flags = 0;
                nl7c_proc_noLRI++;
                return (B_FALSE);
        }
        /* Exit so_lock for now, will be reenter()ed prior to return */
        mutex_exit(&so->so_lock);

        if (pollin)
                sti->sti_nl7c_flags &= ~NL7C_POLLIN;

        /* Initialize some kstrgetmsg() constants */
        pflag = MSG_ANY | MSG_DELAYERROR;
        pri = 0;
        if (nonblocking) {
                /* Non blocking so don't block */
                timout = 0;
        } else if (sti->sti_nl7c_flags & NL7C_SOPERSIST) {
                /* 2nd or more time(s) here so use keep-alive value */
                timout = nca_http_keep_alive_timeout;
        } else {
                /* 1st time here so use connection value */
```

```
		timout = nca_http_timeout;
	}

	rval.r_vals = 0;
	do {
		/*
		 * First time through, if no data left over from a previous
		 * kstrgetmsg() then try to get some, else just process it.
		 *
		 * Thereafter, rmp = NULL after the successful kstrgetmsg()
		 * so try to get some new data and append to list (i.e. until
		 * enough fragments are collected for a successful parse).
		 */
		if (rmp == NULL) {

			error = kstrgetmsg(vp, &rmp, NULL, &pri, &pflag,
			    timout, &rval);
			if (error) {
				if (error == ETIME) {
					/* Timeout */
					nl7c_proc_ETIME++;
				} else if (error != EWOULDBLOCK) {
					/* Error of some sort */
					nl7c_proc_error++;
					rval.r_v.r_v2 = error;
					sti->sti_nl7c_flags = 0;
					break;
				}
				error = 0;
			}
			if (rmp != NULL) {
				mblk_t	*mp = sti->sti_nl7c_rcv_mp;


				if (mp == NULL) {
					/* Just new data, common case */
					sti->sti_nl7c_rcv_mp = rmp;
				} else {
					/* Add new data to tail */
					while (mp->b_cont != NULL)
						mp = mp->b_cont;
					mp->b_cont = rmp;
				}
			}
			if (sti->sti_nl7c_rcv_mp == NULL) {
				/* No data */
				nl7c_proc_nodata++;
				if (timout > 0 || (first && pollin)) {
					/* Expected data so EOF */
					ret = B_TRUE;
				} else if (sti->sti_nl7c_flags &
				    NL7C_SOPERSIST) {
					/* Persistent so just checking */
					ret = B_FALSE;
				}
				break;
			}
			rmp = NULL;
		}
		first = B_FALSE;
	again:
		nl7c_proc_parse++;

		more = nl7c_parse(so, nonblocking, &ret);

		if (ret == B_TRUE && (sti->sti_nl7c_flags & NL7C_SOPERSIST)) {
```

```
			/*
			 * Parse complete, cache hit, response on its way,
			 * socket is persistent so try to process the next
			 * request.
			 */
			if (nonblocking) {
				ret = B_FALSE;
				break;
			}
			if (sti->sti_nl7c_rcv_mp) {
				/* More recv-side data, pipelined */
				nl7c_proc_again++;
				goto again;
			}
			nl7c_proc_next++;
			if (nonblocking)
				timout = 0;
			else
				timout = nca_http_keep_alive_timeout;

			more = B_TRUE;
		}

	} while (more);

	if (sti->sti_nl7c_rcv_mp) {
		nl7c_proc_rcv++;
	}
	sti->sti_nl7c_rcv_rval = rval.r_vals;
	/* Renter so_lock, caller called with it enter()ed */
	mutex_enter(&so->so_lock);
	so_unlock_read(so);

	return (ret);
}
```

```
**********************************************************
  140149 Wed Sep 26 12:51:41 2012
new/usr/src/uts/common/inet/ip.h
inet_pton
**********************************************************
/*
 * 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 2012 Nexenta Systems, Inc. All rights reserved.
 */
#endif /* ! codereview */

#ifndef _INET_IP_H
#define _INET_IP_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/isa_defs.h>
#include <sys/types.h>
#include <inet/mib2.h>
#include <inet/nd.h>
#include <sys/atomic.h>
#include <net/if_dl.h>
#include <net/if.h>
#include <netinet/ip.h>
#include <netinet/igmp.h>
#include <sys/neti.h>
#include <sys/hook.h>
#include <sys/hook_event.h>
#include <sys/hook_impl.h>
#include <inet/ip_stack.h>

#ifdef _KERNEL
#include <netinet/ip6.h>
#include <sys/avl.h>
#include <sys/list.h>
#include <sys/vmem.h>
#include <sys/squeue.h>
#include <net/route.h>
#include <sys/systm.h>
#include <net/radix.h>
```

```
#include <sys/modhash.h>

#ifdef DEBUG
#define CONN_DEBUG
#endif

#define IP_DEBUG
/*
 * The mt-streams(9F) flags for the IP module; put here so that other
 * "drivers" that are actually IP (e.g., ICMP, UDP) can use the same set
 * of flags.
 */
#define IP_DEVMTFLAGS D_MP
#endif  /* _KERNEL */

#define IP_MOD_NAME     "ip"
#define IP_DEV_NAME     "/dev/ip"
#define IP6_DEV_NAME    "/dev/ip6"

#define UDP_MOD_NAME    "udp"
#define UDP_DEV_NAME    "/dev/udp"
#define UDP6_DEV_NAME   "/dev/udp6"

#define TCP_MOD_NAME    "tcp"
#define TCP_DEV_NAME    "/dev/tcp"
#define TCP6_DEV_NAME   "/dev/tcp6"

#define SCTP_MOD_NAME   "sctp"

#ifndef _IPADDR_T
#define _IPADDR_T
typedef uint32_t ipaddr_t;
#endif

/* Number of bits in an address */
#define IP_ABITS                32
#define IPV4_ABITS              IP_ABITS
#define IPV6_ABITS              128
#define IP_MAX_HW_LEN   40

#define IP_HOST_MASK            (ipaddr_t)0xffffffffU

#define IP_CSUM(mp, off, sum)           (~ip_cksum(mp, off, sum) & 0xFFFF)
#define IP_CSUM_PARTIAL(mp, off, sum)   ip_cksum(mp, off, sum)
#define IP_BCSUM_PARTIAL(bp, len, sum)  bcksum(bp, len, sum)

#define ILL_FRAG_HASH_TBL_COUNT ((unsigned int)64)
#define ILL_FRAG_HASH_TBL_SIZE  (ILL_FRAG_HASH_TBL_COUNT * sizeof (ipfb_t))

#define IPV4_ADDR_LEN                   4
#define IP_ADDR_LEN                     IPV4_ADDR_LEN
#define IP_ARP_PROTO_TYPE               0x0800

#define IPV4_VERSION                    4
#define IP_VERSION                      IPV4_VERSION
#define IP_SIMPLE_HDR_LENGTH_IN_WORDS   5
#define IP_SIMPLE_HDR_LENGTH            20
#define IP_MAX_HDR_LENGTH               60

#define IP_MAX_OPT_LENGTH (IP_MAX_HDR_LENGTH-IP_SIMPLE_HDR_LENGTH)

#define IP_MIN_MTU                      (IP_MAX_HDR_LENGTH + 8) /* 68 bytes */

/*
 * XXX IP_MAXPACKET is defined in <netinet/ip.h> as well. At some point the
 * 2 files should be cleaned up to remove all redundant definitions.
 */
#define IP_MAXPACKET                    65535
#define IP_SIMPLE_HDR_VERSION \
        ((IP_VERSION << 4) | IP_SIMPLE_HDR_LENGTH_IN_WORDS)

#define UDPH_SIZE                       8

/*
 * Constants and type definitions to support IP IOCTL commands
 */
#define IP_IOCTL                        (('i'<<8)|'p')
#define IP_IOC_IRE_DELETE               4
#define IP_IOC_IRE_DELETE_NO_REPLY      5
#define IP_IOC_RTS_REQUEST              7

/* Common definitions used by IP IOCTL data structures */
typedef struct ipllcmd_s {
        uint_t  ipllc_cmd;
        uint_t  ipllc_name_offset;
        uint_t  ipllc_name_length;
} ipllc_t;

/* IP IRE Delete Command Structure. */
typedef struct ipid_s {
        ipllc_t ipid_ipllc;
        uint_t  ipid_ire_type;
        uint_t  ipid_addr_offset;
        uint_t  ipid_addr_length;
        uint_t  ipid_mask_offset;
        uint_t  ipid_mask_length;
} ipid_t;

#define ipid_cmd                ipid_ipllc.ipllc_cmd

#ifdef _KERNEL
/*
 * Temporary state for ip options parser.
 */
typedef struct ipoptp_s
{
        uint8_t         *ipoptp_next;   /* next option to look at */
        uint8_t         *ipoptp_end;    /* end of options */
        uint8_t         *ipoptp_cur;    /* start of current option */
        uint8_t         ipoptp_len;     /* length of current option */
        uint32_t        ipoptp_flags;
} ipoptp_t;

/*
 * Flag(s) for ipoptp_flags
 */
#define IPOPTP_ERROR    0x00000001
#endif  /* _KERNEL */

/* Controls forwarding of IP packets, set via ipadm(1M)/ndd(1M) */
#define IP_FORWARD_NEVER        0
#define IP_FORWARD_ALWAYS       1

#define WE_ARE_FORWARDING(ipst) ((ipst)->ips_ip_forwarding == IP_FORWARD_ALWAYS)

#define IPH_HDR_LENGTH(ipha)                                            \
        ((int)(((ipha_t *)ipha)->ipha_version_and_hdr_length & 0xF) << 2)

#define IPH_HDR_VERSION(ipha)                                           \
        ((int)(((ipha_t *)ipha)->ipha_version_and_hdr_length) >> 4)

#ifdef _KERNEL
```

```
/*
 * IP reassembly macros.  We hide starting and ending offsets in b_next and
 * b_prev of messages on the reassembly queue.  The messages are chained using
 * b_cont.  These macros are used in ip_reassemble() so we don't have to see
 * the ugly casts and assignments.
 * Note that the offsets are <= 64k i.e. a uint_t is sufficient to represent
 * them.
 */
#define IP_REASS_START(mp)              ((uint_t)(uintptr_t)((mp)->b_next))
#define IP_REASS_SET_START(mp, u)       \
        ((mp)->b_next = (mblk_t *)(uintptr_t)(u))
#define IP_REASS_END(mp)                ((uint_t)(uintptr_t)((mp)->b_prev))
#define IP_REASS_SET_END(mp, u)         \
        ((mp)->b_prev = (mblk_t *)(uintptr_t)(u))

#define IP_REASS_COMPLETE       0x1
#define IP_REASS_PARTIAL        0x2
#define IP_REASS_FAILED         0x4

/*
 * Test to determine whether this is a module instance of IP or a
 * driver instance of IP.
 */
#define CONN_Q(q)       (WR(q)->q_next == NULL)

#define Q_TO_CONN(q)    ((conn_t *)(q)->q_ptr)
#define Q_TO_TCP(q)     (Q_TO_CONN((q))->conn_tcp)
#define Q_TO_UDP(q)     (Q_TO_CONN((q))->conn_udp)
#define Q_TO_ICMP(q)    (Q_TO_CONN((q))->conn_icmp)
#define Q_TO_RTS(q)     (Q_TO_CONN((q))->conn_rts)

#define CONNP_TO_WQ(connp)      ((connp)->conn_wq)
#define CONNP_TO_RQ(connp)      ((connp)->conn_rq)

#define GRAB_CONN_LOCK(q)       {                               \
        if (q != NULL && CONN_Q(q))                             \
                mutex_enter(&(Q_TO_CONN(q))->conn_lock);        \
}

#define RELEASE_CONN_LOCK(q)    {                               \
        if (q != NULL && CONN_Q(q))                             \
                mutex_exit(&(Q_TO_CONN(q))->conn_lock);         \
}

/*
 * Ref counter macros for ioctls. This provides a guard for TCP to stop
 * tcp_close from removing the rq/wq whilst an ioctl is still in flight on the
 * stream. The ioctl could have been queued on e.g. an ipsq. tcp_close will wait
 * until the ioctlref count is zero before proceeding.
 * Ideally conn_oper_pending_ill would be used for this purpose. However, in the
 * case where an ioctl is aborted or interrupted, it can be cleared prematurely.
 * There are also some race possibilities between ip and the stream head which
 * can also end up with conn_oper_pending_ill being cleared prematurely. So, to
 * avoid these situations, we use a dedicated ref counter for ioctls which is
 * used in addition to and in parallel with the normal conn_ref count.
 */
#define CONN_INC_IOCTLREF_LOCKED(connp) {                       \
        ASSERT(MUTEX_HELD(&(connp)->conn_lock));                \
        DTRACE_PROBE1(conn__inc__ioctlref, conn_t *, (connp));  \
        (connp)->conn_ioctlref++;                               \
        mutex_exit(&(connp)->conn_lock);                        \
}

#define CONN_INC_IOCTLREF(connp)        {                       \
        mutex_enter(&(connp)->conn_lock);                       \
        CONN_INC_IOCTLREF_LOCKED(connp);                        \
```

```c
}

#define CONN_DEC_IOCTLREF(connp)        {                       \
        mutex_enter(&(connp)->conn_lock);                       \
        DTRACE_PROBE1(conn__dec__ioctlref, conn_t *, (connp));  \
        /* Make sure conn_ioctlref will not underflow. */       \
        ASSERT((connp)->conn_ioctlref != 0);                    \
        if ((--(connp)->conn_ioctlref == 0) &&                  \
            ((connp)->conn_state_flags & CONN_CLOSING)) {       \
                cv_broadcast(&(connp)->conn_cv);                \
        }                                                       \
        mutex_exit(&(connp)->conn_lock);                        \
}


/*
 * Complete the pending operation. Usually an ioctl. Can also
 * be a bind or option management request that got enqueued
 * in an ipsq_t. Called on completion of the operation.
 */
#define CONN_OPER_PENDING_DONE(connp)   {                       \
        mutex_enter(&(connp)->conn_lock);                       \
        (connp)->conn_oper_pending_ill = NULL;                  \
        cv_broadcast(&(connp)->conn_refcv);                     \
        mutex_exit(&(connp)->conn_lock);                        \
        CONN_DEC_REF(connp);                                    \
}

/*
 * Values for squeue switch:
 */
#define IP_SQUEUE_ENTER_NODRAIN 1
#define IP_SQUEUE_ENTER 2
#define IP_SQUEUE_FILL 3

extern int ip_squeue_flag;

/* IP Fragmentation Reassembly Header */
typedef struct ipf_s {
        struct ipf_s    *ipf_hash_next;
        struct ipf_s    **ipf_ptphn;    /* Pointer to previous hash next. */
        uint32_t        ipf_ident;      /* Ident to match. */
        uint8_t         ipf_protocol;   /* Protocol to match. */
        uchar_t         ipf_last_frag_seen : 1; /* Last fragment seen ? */
        time_t          ipf_timestamp;  /* Reassembly start time. */
        mblk_t          *ipf_mp;        /* mblk we live in. */
        mblk_t          *ipf_tail_mp;   /* Frag queue tail pointer. */
        int             ipf_hole_cnt;   /* Number of holes (hard-case). */
        int             ipf_end;        /* Tail end offset (0 -> hard-case). */
        uint_t          ipf_gen;        /* Frag queue generation */
        size_t          ipf_count;      /* Count of bytes used by frag */
        uint_t          ipf_nf_hdr_len; /* Length of nonfragmented header */
        in6_addr_t      ipf_v6src;      /* IPv6 source address */
        in6_addr_t      ipf_v6dst;      /* IPv6 dest address */
        uint_t          ipf_prev_nexthdr_offset; /* Offset for nexthdr value */
        uint8_t         ipf_ecn;        /* ECN info for the fragments */
        uint8_t         ipf_num_dups;   /* Number of times dup frags recvd */
        uint16_t        ipf_checksum_flags; /* Hardware checksum flags */
        uint32_t        ipf_checksum;   /* Partial checksum of fragment data */
} ipf_t;

/*
 * IPv4 Fragments
 */
#define IS_V4_FRAGMENT(ipha_fragment_offset_and_flags)                  \
        (((ntohs(ipha_fragment_offset_and_flags) & IPH_OFFSET) != 0) || \
```

```c
        ((ntohs(ipha_fragment_offset_and_flags) & IPH_MF) != 0))

#define ipf_src V4_PART_OF_V6(ipf_v6src)
#define ipf_dst V4_PART_OF_V6(ipf_v6dst)

#endif /* _KERNEL */

/* ICMP types */
#define ICMP_ECHO_REPLY                 0
#define ICMP_DEST_UNREACHABLE           3
#define ICMP_SOURCE_QUENCH              4
#define ICMP_REDIRECT                   5
#define ICMP_ECHO_REQUEST               8
#define ICMP_ROUTER_ADVERTISEMENT       9
#define ICMP_ROUTER_SOLICITATION        10
#define ICMP_TIME_EXCEEDED              11
#define ICMP_PARAM_PROBLEM              12
#define ICMP_TIME_STAMP_REQUEST         13
#define ICMP_TIME_STAMP_REPLY           14
#define ICMP_INFO_REQUEST               15
#define ICMP_INFO_REPLY                 16
#define ICMP_ADDRESS_MASK_REQUEST       17
#define ICMP_ADDRESS_MASK_REPLY         18

/* Evaluates to true if the ICMP type is an ICMP error */
#define ICMP_IS_ERROR(type)     (               \
        (type) == ICMP_DEST_UNREACHABLE ||      \
        (type) == ICMP_SOURCE_QUENCH ||         \
        (type) == ICMP_TIME_EXCEEDED ||         \
        (type) == ICMP_PARAM_PROBLEM)

/* ICMP_TIME_EXCEEDED codes */
#define ICMP_TTL_EXCEEDED               0
#define ICMP_REASSEMBLY_TIME_EXCEEDED   1

/* ICMP_DEST_UNREACHABLE codes */
#define ICMP_NET_UNREACHABLE            0
#define ICMP_HOST_UNREACHABLE           1
#define ICMP_PROTOCOL_UNREACHABLE       2
#define ICMP_PORT_UNREACHABLE           3
#define ICMP_FRAGMENTATION_NEEDED       4
#define ICMP_SOURCE_ROUTE_FAILED        5
#define ICMP_DEST_NET_UNKNOWN           6
#define ICMP_DEST_HOST_UNKNOWN          7
#define ICMP_SRC_HOST_ISOLATED          8
#define ICMP_DEST_NET_UNREACH_ADMIN     9
#define ICMP_DEST_HOST_UNREACH_ADMIN    10
#define ICMP_DEST_NET_UNREACH_TOS       11
#define ICMP_DEST_HOST_UNREACH_TOS      12

/* ICMP Header Structure */
typedef struct icmph_s {
        uint8_t         icmph_type;
        uint8_t         icmph_code;
        uint16_t        icmph_checksum;
        union {
                struct { /* ECHO request/response structure */
                        uint16_t        u_echo_ident;
                        uint16_t        u_echo_seqnum;
                } u_echo;
                struct { /* Destination unreachable structure */
                        uint16_t        u_du_zero;
                        uint16_t        u_du_mtu;
                } u_du;
                struct { /* Parameter problem structure */
                        uint8_t         u_pp_ptr;
                        uint8_t         u_pp_rsvd[3];
                } u_pp;
                struct { /* Redirect structure */
                        ipaddr_t        u_rd_gateway;
                } u_rd;
        } icmph_u;
} icmph_t;

#define icmph_echo_ident        icmph_u.u_echo.u_echo_ident
#define icmph_echo_seqnum       icmph_u.u_echo.u_echo_seqnum
#define icmph_du_zero           icmph_u.u_du.u_du_zero
#define icmph_du_mtu            icmph_u.u_du.u_du_mtu
#define icmph_pp_ptr            icmph_u.u_pp.u_pp_ptr
#define icmph_rd_gateway        icmph_u.u_rd.u_rd_gateway

#define ICMPH_SIZE      8

/*
 * Minimum length of transport layer header included in an ICMP error
 * message for it to be considered valid.
 */
#define ICMP_MIN_TP_HDR_LEN     8

/* Aligned IP header */
typedef struct ipha_s {
        uint8_t         ipha_version_and_hdr_length;
        uint8_t         ipha_type_of_service;
        uint16_t        ipha_length;
        uint16_t        ipha_ident;
        uint16_t        ipha_fragment_offset_and_flags;
        uint8_t         ipha_ttl;
        uint8_t         ipha_protocol;
        uint16_t        ipha_hdr_checksum;
        ipaddr_t        ipha_src;
        ipaddr_t        ipha_dst;
} ipha_t;

/*
 * IP Flags
 *
 * Some of these constant names are copied for the DTrace IP provider in
 * usr/src/lib/libdtrace/common/{ip.d.in, ip.sed.in}, which should be kept
 * in sync.
 */
#define IPH_DF          0x4000  /* Don't fragment */
#define IPH_MF          0x2000  /* More fragments to come */
#define IPH_OFFSET      0x1FFF  /* Where the offset lives */

/* Byte-order specific values */
#ifdef  _BIG_ENDIAN
#define IPH_DF_HTONS    0x4000  /* Don't fragment */
#define IPH_MF_HTONS    0x2000  /* More fragments to come */
#define IPH_OFFSET_HTONS 0x1FFF /* Where the offset lives */
#else
#define IPH_DF_HTONS    0x0040  /* Don't fragment */
#define IPH_MF_HTONS    0x0020  /* More fragments to come */
#define IPH_OFFSET_HTONS 0xFF1F /* Where the offset lives */
#endif

/* ECN code points for IPv4 TOS byte and IPv6 traffic class octet. */
#define IPH_ECN_NECT    0x0     /* Not ECN-Capable Transport */
#define IPH_ECN_ECT1    0x1     /* ECN-Capable Transport, ECT(1) */
#define IPH_ECN_ECT0    0x2     /* ECN-Capable Transport, ECT(0) */
#define IPH_ECN_CE      0x3     /* ECN-Congestion Experienced (CE) */

struct ill_s;
```

```
typedef void ip_v6intfid_func_t(struct ill_s *, in6_addr_t *);
typedef void ip_v6mapinfo_func_t(struct ill_s *, uchar_t *, uchar_t *);
typedef void ip_v4mapinfo_func_t(struct ill_s *, uchar_t *, uchar_t *);

/* IP Mac info structure */
typedef struct ip_m_s {
        t_uscalar_t             ip_m_mac_type;  /* From <sys/dlpi.h> */
        int                     ip_m_type;      /* From <net/if_types.h> */
        t_uscalar_t             ip_m_ipv4sap;
        t_uscalar_t             ip_m_ipv6sap;
        ip_v4mapinfo_func_t     *ip_m_v4mapping;
        ip_v6mapinfo_func_t     *ip_m_v6mapping;
        ip_v6intfid_func_t      *ip_m_v6intfid;
        ip_v6intfid_func_t      *ip_m_v6destintfid;
} ip_m_t;

/*
 * The following functions attempt to reduce the link layer dependency
 * of the IP stack. The current set of link specific operations are:
 * a. map from IPv4 class D (224.0/4) multicast address range or the
 * IPv6 multicast address range (ff00::/8) to the link layer multicast
 * address.
 * b. derive the default IPv6 interface identifier from the interface.
 * c. derive the default IPv6 destination interface identifier from
 * the interface (point-to-point only).
 */
extern  void ip_mcast_mapping(struct ill_s *, uchar_t *, uchar_t *);
/* ip_m_v6*intfid return void and are never NULL */
#define MEDIA_V6INTFID(ip_m, ill, v6ptr) (ip_m)->ip_m_v6intfid(ill, v6ptr)
#define MEDIA_V6DESTINTFID(ip_m, ill, v6ptr) \
        (ip_m)->ip_m_v6destintfid(ill, v6ptr)

/* Router entry types */
#define IRE_BROADCAST           0x0001  /* Route entry for broadcast address */
#define IRE_DEFAULT             0x0002  /* Route entry for default gateway */
#define IRE_LOCAL               0x0004  /* Route entry for local address */
#define IRE_LOOPBACK            0x0008  /* Route entry for loopback address */
#define IRE_PREFIX              0x0010  /* Route entry for prefix routes */
#ifndef _KERNEL
/* Keep so user-level still compiles */
#define IRE_CACHE               0x0020  /* Cached Route entry */
#endif
#define IRE_IF_NORESOLVER       0x0040  /* Route entry for local interface */
                                        /* net without any address mapping. */
#define IRE_IF_RESOLVER         0x0080  /* Route entry for local interface */
                                        /* net with resolver. */
#define IRE_HOST                0x0100  /* Host route entry */
/* Keep so user-level still compiles */
#define IRE_HOST_REDIRECT       0x0200  /* only used for T_SVR4_OPTMGMT_REQ */
#define IRE_IF_CLONE            0x0400  /* Per host clone of IRE_IF */
#define IRE_MULTICAST           0x0800  /* Special - not in table */
#define IRE_NOROUTE             0x1000  /* Special - not in table */

#define IRE_INTERFACE           (IRE_IF_NORESOLVER | IRE_IF_RESOLVER)

#define IRE_IF_ALL              (IRE_IF_NORESOLVER | IRE_IF_RESOLVER | \
                                    IRE_IF_CLONE)
#define IRE_OFFSUBNET           (IRE_DEFAULT | IRE_PREFIX | IRE_HOST)
#define IRE_OFFLINK             IRE_OFFSUBNET
/*
 * Note that we view IRE_NOROUTE as ONLINK since we can "send" to them without
 * going through a router; the result of sending will be an error/icmp error.
 */
#define IRE_ONLINK              (IRE_IF_ALL|IRE_LOCAL|IRE_LOOPBACK| \
                                    IRE_BROADCAST|IRE_MULTICAST|IRE_NOROUTE)
```

```
/* Arguments to ire_flush_cache() */
#define IRE_FLUSH_DELETE        0
#define IRE_FLUSH_ADD           1
#define IRE_FLUSH_GWCHANGE      2

/*
 * Flags to ire_route_recursive
 */
#define IRR_NONE                0
#define IRR_ALLOCATE            1       /* OK to allocate IRE_IF_CLONE */
#define IRR_INCOMPLETE          2       /* OK to return incomplete chain */

/*
 * Open/close synchronization flags.
 * These are kept in a separate field in the conn and the synchronization
 * depends on the atomic 32 bit access to that field.
 */
#define CONN_CLOSING            0x01    /* ip_close waiting for ip_wsrv */
#define CONN_CONDEMNED          0x02    /* conn is closing, no more refs */
#define CONN_INCIPIENT          0x04    /* conn not yet visible, no refs */
#define CONN_QUIESCED           0x08    /* conn is now quiescent */
#define CONN_UPDATE_ILL         0x10    /* conn_update_ill in progress */

/*
 * Flags for dce_flags field. Specifies which information has been set.
 * dce_ident is always present, but the other ones are identified by the flags.
 */
#define DCEF_DEFAULT            0x0001  /* Default DCE - no pmtu or uinfo */
#define DCEF_PMTU               0x0002  /* Different than interface MTU */
#define DCEF_UINFO              0x0004  /* dce_uinfo set */
#define DCEF_TOO_SMALL_PMTU     0x0008  /* Smaller than IPv4/IPv6 MIN */

#ifdef _KERNEL
/*
 * Extra structures need for per-src-addr filtering (IGMPv3/MLDv2)
 */
#define MAX_FILTER_SIZE 64

typedef struct slist_s {
        int             sl_numsrc;
        in6_addr_t      sl_addr[MAX_FILTER_SIZE];
} slist_t;

/*
 * Following struct is used to maintain retransmission state for
 * a multicast group.  One rtx_state_t struct is an in-line field
 * of the ilm_t struct; the slist_ts in the rtx_state_t struct are
 * alloc'd as needed.
 */
typedef struct rtx_state_s {
        uint_t          rtx_timer;      /* retrans timer */
        int             rtx_cnt;        /* retrans count */
        int             rtx_fmode_cnt;  /* retrans count for fmode change */
        slist_t         *rtx_allow;
        slist_t         *rtx_block;
} rtx_state_t;

/*
 * Used to construct list of multicast address records that will be
 * sent in a single listener report.
 */
typedef struct mrec_s {
        struct mrec_s   *mrec_next;
        uint8_t         mrec_type;
        uint8_t         mrec_auxlen;    /* currently unused */
```

```
        in6_addr_t      mrec_group;
        slist_t         mrec_srcs;
} mrec_t;

/* Group membership list per upper conn */

/*
 * We record the multicast information from the socket option in
 * ilg_ifaddr/ilg_ifindex. This allows rejoining the group in the case when
 * the ifaddr (or ifindex) disappears and later reappears, potentially on
 * a different ill. The IPv6 multicast socket options and ioctls all specify
 * the interface using an ifindex. For IPv4 some socket options/ioctls use
 * the interface address and others use the index. We record here the method
 * that was actually used (and leave the other of ilg_ifaddr or ilg_ifindex)
 * at zero so that we can rejoin the way the application intended.
 *
 * We track the ill on which we will or already have joined an ilm using
 * ilg_ill. When we have succeeded joining the ilm and have a refhold on it
 * then we set ilg_ilm. Thus intentionally there is a window where ilg_ill is
 * set and ilg_ilm is not set. This allows clearing ilg_ill as a signal that
 * the ill is being unplumbed and the ilm should be discarded.
 *
 * ilg records the state of multicast memberships of a socket end point.
 * ilm records the state of multicast memberships with the driver and is
 * maintained per interface.
 *
 * The ilg state is protected by conn_ilg_lock.
 * The ilg will not be freed until ilg_refcnt drops to zero.
 */
typedef struct ilg_s {
        struct ilg_s    *ilg_next;
        struct ilg_s    **ilg_ptpn;
        struct conn_s   *ilg_connp;     /* Back pointer to get lock */
        in6_addr_t      ilg_v6group;
        ipaddr_t        ilg_ifaddr;     /* For some IPv4 cases */
        uint_t          ilg_ifindex;    /* IPv6 and some other IPv4 cases */
        struct ill_s    *ilg_ill;       /* Where ilm is joined. No refhold */
        struct ilm_s    *ilg_ilm;       /* With ilm_refhold */
        uint_t          ilg_refcnt;
        mcast_record_t  ilg_fmode;      /* MODE_IS_INCLUDE/MODE_IS_EXCLUDE */
        slist_t         *ilg_filter;
        boolean_t       ilg_condemned;  /* Conceptually deleted */
} ilg_t;

/*
 * Multicast address list entry for ill.
 * ilm_ill is used by IPv4 and IPv6
 *
 * The ilm state (and other multicast state on the ill) is protected by
 * ill_mcast_lock. Operations that change state on both an ilg and ilm
 * in addition use ill_mcast_serializer to ensure that we can't have
 * interleaving between e.g., add and delete operations for the same conn_t,
 * group, and ill. The ill_mcast_serializer is also used to ensure that
 * multicast group joins do not occur on an interface that is in the process
 * of joining an IPMP group.
 *
 * The comment below (and for other netstack_t references) refers
 * to the fact that we only do netstack_hold in particular cases,
 * such as the references from open endpoints (ill_t and conn_t's
 * pointers). Internally within IP we rely on IP's ability to cleanup e.g.
 * ire_t's when an ill goes away.
 */
typedef struct ilm_s {
        in6_addr_t      ilm_v6addr;
        int             ilm_refcnt;
        uint_t          ilm_timer;      /* IGMP/MLD query resp timer, in msec */
        struct ilm_s    *ilm_next;      /* Linked list for each ill */
        uint_t          ilm_state;      /* state of the membership */
        struct ill_s    *ilm_ill;       /* Back pointer to ill - ill_ilm_cnt */
        zoneid_t        ilm_zoneid;
        int             ilm_no_ilg_cnt; /* number of joins w/ no ilg */
        mcast_record_t  ilm_fmode;      /* MODE_IS_INCLUDE/MODE_IS_EXCLUDE */
        slist_t         *ilm_filter;    /* source filter list */
        slist_t         *ilm_pendsrcs;  /* relevant src addrs for pending req */
        rtx_state_t     ilm_rtx;        /* SCR retransmission state */
        ipaddr_t        ilm_ifaddr;     /* For IPv4 netstat */
        ip_stack_t      *ilm_ipst;      /* Does not have a netstack_hold */
} ilm_t;

#define ilm_addr        V4_PART_OF_V6(ilm_v6addr)

/*
 * Soft reference to an IPsec SA.
 *
 * On relative terms, conn's can be persistent (living as long as the
 * processes which create them), while SA's are ephemeral (dying when
 * they hit their time-based or byte-based lifetimes).
 *
 * We could hold a hard reference to an SA from an ipsec_latch_t,
 * but this would cause expired SA's to linger for a potentially
 * unbounded time.
 *
 * Instead, we remember the hash bucket number and bucket generation
 * in addition to the pointer.  The bucket generation is incremented on
 * each deletion.
 */
typedef struct ipsa_ref_s
{
        struct ipsa_s   *ipsr_sa;
        struct isaf_s   *ipsr_bucket;
        uint64_t        ipsr_gen;
} ipsa_ref_t;

/*
 * IPsec "latching" state.
 *
 * In the presence of IPsec policy, fully-bound conn's bind a connection
 * to more than just the 5-tuple, but also a specific IPsec action and
 * identity-pair.
 * The identity pair is accessed from both the receive and transmit side
 * hence it is maintained in the ipsec_latch_t structure. conn_latch and
 * ixa_ipsec_latch points to it.
 * The policy and actions are stored in conn_latch_in_policy and
 * conn_latch_in_action for the inbound side, and in ixa_ipsec_policy and
 * ixa_ipsec_action for the transmit side.
 *
 * As an optimization, we also cache soft references to IPsec SA's in
 * ip_xmit_attr_t so that we can fast-path around most of the work needed for
 * outbound IPsec SA selection.
 */
typedef struct ipsec_latch_s
{
        kmutex_t        ipl_lock;
        uint32_t        ipl_refcnt;

        struct ipsid_s  *ipl_local_cid;
        struct ipsid_s  *ipl_remote_cid;
        unsigned int
                        ipl_ids_latched : 1,

                        ipl_pad_to_bit_31 : 31;
} ipsec_latch_t;
```

```
#define IPLATCH_REFHOLD(ipl) { \
        atomic_add_32(&(ipl)->ipl_refcnt, 1);           \
        ASSERT((ipl)->ipl_refcnt != 0);                 \
}

#define IPLATCH_REFRELE(ipl) {                          \
        ASSERT((ipl)->ipl_refcnt != 0);                         \
        membar_exit();                                          \
        if (atomic_add_32_nv(&(ipl)->ipl_refcnt, -1) == 0)      \
                iplatch_free(ipl);                              \
}

/*
 * peer identity structure.
 */
typedef struct conn_s conn_t;

/*
 * This is used to match an inbound/outbound datagram with policy.
 */
typedef struct ipsec_selector {
        in6_addr_t      ips_local_addr_v6;
        in6_addr_t      ips_remote_addr_v6;
        uint16_t        ips_local_port;
        uint16_t        ips_remote_port;
        uint8_t         ips_icmp_type;
        uint8_t         ips_icmp_code;
        uint8_t         ips_protocol;
        uint8_t         ips_isv4 : 1,
                        ips_is_icmp_inv_acq: 1;
} ipsec_selector_t;

/*
 * Note that we put v4 addresses in the *first* 32-bit word of the
 * selector rather than the last to simplify the prefix match/mask code
 * in spd.c
 */
#define ips_local_addr_v4 ips_local_addr_v6.s6_addr32[0]
#define ips_remote_addr_v4 ips_remote_addr_v6.s6_addr32[0]

/* Values used in IP by IPSEC Code */
#define         IPSEC_OUTBOUND          B_TRUE
#define         IPSEC_INBOUND           B_FALSE

/*
 * There are two variants in policy failures. The packet may come in
 * secure when not needed (IPSEC_POLICY_???_NOT_NEEDED) or it may not
 * have the desired level of protection (IPSEC_POLICY_MISMATCH).
 */
#define IPSEC_POLICY_NOT_NEEDED         0
#define IPSEC_POLICY_MISMATCH           1
#define IPSEC_POLICY_AUTH_NOT_NEEDED    2
#define IPSEC_POLICY_ENCR_NOT_NEEDED    3
#define IPSEC_POLICY_SE_NOT_NEEDED      4
#define IPSEC_POLICY_MAX                5       /* Always max + 1. */

/*
 * Check with IPSEC inbound policy if
 *
 * 1) per-socket policy is present - indicated by conn_in_enforce_policy.
 * 2) Or if we have not cached policy on the conn and the global policy is
 *    non-empty.
 */
#define CONN_INBOUND_POLICY_PRESENT(connp, ipss)        \
        ((connp)->conn_in_enforce_policy ||             \
        (!((connp)->conn_policy_cached) &&              \
        (ipss)->ipsec_inbound_v4_policy_present))

#define CONN_INBOUND_POLICY_PRESENT_V6(connp, ipss)     \
        ((connp)->conn_in_enforce_policy ||             \
        (!(connp)->conn_policy_cached &&                \
        (ipss)->ipsec_inbound_v6_policy_present))

#define CONN_OUTBOUND_POLICY_PRESENT(connp, ipss)       \
        ((connp)->conn_out_enforce_policy ||            \
        (!((connp)->conn_policy_cached) &&              \
        (ipss)->ipsec_outbound_v4_policy_present))

#define CONN_OUTBOUND_POLICY_PRESENT_V6(connp, ipss)    \
        ((connp)->conn_out_enforce_policy ||            \
        (!(connp)->conn_policy_cached &&                \
        (ipss)->ipsec_outbound_v6_policy_present))

/*
 * Information cached in IRE for upper layer protocol (ULP).
 */
typedef struct iulp_s {
        boolean_t       iulp_set;       /* Is any metric set? */
        uint32_t        iulp_ssthresh;  /* Slow start threshold (TCP). */
        clock_t         iulp_rtt;       /* Guestimate in millisecs. */
        clock_t         iulp_rtt_sd;    /* Cached value of RTT variance. */
        uint32_t        iulp_spipe;     /* Send pipe size. */
        uint32_t        iulp_rpipe;     /* Receive pipe size. */
        uint32_t        iulp_rtomax;    /* Max round trip timeout. */
        uint32_t        iulp_sack;      /* Use SACK option (TCP)? */
        uint32_t        iulp_mtu;       /* Setable with routing sockets */

        uint32_t
                iulp_tstamp_ok : 1,     /* Use timestamp option (TCP)? */
                iulp_wscale_ok : 1,     /* Use window scale option (TCP)? */
                iulp_ecn_ok : 1,        /* Enable ECN (for TCP)? */
                iulp_pmtud_ok : 1,      /* Enable PMTUd? */

                /* These three are passed out by ip_set_destination */
                iulp_localnet: 1,       /* IRE_ONLINK */
                iulp_loopback: 1,       /* IRE_LOOPBACK */
                iulp_local: 1,          /* IRE_LOCAL */

                iulp_not_used : 25;
} iulp_t;

/*
 * The conn drain list structure (idl_t), protected by idl_lock.  Each conn_t
 * inserted in the list points back at this idl_t using conn_idl, and is
 * chained by conn_drain_next and conn_drain_prev, which are also protected by
 * idl_lock.  When flow control is relieved, either ip_wsrv() (STREAMS) or
 * ill_flow_enable() (non-STREAMS) will call conn_drain().
 *
 * The conn drain list, idl_t, itself is part of tx cookie list structure.
 * A tx cookie list points to a blocked Tx ring and contains the list of
 * all conn's that are blocked due to the flow-controlled Tx ring (via
 * the idl drain list). Note that a link can have multiple Tx rings. The
 * drain list will store the conn's blocked due to Tx ring being flow
 * controlled.
 */

typedef uintptr_t ip_mac_tx_cookie_t;
typedef struct idl_s idl_t;
typedef struct idl_tx_list_s idl_tx_list_t;

struct idl_tx_list_s {
```

```
        ip_mac_tx_cookie_t      txl_cookie;
        kmutex_t                txl_lock;       /* Lock for this list */
        idl_t                   *txl_drain_list;
        int                     txl_drain_index;
};

struct idl_s {
        conn_t          *idl_conn;              /* Head of drain list */
        kmutex_t        idl_lock;               /* Lock for this list */
        idl_tx_list_t   *idl_itl;
};

/*
 * Interface route structure which holds the necessary information to recreate
 * routes that are tied to an interface i.e. have ire_ill set.
 *
 * These routes which were initially created via a routing socket or via the
 * SIOCADDRT ioctl may be gateway routes (RTF_GATEWAY being set) or may be
 * traditional interface routes.  When an ill comes back up after being
 * down, this information will be used to recreate the routes.  These
 * are part of an mblk_t chain that hangs off of the ILL (ill_saved_ire_mp).
 */
typedef struct ifrt_s {
        ushort_t        ifrt_type;              /* Type of IRE */
        in6_addr_t      ifrt_v6addr;            /* Address IRE represents. */
        in6_addr_t      ifrt_v6gateway_addr;    /* Gateway if IRE_OFFLINK */
        in6_addr_t      ifrt_v6setsrc_addr;     /* Src addr if RTF_SETSRC */
        in6_addr_t      ifrt_v6mask;            /* Mask for matching IRE. */
        uint32_t        ifrt_flags;             /* flags related to route */
        iulp_t          ifrt_metrics;           /* Routing socket metrics */
        zoneid_t        ifrt_zoneid;            /* zoneid for route */
} ifrt_t;

#define ifrt_addr               V4_PART_OF_V6(ifrt_v6addr)
#define ifrt_gateway_addr       V4_PART_OF_V6(ifrt_v6gateway_addr)
#define ifrt_mask               V4_PART_OF_V6(ifrt_v6mask)
#define ifrt_setsrc_addr        V4_PART_OF_V6(ifrt_v6setsrc_addr)

/* Number of IP addresses that can be hosted on a physical interface */
#define MAX_ADDRS_PER_IF        8192
/*
 * Number of Source addresses to be considered for source address
 * selection. Used by ipif_select_source_v4/v6.
 */
#define MAX_IPIF_SELECT_SOURCE  50

#ifdef IP_DEBUG
/*
 * Trace refholds and refreles for debugging.
 */
#define TR_STACK_DEPTH  14
typedef struct tr_buf_s {
        int     tr_depth;
        clock_t tr_time;
        pc_t    tr_stack[TR_STACK_DEPTH];
} tr_buf_t;

typedef struct th_trace_s {
        int             th_refcnt;
        uint_t          th_trace_lastref;
        kthread_t       *th_id;
#define TR_BUF_MAX      38
        tr_buf_t        th_trbuf[TR_BUF_MAX];
} th_trace_t;

typedef struct th_hash_s {
        list_node_t     thh_link;
        mod_hash_t      *thh_hash;
        ip_stack_t      *thh_ipst;
} th_hash_t;
#endif

/* The following are ipif_state_flags */
#define IPIF_CONDEMNED          0x1     /* The ipif is being removed */
#define IPIF_CHANGING           0x2     /* A critcal ipif field is changing */
#define IPIF_SET_LINKLOCAL      0x10    /* transient flag during bringup */

/* IP interface structure, one per local address */
typedef struct ipif_s {
        struct  ipif_s  *ipif_next;
        struct  ill_s   *ipif_ill;      /* Back pointer to our ill */
        int     ipif_id;                /* Logical unit number */
        in6_addr_t ipif_v6lcl_addr;     /* Local IP address for this if. */
        in6_addr_t ipif_v6subnet;       /* Subnet prefix for this if. */
        in6_addr_t ipif_v6net_mask;     /* Net mask for this interface. */
        in6_addr_t ipif_v6brd_addr;     /* Broadcast addr for this interface. */
        in6_addr_t ipif_v6pp_dst_addr;  /* Point-to-point dest address. */
        uint64_t ipif_flags;            /* Interface flags. */
        uint_t  ipif_ire_type;          /* IRE_LOCAL or IRE_LOOPBACK */

        /*
         * The packet count in the ipif contain the sum of the
         * packet counts in dead IRE_LOCAL/LOOPBACK for this ipif.
         */
        uint_t  ipif_ib_pkt_count;      /* Inbound packets for our dead IREs */

        /* Exclusive bit fields, protected by ipsq_t */
        unsigned int
                ipif_was_up : 1,        /* ipif was up before */
                ipif_addr_ready : 1,    /* DAD is done */
                ipif_was_dup : 1,       /* DAD had failed */
                ipif_added_nce : 1,     /* nce added for local address */

                ipif_pad_to_31 : 28;

        ilm_t   *ipif_allhosts_ilm;     /* For all-nodes join */
        ilm_t   *ipif_solmulti_ilm;     /* For IPv6 solicited multicast join */

        uint_t  ipif_seqid;             /* unique index across all ills */
        uint_t  ipif_state_flags;       /* See IPIF_* flag defs above */
        uint_t  ipif_refcnt;            /* active consistent reader cnt */

        zoneid_t ipif_zoneid;           /* zone ID number */
        timeout_id_t ipif_recovery_id;  /* Timer for DAD recovery */
        boolean_t ipif_trace_disable;   /* True when alloc fails */
        /*
         * For an IPMP interface, ipif_bound_ill tracks the ill whose hardware
         * information this ipif is associated with via ARP/NDP.  We can use
         * an ill pointer (rather than an index) because only ills that are
         * part of a group will be pointed to, and an ill cannot disappear
         * while it's in a group.
         */
        struct ill_s    *ipif_bound_ill;
        struct ipif_s   *ipif_bound_next; /* bound ipif chain */
        boolean_t       ipif_bound;     /* B_TRUE if we successfully bound */

        struct ire_s    *ipif_ire_local; /* Our IRE_LOCAL or LOOPBACK */
        struct ire_s    *ipif_ire_if;    /* Our IRE_INTERFACE */
} ipif_t;

/*
 * The following table lists the protection levels of the various members
```

```
 * of the ipif_t. The following notation is used.
 *
 * Write once - Written to only once at the time of bringing up
 * the interface and can be safely read after the bringup without any lock.
 *
 * ipsq - Need to execute in the ipsq to perform the indicated access.
 *
 * ill_lock - Need to hold this mutex to perform the indicated access.
 *
 * ill_g_lock - Need to hold this rw lock as reader/writer for read access or
 * write access respectively.
 *
 * down ill - Written to only when the ill is down (i.e all ipifs are down)
 * up ill - Read only when the ill is up (i.e. at least 1 ipif is up)
 *
 *                Table of ipif_t members and their protection
 *
 * ipif_next              ipsq + ill_lock +       ipsq OR ill_lock OR
 *                        ill_g_lock              ill_g_lock
 * ipif_ill               ipsq + down ipif        write once
 * ipif_id                ipsq + down ipif        write once
 * ipif_v6lcl_addr        ipsq + down ipif        up ipif
 * ipif_v6subnet          ipsq + down ipif        up ipif
 * ipif_v6net_mask        ipsq + down ipif        up ipif
 *
 * ipif_v6brd_addr
 * ipif_v6pp_dst_addr
 * ipif_flags             ill_lock                ill_lock
 * ipif_ire_type          ipsq + down ill         up ill
 *
 * ipif_ib_pkt_count      Approx
 *
 * bit fields             ill_lock                ill_lock
 *
 * ipif_allhosts_ilm      ipsq                    ipsq
 * ipif_solmulti_ilm      ipsq                    ipsq
 *
 * ipif_seqid             ipsq                    Write once
 *
 * ipif_state_flags       ill_lock                ill_lock
 * ipif_refcnt            ill_lock                ill_lock
 * ipif_bound_ill         ipsq + ipmp_lock        ipsq OR ipmp_lock
 * ipif_bound_next        ipsq                    ipsq
 * ipif_bound             ipsq                    ipsq
 *
 * ipif_ire_local         ipsq + ips_ill_g_lock   ipsq OR ips_ill_g_lock
 * ipif_ire_if            ipsq + ips_ill_g_lock   ipsq OR ips_ill_g_lock
 */

/*
 * Return values from ip_laddr_verify_{v4,v6}
 */
typedef enum { IPVL_UNICAST_UP, IPVL_UNICAST_DOWN, IPVL_MCAST, IPVL_BCAST,
            IPVL_BAD} ip_laddr_t;


#define IP_TR_HASH(tid) ((((uintptr_t)tid) >> 6) & (IP_TR_HASH_MAX - 1))

#ifdef DEBUG
#define IPIF_TRACE_REF(ipif)    ipif_trace_ref(ipif)
#define ILL_TRACE_REF(ill)      ill_trace_ref(ill)
#define IPIF_UNTRACE_REF(ipif)  ipif_untrace_ref(ipif)
#define ILL_UNTRACE_REF(ill)    ill_untrace_ref(ill)
#else
#define IPIF_TRACE_REF(ipif)
#define ILL_TRACE_REF(ill)
```

```
#define IPIF_UNTRACE_REF(ipif)
#define ILL_UNTRACE_REF(ill)
#endif

/* IPv4 compatibility macros */
#define ipif_lcl_addr           V4_PART_OF_V6(ipif_v6lcl_addr)
#define ipif_subnet             V4_PART_OF_V6(ipif_v6subnet)
#define ipif_net_mask           V4_PART_OF_V6(ipif_v6net_mask)
#define ipif_brd_addr           V4_PART_OF_V6(ipif_v6brd_addr)
#define ipif_pp_dst_addr        V4_PART_OF_V6(ipif_v6pp_dst_addr)

/* Macros for easy backreferences to the ill. */
#define ipif_isv6               ipif_ill->ill_isv6

#define SIOCLIFADDR_NDX 112     /* ndx of SIOCLIFADDR in the ndx ioctl table */

/*
 * mode value for ip_ioctl_finish for finishing an ioctl
 */
#define CONN_CLOSE      1               /* No mi_copy */
#define COPYOUT         2               /* do an mi_copyout if needed */
#define NO_COPYOUT      3               /* do an mi_copy_done */
#define IPI2MODE(ipi)   ((ipi)->ipi_flags & IPI_GET_CMD ? COPYOUT : NO_COPYOUT)

/*
 * The IP-MT design revolves around the serialization objects ipsq_t (IPSQ)
 * and ipxop_t (exclusive operation or "xop").  Becoming "writer" on an IPSQ
 * ensures that no other threads can become "writer" on any IPSQs sharing that
 * IPSQ's xop until the writer thread is done.
 *
 * Each phyint points to one IPSQ that remains fixed over the phyint's life.
 * Each IPSQ points to one xop that can change over the IPSQ's life.  If a
 * phyint is *not* in an IPMP group, then its IPSQ will refer to the IPSQ's
 * "own" xop (ipsq_ownxop).  If a phyint *is* part of an IPMP group, then its
 * IPSQ will refer to the "group" xop, which is shorthand for the xop of the
 * IPSQ of the IPMP meta-interface's phyint.  Thus, all phyints that are part
 * of the same IPMP group will have their IPSQ's point to the group xop, and
 * thus becoming "writer" on any phyint in the group will prevent any other
 * writer on any other phyint in the group.  All IPSQs sharing the same xop
 * are chained together through ipsq_next (in the degenerate common case,
 * ipsq_next simply refers to itself).  Note that the group xop is guaranteed
 * to exist at least as long as there are members in the group, since the IPMP
 * meta-interface can only be destroyed if the group is empty.
 *
 * Incoming exclusive operation requests are enqueued on the IPSQ they arrived
 * on rather than the xop.  This makes switching xop's (as would happen when a
 * phyint leaves an IPMP group) simple, because after the phyint leaves the
 * group, any operations enqueued on its IPSQ can be safely processed with
 * respect to its new xop, and any operations enqueued on the IPSQs of its
 * former group can be processed with respect to their existing group xop.
 * Even so, switching xops is a subtle dance; see ipsq_dq() for details.
 *
 * An IPSQ's "own" xop is embedded within the IPSQ itself since they have have
 * identical lifetimes, and because doing so simplifies pointer management.
 * While each phyint and IPSQ point to each other, it is not possible to free
 * the IPSQ when the phyint is freed, since we may still *inside* the IPSQ
 * when the phyint is being freed.  Thus, ipsq_phyint is set to NULL when the
 * phyint is freed, and the IPSQ free is later done in ipsq_exit().
 *
 * ipsq_t synchronization:      read                    write
 *
 *      ipsq_xopq_mphead        ipx_lock                ipx_lock
 *      ipsq_xopq_mptail        ipx_lock                ipx_lock
 *      ipsq_xop_switch_mp      ipsq_lock               ipsq_lock
 *      ipsq_phyint             write once              write once
 *      ipsq_next               RW_READER ill_g_lock    RW_WRITER ill_g_lock
```

```
 *      ipsq_xop                ipsq_lock or ipsq       ipsq_lock + ipsq
 *      ipsq_swxop              ipsq                    ipsq
 *      ipsq_ownxop             see ipxop_t             see ipxop_t
 *      ipsq_ipst               write once              write once
 *
 * ipxop_t synchronization:     read                    write
 *
 *      ipx_writer              ipx_lock                ipx_lock
 *      ipx_xop_queued          ipx_lock                ipx_lock
 *      ipx_mphead              ipx_lock                ipx_lock
 *      ipx_mptail              ipx_lock                ipx_lock
 *      ipx_ipsq                write once              write once
 *      ips_ipsq_queued         ipx_lock                ipx_lock
 *      ipx_waitfor             ipsq or ipx_lock        ipsq + ipx_lock
 *      ipx_reentry_cnt         ipsq or ipx_lock        ipsq + ipx_lock
 *      ipx_current_done        ipsq                    ipsq
 *      ipx_current_ioctl       ipsq                    ipsq
 *      ipx_current_ipif        ipsq or ipx_lock        ipsq + ipx_lock
 *      ipx_pending_ipif        ipsq or ipx_lock        ipsq + ipx_lock
 *      ipx_pending_mp          ipsq or ipx_lock        ipsq + ipx_lock
 *      ipx_forced              ipsq                    ipsq
 *      ipx_depth               ipsq                    ipsq
 *      ipx_stack               ipsq                    ipsq
 */
typedef struct ipxop_s {
        kmutex_t        ipx_lock;       /* see above */
        kthread_t       *ipx_writer;    /* current owner */
        mblk_t          *ipx_mphead;    /* messages tied to this op */
        mblk_t          *ipx_mptail;
        struct ipsq_s   *ipx_ipsq;      /* associated ipsq */
        boolean_t       ipx_ipsq_queued; /* ipsq using xop has queued op */
        int             ipx_waitfor;    /* waiting; values encoded below */
        int             ipx_reentry_cnt;
        boolean_t       ipx_current_done;  /* is the current operation done? */
        int             ipx_current_ioctl; /* current ioctl, or 0 if no ioctl */
        ipif_t          *ipx_current_ipif; /* ipif for current op */
        ipif_t          *ipx_pending_ipif; /* ipif for ipx_pending_mp */
        mblk_t          *ipx_pending_mp;   /* current ioctl mp while waiting */
        boolean_t       ipx_forced;                     /* debugging aid */
#ifdef DEBUG
        int             ipx_depth;                      /* debugging aid */
#define IPX_STACK_DEPTH 15
        pc_t            ipx_stack[IPX_STACK_DEPTH];     /* debugging aid */
#endif
} ipxop_t;

typedef struct ipsq_s {
        kmutex_t ipsq_lock;             /* see above */
        mblk_t  *ipsq_switch_mp;        /* op to handle right after switch */
        mblk_t  *ipsq_xopq_mphead;      /* list of excl ops (mostly ioctls) */
        mblk_t  *ipsq_xopq_mptail;
        struct phyint   *ipsq_phyint;   /* associated phyint */
        struct ipsq_s   *ipsq_next;     /* next ipsq sharing ipsq_xop */
        struct ipxop_s  *ipsq_xop;      /* current xop synchronization info */
        struct ipxop_s  *ipsq_swxop;    /* switch xop to on ipsq_exit() */
        struct ipxop_s  ipsq_ownxop;    /* our own xop (may not be in-use) */
        ip_stack_t      *ipsq_ipst;     /* does not have a netstack_hold */
} ipsq_t;

/*
 * ipx_waitfor values:
 */
enum {
        IPIF_DOWN = 1,  /* ipif_down() waiting for refcnts to drop */
        ILL_DOWN,       /* ill_down() waiting for refcnts to drop */
        IPIF_FREE,      /* ipif_free() waiting for refcnts to drop */
	ILL_FREE	/* ill unplumb waiting for refcnts to drop */
};

/* Operation types for ipsq_try_enter() */
#define CUR_OP 0	/* request writer within current operation */
#define NEW_OP 1	/* request writer for a new operation */
#define SWITCH_OP 2	/* request writer once IPSQ XOP switches */

/*
 * Kstats tracked on each IPMP meta-interface.  Order here must match
 * ipmp_kstats[] in ip/ipmp.c.
 */
enum {
	IPMP_KSTAT_OBYTES,	IPMP_KSTAT_OBYTES64,	IPMP_KSTAT_RBYTES,
	IPMP_KSTAT_RBYTES64,	IPMP_KSTAT_OPACKETS,	IPMP_KSTAT_OPACKETS64,
	IPMP_KSTAT_OERRORS,	IPMP_KSTAT_IPACKETS,	IPMP_KSTAT_IPACKETS64,
	IPMP_KSTAT_IERRORS,	IPMP_KSTAT_MULTIRCV,	IPMP_KSTAT_MULTIXMT,
	IPMP_KSTAT_BRDCSTRCV,	IPMP_KSTAT_BRDCSTXMT,	IPMP_KSTAT_LINK_UP,
	IPMP_KSTAT_MAX		/* keep last */
};

/*
 * phyint represents state that is common to both IPv4 and IPv6 interfaces.
 * There is a separate ill_t representing IPv4 and IPv6 which has a
 * backpointer to the phyint structure for accessing common state.
 */
typedef struct phyint {
	struct ill_s	*phyint_illv4;
	struct ill_s	*phyint_illv6;
	uint_t		phyint_ifindex;		/* SIOCSLIFINDEX */
	uint64_t	phyint_flags;
	avl_node_t	phyint_avl_by_index;	/* avl tree by index */
	avl_node_t	phyint_avl_by_name;	/* avl tree by name */
	kmutex_t	phyint_lock;
	struct ipsq_s	*phyint_ipsq;		/* back pointer to ipsq */
	struct ipmp_grp_s *phyint_grp;		/* associated IPMP group */
	char		phyint_name[LIFNAMSIZ]; /* physical interface name */
	uint64_t	phyint_kstats0[IPMP_KSTAT_MAX]; /* baseline kstats */
} phyint_t;

#define CACHE_ALIGN_SIZE 64
#define CACHE_ALIGN(align_struct)	P2ROUNDUP(sizeof (struct align_struct),\
							CACHE_ALIGN_SIZE)
struct _phyint_list_s_ {
	avl_tree_t	phyint_list_avl_by_index;	/* avl tree by index */
	avl_tree_t	phyint_list_avl_by_name;	/* avl tree by name */
};

typedef union phyint_list_u {
	struct	_phyint_list_s_ phyint_list_s;
	char	phyint_list_filler[CACHE_ALIGN(_phyint_list_s_)];
} phyint_list_t;

#define phyint_list_avl_by_index	phyint_list_s.phyint_list_avl_by_index
#define phyint_list_avl_by_name		phyint_list_s.phyint_list_avl_by_name

/*
 * Fragmentation hash bucket
 */
typedef struct ipfb_s {
	struct ipf_s	*ipfb_ipf;	/* List of ... */
	size_t		ipfb_count;	/* Count of bytes used by frag(s) */
	kmutex_t	ipfb_lock;	/* Protect all ipf in list */
	uint_t		ipfb_frag_pkts; /* num of distinct fragmented pkts */
} ipfb_t;
```

```
/*
 * IRE bucket structure. Usually there is an array of such structures,
 * each pointing to a linked list of ires. irb_refcnt counts the number
 * of walkers of a given hash bucket. Usually the reference count is
 * bumped up if the walker wants no IRES to be DELETED while walking the
 * list. Bumping up does not PREVENT ADDITION. This allows walking a given
 * hash bucket without stumbling up on a free pointer.
 *
 * irb_t structures in ip_ftable are dynamically allocated and freed.
 * In order to identify the irb_t structures that can be safely kmem_free'd
 * we need to ensure that
 *  - the irb_refcnt is quiescent, indicating no other walkers,
 *  - no other threads or ire's are holding references to the irb,
 *	i.e., irb_nire == 0,
 *  - there are no active ire's in the bucket, i.e., irb_ire_cnt == 0
 */
typedef struct irb {
	struct ire_s	*irb_ire;	/* First ire in this bucket */
					/* Should be first in this struct */
	krwlock_t	irb_lock;	/* Protect this bucket */
	uint_t		irb_refcnt;	/* Protected by irb_lock */
	uchar_t		irb_marks;	/* CONDEMNED ires in this bucket ? */
#define IRB_MARK_CONDEMNED	0x0001	/* Contains some IRE_IS_CONDEMNED */
#define IRB_MARK_DYNAMIC	0x0002	/* Dynamically allocated */
	/* Once IPv6 uses radix then IRB_MARK_DYNAMIC will be always be set */
	uint_t		irb_ire_cnt;	/* Num of active IRE in this bucket */
	int		irb_nire;	/* Num of ftable ire's that ref irb */
	ip_stack_t	*irb_ipst;	/* Does not have a netstack_hold */
} irb_t;

/*
 * This is the structure used to store the multicast physical addresses
 * that an interface has joined.
 * The refcnt keeps track of the number of multicast IP addresses mapping
 * to a physical multicast address.
 */
typedef struct multiphysaddr_s {
	struct	multiphysaddr_s	*mpa_next;
	char	mpa_addr[IP_MAX_HW_LEN];
	int	mpa_refcnt;
} multiphysaddr_t;

#define IRB2RT(irb)	(rt_t *)((caddr_t)(irb) - offsetof(rt_t, rt_irb))

/* Forward declarations */
struct dce_s;
typedef struct dce_s dce_t;
struct ire_s;
typedef struct ire_s ire_t;
struct ncec_s;
typedef struct ncec_s ncec_t;
struct nce_s;
typedef struct nce_s nce_t;
struct ip_recv_attr_s;
typedef struct ip_recv_attr_s ip_recv_attr_t;
struct ip_xmit_attr_s;
typedef struct ip_xmit_attr_s ip_xmit_attr_t;

struct tsol_ire_gw_secattr_s;
typedef struct tsol_ire_gw_secattr_s tsol_ire_gw_secattr_t;

/*
 * This is a structure for a one-element route cache that is passed
 * by reference between ip_input and ill_inputfn.
 */
typedef struct {
        ire_t           *rtc_ire;
        ipaddr_t        rtc_ipaddr;
        in6_addr_t      rtc_ip6addr;
} rtc_t;

/*
 * Note: Temporarily use 64 bits, and will probably go back to 32 bits after
 * more cleanup work is done.
 */
typedef uint64_t iaflags_t;

/* The ill input function pointer type */
typedef void (*pfillinput_t)(mblk_t *, void *, void *, ip_recv_attr_t *,
    rtc_t *);

/* The ire receive function pointer type */
typedef void (*pfirerecv_t)(ire_t *, mblk_t *, void *, ip_recv_attr_t *);

/* The ire send and postfrag function pointer types */
typedef int (*pfiresend_t)(ire_t *, mblk_t *, void *,
    ip_xmit_attr_t *, uint32_t *);
typedef int (*pfirepostfrag_t)(mblk_t *, nce_t *, iaflags_t, uint_t, uint32_t,
    zoneid_t, zoneid_t, uintptr_t *);


#define IP_V4_G_HEAD    0
#define IP_V6_G_HEAD    1

#define MAX_G_HEADS     2

/*
 * unpadded ill_if structure
 */
struct  _ill_if_s_ {
        union ill_if_u  *illif_next;
        union ill_if_u  *illif_prev;
        avl_tree_t      illif_avl_by_ppa;       /* AVL tree sorted on ppa */
        vmem_t          *illif_ppa_arena;       /* ppa index space */
        uint16_t        illif_mcast_v1;         /* hints for              */
        uint16_t        illif_mcast_v2;         /* [igmp|mld]_slowtimo    */
        int             illif_name_len;         /* name length */
        char            illif_name[LIFNAMSIZ];  /* name of interface type */
};

/* cache aligned ill_if structure */
typedef union   ill_if_u {
        struct  _ill_if_s_ ill_if_s;
        char    illif_filler[CACHE_ALIGN(_ill_if_s_)];
} ill_if_t;

#define illif_next              ill_if_s.illif_next
#define illif_prev              ill_if_s.illif_prev
#define illif_avl_by_ppa        ill_if_s.illif_avl_by_ppa
#define illif_ppa_arena         ill_if_s.illif_ppa_arena
#define illif_mcast_v1          ill_if_s.illif_mcast_v1
#define illif_mcast_v2          ill_if_s.illif_mcast_v2
#define illif_name              ill_if_s.illif_name
#define illif_name_len          ill_if_s.illif_name_len

typedef struct ill_walk_context_s {
        int     ctx_current_list; /* current list being searched */
        int     ctx_last_list;   /* last list to search */
} ill_walk_context_t;

/*
 * ill_g_heads structure, one for IPV4 and one for IPV6
```

```
 */
struct _ill_g_head_s_ {
        ill_if_t        *ill_g_list_head;
        ill_if_t        *ill_g_list_tail;
};

typedef union ill_g_head_u {
        struct _ill_g_head_s_ ill_g_head_s;
        char    ill_g_head_filler[CACHE_ALIGN(_ill_g_head_s_)];
} ill_g_head_t;

#define ill_g_list_head ill_g_head_s.ill_g_list_head
#define ill_g_list_tail ill_g_head_s.ill_g_list_tail

#define IP_V4_ILL_G_LIST(ipst)  \
        (ipst)->ips_ill_g_heads[IP_V4_G_HEAD].ill_g_list_head
#define IP_V6_ILL_G_LIST(ipst)  \
        (ipst)->ips_ill_g_heads[IP_V6_G_HEAD].ill_g_list_head
#define IP_VX_ILL_G_LIST(i, ipst)       \
        (ipst)->ips_ill_g_heads[i].ill_g_list_head

#define ILL_START_WALK_V4(ctx_ptr, ipst)        \
        ill_first(IP_V4_G_HEAD, IP_V4_G_HEAD, ctx_ptr, ipst)
#define ILL_START_WALK_V6(ctx_ptr, ipst)        \
        ill_first(IP_V6_G_HEAD, IP_V6_G_HEAD, ctx_ptr, ipst)
#define ILL_START_WALK_ALL(ctx_ptr, ipst)       \
        ill_first(MAX_G_HEADS, MAX_G_HEADS, ctx_ptr, ipst)

/*
 * Capabilities, possible flags for ill_capabilities.
 */
#define ILL_CAPAB_LSO           0x04            /* Large Send Offload */
#define ILL_CAPAB_HCKSUM        0x08            /* Hardware checksumming */
#define ILL_CAPAB_ZEROCOPY      0x10            /* Zero-copy */
#define ILL_CAPAB_DLD           0x20            /* DLD capabilities */
#define ILL_CAPAB_DLD_POLL      0x40            /* Polling */
#define ILL_CAPAB_DLD_DIRECT    0x80            /* Direct function call */

/*
 * Per-ill Hardware Checksumming capbilities.
 */
typedef struct ill_hcksum_capab_s ill_hcksum_capab_t;

/*
 * Per-ill Zero-copy capabilities.
 */
typedef struct ill_zerocopy_capab_s ill_zerocopy_capab_t;

/*
 * DLD capbilities.
 */
typedef struct ill_dld_capab_s ill_dld_capab_t;

/*
 * Per-ill polling resource map.
 */
typedef struct ill_rx_ring ill_rx_ring_t;

/*
 * Per-ill Large Send Offload capabilities.
 */
typedef struct ill_lso_capab_s ill_lso_capab_t;

/* The following are ill_state_flags */
#define ILL_LL_SUBNET_PENDING   0x01    /* Waiting for DL_INFO_ACK from drv */
#define ILL_CONDEMNED           0x02    /* No more new ref's to the ILL */
```

```
#define ILL_DL_UNBIND_IN_PROGRESS       0x04    /* UNBIND_REQ is sent */
/*
 * ILL_DOWN_IN_PROGRESS is set to ensure the following:
 * - no packets are sent to the driver after the DL_UNBIND_REQ is sent,
 * - no longstanding references will be acquired on objects that are being
 *   brought down.
 */
#define ILL_DOWN_IN_PROGRESS    0x08

/* Is this an ILL whose source address is used by other ILL's ? */
#define IS_USESRC_ILL(ill)                      \
        (((ill)->ill_usesrc_ifindex == 0) &&    \
        ((ill)->ill_usesrc_grp_next != NULL))

/* Is this a client/consumer of the usesrc ILL ? */
#define IS_USESRC_CLI_ILL(ill)                  \
        (((ill)->ill_usesrc_ifindex != 0) &&    \
        ((ill)->ill_usesrc_grp_next != NULL))

/* Is this an virtual network interface (vni) ILL ? */
#define IS_VNI(ill)                                                     \
        (((ill)->ill_phyint->phyint_flags & (PHYI_LOOPBACK|PHYI_VIRTUAL)) == \
        PHYI_VIRTUAL)

/* Is this a loopback ILL? */
#define IS_LOOPBACK(ill) \
        ((ill)->ill_phyint->phyint_flags & PHYI_LOOPBACK)

/* Is this an IPMP meta-interface ILL? */
#define IS_IPMP(ill)                                                    \
        ((ill)->ill_phyint->phyint_flags & PHYI_IPMP)

/* Is this ILL under an IPMP meta-interface? (aka "in a group?") */
#define IS_UNDER_IPMP(ill)                                              \
        ((ill)->ill_grp != NULL && !IS_IPMP(ill))

/* Is ill1 in the same illgrp as ill2? */
#define IS_IN_SAME_ILLGRP(ill1, ill2)                                   \
        ((ill1)->ill_grp != NULL && ((ill1)->ill_grp == (ill2)->ill_grp))

/* Is ill1 on the same LAN as ill2? */
#define IS_ON_SAME_LAN(ill1, ill2)                                      \
        ((ill1) == (ill2) || IS_IN_SAME_ILLGRP(ill1, ill2))

#define ILL_OTHER(ill)                                                  \
        ((ill)->ill_isv6 ? (ill)->ill_phyint->phyint_illv4 :            \
            (ill)->ill_phyint->phyint_illv6)

/*
 * IPMP group ILL state structure -- up to two per IPMP group (V4 and V6).
 * Created when the V4 and/or V6 IPMP meta-interface is I_PLINK'd.  It is
 * guaranteed to persist while there are interfaces of that type in the group.
 * In general, most fields are accessed outside of the IPSQ (e.g., in the
 * datapath), and thus use locks in addition to the IPSQ for protection.
 *
 * synchronization:             read                    write
 *
 *      ig_if                   ipsq or ill_g_lock      ipsq and ill_g_lock
 *      ig_actif                ipsq or ipmp_lock       ipsq and ipmp_lock
 *      ig_nactif               ipsq or ipmp_lock       ipsq and ipmp_lock
 *      ig_next_ill             ipsq or ipmp_lock       ipsq and ipmp_lock
 *      ig_ipmp_ill             write once              write once
 *      ig_cast_ill             ipsq or ipmp_lock       ipsq and ipmp_lock
 *      ig_arpent               ipsq                    ipsq
 *      ig_mtu                  ipsq                    ipsq
 *      ig_mc_mtu               ipsq                    ipsq
```

```
 */
typedef struct ipmp_illgrp_s {
        list_t          ig_if;          /* list of all interfaces */
        list_t          ig_actif;       /* list of active interfaces */
        uint_t          ig_nactif;      /* number of active interfaces */
        struct ill_s    *ig_next_ill;   /* next active interface to use */
        struct ill_s    *ig_ipmp_ill;   /* backpointer to IPMP meta-interface */
        struct ill_s    *ig_cast_ill;   /* nominated ill for multi/broadcast */
        list_t          ig_arpent;      /* list of ARP entries */
        uint_t          ig_mtu;         /* ig_ipmp_ill->ill_mtu */
        uint_t          ig_mc_mtu;      /* ig_ipmp_ill->ill_mc_mtu */
} ipmp_illgrp_t;

/*
 * IPMP group state structure -- one per IPMP group.  Created when the
 * IPMP meta-interface is plumbed; it is guaranteed to persist while there
 * are interfaces in it.
 *
 * ipmp_grp_t synchronization:          read                    write
 *
 *      gr_name                         ipmp_lock               ipmp_lock
 *      gr_ifname                       write once              write once
 *      gr_mactype                      ipmp_lock               ipmp_lock
 *      gr_phyint                       write once              write once
 *      gr_nif                          ipmp_lock               ipmp_lock
 *      gr_nactif                       ipsq                    ipsq
 *      gr_v4                           ipmp_lock               ipmp_lock
 *      gr_v6                           ipmp_lock               ipmp_lock
 *      gr_nv4                          ipmp_lock               ipmp_lock
 *      gr_nv6                          ipmp_lock               ipmp_lock
 *      gr_pendv4                       ipmp_lock               ipmp_lock
 *      gr_pendv6                       ipmp_lock               ipmp_lock
 *      gr_linkdownmp                   ipsq                    ipsq
 *      gr_ksp                          ipmp_lock               ipmp_lock
 *      gr_kstats0                      atomic                  atomic
 */
typedef struct ipmp_grp_s {
        char            gr_name[LIFGRNAMSIZ];   /* group name */
        char            gr_ifname[LIFNAMSIZ];   /* interface name */
        t_uscalar_t     gr_mactype;     /* DLPI mactype of group */
        phyint_t        *gr_phyint;     /* IPMP group phyint */
        uint_t          gr_nif;         /* number of interfaces in group */
        uint_t          gr_nactif;      /* number of active interfaces */
        ipmp_illgrp_t   *gr_v4;         /* V4 group information */
        ipmp_illgrp_t   *gr_v6;         /* V6 group information */
        uint_t          gr_nv4;         /* number of ills in V4 group */
        uint_t          gr_nv6;         /* number of ills in V6 group */
        uint_t          gr_pendv4;      /* number of pending ills in V4 group */
        uint_t          gr_pendv6;      /* number of pending ills in V6 group */
        mblk_t          *gr_linkdownmp; /* message used to bring link down */
        kstat_t         *gr_ksp;        /* group kstat pointer */
        uint64_t        gr_kstats0[IPMP_KSTAT_MAX]; /* baseline group kstats */
} ipmp_grp_t;

/*
 * IPMP ARP entry -- one per SIOCS*ARP entry tied to the group.  Used to keep
 * ARP up-to-date as the active set of interfaces in the group changes.
 */
typedef struct ipmp_arpent_s {
        ipaddr_t        ia_ipaddr;      /* IP address for this entry */
        boolean_t       ia_proxyarp;    /* proxy ARP entry? */
        boolean_t       ia_notified;    /* ARP notified about this entry? */
        list_node_t     ia_node;        /* next ARP entry in list */
        uint16_t        ia_flags;       /* nce_flags for the address */
        size_t          ia_lladdr_len;
        uchar_t         *ia_lladdr;
```

```c
} ipmp_arpent_t;

struct arl_s;

/*
 * Per-ill capabilities.
 */
struct ill_hcksum_capab_s {
        uint_t  ill_hcksum_version;     /* interface version */
        uint_t  ill_hcksum_txflags;     /* capabilities on transmit */
};

struct ill_zerocopy_capab_s {
        uint_t  ill_zerocopy_version;   /* interface version */
        uint_t  ill_zerocopy_flags;     /* capabilities */
};

struct ill_lso_capab_s {
        uint_t  ill_lso_flags;          /* capabilities */
        uint_t  ill_lso_max;            /* maximum size of payload */
};

/*
 * IP Lower level Structure.
 * Instance data structure in ip_open when there is a device below us.
 */
typedef struct ill_s {
        pfillinput_t ill_inputfn;       /* Fast input function selector */
        ill_if_t *ill_ifptr;            /* pointer to interface type */
        queue_t *ill_rq;                /* Read queue. */
        queue_t *ill_wq;                /* Write queue. */

        int     ill_error;              /* Error value sent up by device. */

        ipif_t  *ill_ipif;              /* Interface chain for this ILL. */

        uint_t  ill_ipif_up_count;      /* Number of IPIFs currently up. */
        uint_t  ill_max_frag;           /* Max IDU from DLPI. */
        uint_t  ill_current_frag;       /* Current IDU from DLPI. */
        uint_t  ill_mtu;                /* User-specified MTU; SIOCSLIFMTU */
        uint_t  ill_mc_mtu;             /* MTU for multi/broadcast */
        uint_t  ill_metric;             /* BSD if metric, for compatibility. */
        char    *ill_name;              /* Our name. */
        uint_t  ill_ipif_dup_count;     /* Number of duplicate addresses. */
        uint_t  ill_name_length;        /* Name length, incl. terminator. */
        uint_t  ill_net_type;           /* IRE_IF_RESOLVER/IRE_IF_NORESOLVER. */
        /*
         * Physical Point of Attachment num.  If DLPI style 1 provider
         * then this is derived from the devname.
         */
        uint_t  ill_ppa;
        t_uscalar_t     ill_sap;
        t_scalar_t      ill_sap_length; /* Including sign (for position) */
        uint_t  ill_phys_addr_length;   /* Excluding the sap. */
        uint_t  ill_bcast_addr_length;  /* Only set when the DL provider */
                                        /* supports broadcast. */
        t_uscalar_t     ill_mactype;
        uint8_t *ill_frag_ptr;          /* Reassembly state. */
        timeout_id_t ill_frag_timer_id; /* timeout id for the frag timer */
        ipfb_t  *ill_frag_hash_tbl;     /* Fragment hash list head. */

        krwlock_t ill_mcast_lock;       /* Protects multicast state */
        kmutex_t ill_mcast_serializer;  /* Serialize across ilg and ilm state */
        ilm_t   *ill_ilm;               /* Multicast membership for ill */
        uint_t  ill_global_timer;       /* for IGMPv3/MLDv2 general queries */
        int     ill_mcast_type;         /* type of router which is querier */
```

```c
                                        /* on this interface */
        uint16_t ill_mcast_v1_time;     /* # slow timeouts since last v1 qry */
        uint16_t ill_mcast_v2_time;     /* # slow timeouts since last v2 qry */
        uint8_t ill_mcast_v1_tset;      /* 1 => timer is set; 0 => not set */
        uint8_t ill_mcast_v2_tset;      /* 1 => timer is set; 0 => not set */

        uint8_t ill_mcast_rv;           /* IGMPv3/MLDv2 robustness variable */
        int     ill_mcast_qi;           /* IGMPv3/MLDv2 query interval var */

        /*
         * All non-NULL cells between 'ill_first_mp_to_free' and
         * 'ill_last_mp_to_free' are freed in ill_delete.
         */
#define ill_first_mp_to_free    ill_bcast_mp
        mblk_t  *ill_bcast_mp;          /* DLPI header for broadcasts. */
        mblk_t  *ill_unbind_mp;         /* unbind mp from ill_dl_up() */
        mblk_t  *ill_promiscoff_mp;     /* for ill_leave_allmulti() */
        mblk_t  *ill_dlpi_deferred;     /* b_next chain of control messages */
        mblk_t  *ill_dest_addr_mp;      /* mblk which holds ill_dest_addr */
        mblk_t  *ill_replumb_mp;        /* replumb mp from ill_replumb() */
        mblk_t  *ill_phys_addr_mp;      /* mblk which holds ill_phys_addr */
        mblk_t  *ill_mcast_deferred;    /* b_next chain of IGMP/MLD packets */
#define ill_last_mp_to_free     ill_mcast_deferred

        cred_t  *ill_credp;             /* opener's credentials */
        uint8_t *ill_phys_addr;         /* ill_phys_addr_mp->b_rptr + off */
        uint8_t *ill_dest_addr;         /* ill_dest_addr_mp->b_rptr + off */

        uint_t  ill_state_flags;        /* see ILL_* flags above */

        /* Following bit fields protected by ipsq_t */
        uint_t
                ill_needs_attach : 1,
                ill_reserved : 1,
                ill_isv6 : 1,
                ill_dlpi_style_set : 1,

                ill_ifname_pending : 1,
                ill_logical_down : 1,
                ill_dl_up : 1,
                ill_up_ipifs : 1,

                ill_note_link : 1,      /* supports link-up notification */
                ill_capab_reneg : 1, /* capability renegotiation to be done */
                ill_dld_capab_inprog : 1, /* direct dld capab call in prog */
                ill_need_recover_multicast : 1,

                ill_replumbing : 1,
                ill_arl_dlpi_pending : 1,
                ill_grp_pending : 1,

                ill_pad_to_bit_31 : 17;

        /* Following bit fields protected by ill_lock */
        uint_t
                ill_fragtimer_executing : 1,
                ill_fragtimer_needrestart : 1,
                ill_manual_token : 1,   /* system won't override ill_token */
                /*
                 * ill_manual_linklocal : system will not change the
                 * linklocal whenever ill_token changes.
                 */
                ill_manual_linklocal : 1,

                ill_manual_dst_linklocal : 1, /* same for pt-pt dst linklocal */
```

```
		ill_pad_bit_31 : 27;

	/*
	 * Used in SIOCSIFMUXID and SIOCGIFMUXID for 'ifconfig unplumb'.
	 */
	int	ill_muxid;		/* muxid returned from plink */

	/* Used for IP frag reassembly throttling on a per ILL basis.  */
	uint_t	ill_ipf_gen;		/* Generation of next fragment queue */
	uint_t	ill_frag_count;		/* Count of all reassembly mblk bytes */
	uint_t	ill_frag_free_num_pkts;	 /* num of fragmented packets to free */
	clock_t	ill_last_frag_clean_time; /* time when frag's were pruned */
	int	ill_type;		/* From <net/if_types.h> */
	uint_t	ill_dlpi_multicast_state;	/* See below IDS_* */
	uint_t	ill_dlpi_fastpath_state;	/* See below IDS_* */

	/*
	 * Capabilities related fields.
	 */
	uint_t	ill_dlpi_capab_state;	/* State of capability query, IDCS_* */
	uint_t	ill_capab_pending_cnt;
	uint64_t ill_capabilities;	/* Enabled capabilities, ILL_CAPAB_* */
	ill_hcksum_capab_t *ill_hcksum_capab; /* H/W cksumming capabilities */
	ill_zerocopy_capab_t *ill_zerocopy_capab; /* Zero-copy capabilities */
	ill_dld_capab_t *ill_dld_capab; /* DLD capabilities */
	ill_lso_capab_t *ill_lso_capab;	/* Large Segment Offload capabilities */
	mblk_t	*ill_capab_reset_mp;	/* Preallocated mblk for capab reset */

	uint8_t	ill_max_hops;	/* Maximum hops for any logical interface */
	uint_t	ill_user_mtu;	/* User-specified MTU via SIOCSLIFLNKINFO */
	uint32_t ill_reachable_time;	/* Value for ND algorithm in msec */
	uint32_t ill_reachable_retrans_time; /* Value for ND algorithm msec */
	uint_t	ill_max_buf;		/* Max # of req to buffer for ND */
	in6_addr_t	ill_token;	/* IPv6 interface id */
	in6_addr_t	ill_dest_token;	/* Destination IPv6 interface id */
	uint_t		ill_token_length;
	uint32_t	ill_xmit_count;		/* ndp max multicast xmits */
	mib2_ipIfStatsEntry_t	*ill_ip_mib;	/* ver indep. interface mib */
	mib2_ipv6IfIcmpEntry_t	*ill_icmp6_mib; /* Per interface mib */

	phyint_t		*ill_phyint;
	uint64_t		ill_flags;

	kmutex_t	ill_lock;	/* Please see table below */
	/*
	 * The ill_nd_lla* fields handle the link layer address option
	 * from neighbor discovery. This is used for external IPv6
	 * address resolution.
	 */
	mblk_t		*ill_nd_lla_mp; /* mblk which holds ill_nd_lla */
	uint8_t		*ill_nd_lla;	/* Link Layer Address */
	uint_t		ill_nd_lla_len; /* Link Layer Address length */
	/*
	 * We have 4 phys_addr_req's sent down. This field keeps track
	 * of which one is pending.
	 */
	t_uscalar_t	ill_phys_addr_pend; /* which dl_phys_addr_req pending */
	/*
	 * Used to save errors that occur during plumbing
	 */
	uint_t		ill_ifname_pending_err;
	avl_node_t	ill_avl_byppa; /* avl node based on ppa */
	list_t		ill_nce; /* pointer to nce_s list */
	uint_t		ill_refcnt;	/* active refcnt by threads */
	uint_t		ill_ire_cnt;	/* ires associated with this ill */
	kcondvar_t	ill_cv;
```

```
	uint_t		ill_ncec_cnt;	/* ncecs associated with this ill */
	uint_t		ill_nce_cnt;	/* nces associated with this ill */
	uint_t		ill_waiters;	/* threads waiting in ipsq_enter */
	/*
	 * Contains the upper read queue pointer of the module immediately
	 * beneath IP.  This field allows IP to validate sub-capability
	 * acknowledgments coming up from downstream.
	 */
	queue_t		*ill_lmod_rq;	/* read queue pointer of module below */
	uint_t		ill_lmod_cnt;	/* number of modules beneath IP */
	ip_m_t		*ill_media;	/* media specific params/functions */
	t_uscalar_t	ill_dlpi_pending; /* Last DLPI primitive issued */
	uint_t		ill_usesrc_ifindex; /* use src addr from this ILL */
	struct ill_s	*ill_usesrc_grp_next; /* Next ILL in the usesrc group */
	boolean_t	ill_trace_disable;	/* True when alloc fails */
	zoneid_t	ill_zoneid;
	ip_stack_t	*ill_ipst;	/* Corresponds to a netstack_hold */
	uint32_t	ill_dhcpinit;	/* IP_DHCPINIT_IFs for ill */
	void		*ill_flownotify_mh; /* Tx flow ctl, mac cb handle */
	uint_t		ill_ilm_cnt;	/* ilms referencing this ill */
	uint_t		ill_ipallmulti_cnt; /* ip_join_allmulti() calls */
	ilm_t		*ill_ipallmulti_ilm;

	mblk_t		*ill_saved_ire_mp; /* Allocated for each extra IRE */
					/* with ire_ill set so they can */
					/* survive the ill going down and up. */
	kmutex_t	ill_saved_ire_lock; /* Protects ill_saved_ire_mp, cnt */
	uint_t		ill_saved_ire_cnt;	/* # entries */
	struct arl_ill_common_s	*ill_common;
	ire_t		*ill_ire_multicast; /* IRE_MULTICAST for ill */
	clock_t		ill_defend_start;   /* start of 1 hour period */
	uint_t		ill_defend_count;   /* # of announce/defends per ill */
	/*
	 * IPMP fields.
	 */
	ipmp_illgrp_t	*ill_grp;	/* IPMP group information */
	list_node_t	ill_actnode;	/* next active ill in group */
	list_node_t	ill_grpnode;	/* next ill in group */
	ipif_t		*ill_src_ipif;	/* source address selection rotor */
	ipif_t		*ill_move_ipif; /* ipif awaiting move to new ill */
	boolean_t	ill_nom_cast;	/* nominated for mcast/bcast */
	uint_t		ill_bound_cnt;	/* # of data addresses bound to ill */
	ipif_t		*ill_bound_ipif; /* ipif chain bound to ill */
	timeout_id_t	ill_refresh_tid; /* ill refresh retry timeout id */

	uint32_t	ill_mrouter_cnt; /* mrouter allmulti joins */
	uint32_t	ill_allowed_ips_cnt;
	in6_addr_t	*ill_allowed_ips;

	/* list of multicast physical addresses joined on this ill */
	multiphysaddr_t *ill_mphysaddr_list;
} ill_t;

/*
 * ILL_FREE_OK() means that there are no incoming pointer references
 * to the ill.
 */
#define ILL_FREE_OK(ill)					\
	((ill)->ill_ire_cnt == 0 && (ill)->ill_ilm_cnt == 0 &&	\
	(ill)->ill_ncec_cnt == 0 && (ill)->ill_nce_cnt == 0)

/*
 * An ipif/ill can be marked down only when the ire and ncec references
 * to that ipif/ill goes to zero. ILL_DOWN_OK() is a necessary condition
 * quiescence checks. See comments above IPIF_DOWN_OK for details
 * on why ires and nces are selectively considered for this macro.
 */
#define ILL_DOWN_OK(ill)                                        \
        (ill->ill_ire_cnt == 0 && ill->ill_ncec_cnt == 0 &&     \
        ill->ill_nce_cnt == 0)

/*
 * The following table lists the protection levels of the various members
 * of the ill_t. Same notation as that used for ipif_t above is used.
 *
 *                              Write                   Read
 *
 * ill_ifptr                    ill_g_lock + s          Write once
 * ill_rq                       ipsq                    Write once
 * ill_wq                       ipsq                    Write once
 *
 * ill_error                    ipsq                    None
 * ill_ipif                     ill_g_lock + ipsq       ill_g_lock OR ipsq
 * ill_ipif_up_count            ill_lock + ipsq         ill_lock OR ipsq
 * ill_max_frag                 ill_lock                ill_lock
 * ill_current_frag             ill_lock                ill_lock
 *
 * ill_name                     ill_g_lock + ipsq       Write once
 * ill_name_length              ill_g_lock + ipsq       Write once
 * ill_ndd_name                 ipsq                    Write once
 * ill_net_type                 ipsq                    Write once
 * ill_ppa                      ill_g_lock + ipsq       Write once
 * ill_sap                      ipsq + down ill         Write once
 * ill_sap_length               ipsq + down ill         Write once
 * ill_phys_addr_length         ipsq + down ill         Write once
 *
 * ill_bcast_addr_length        ipsq                    ipsq
 * ill_mactype                  ipsq                    ipsq
 * ill_frag_ptr                 ipsq                    ipsq
 *
 * ill_frag_timer_id            ill_lock                ill_lock
 * ill_frag_hash_tbl            ipsq                    up ill
 * ill_ilm                      ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_global_timer             ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_type               ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_v1_time            ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_v2_time            ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_v1_tset            ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_v2_tset            ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_rv                 ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 * ill_mcast_qi                 ill_mcast_lock(WRITER)  ill_mcast_lock(READER)
 *
 * ill_down_mp                  ipsq                    ipsq
 * ill_dlpi_deferred            ill_lock                ill_lock
 * ill_dlpi_pending             ipsq + ill_lock         ipsq or ill_lock or
 *                                                      absence of ipsq writer.
 * ill_phys_addr_mp             ipsq + down ill         only when ill is up
 * ill_mcast_deferred           ill_lock                ill_lock
 * ill_phys_addr                ipsq + down ill         only when ill is up
 * ill_dest_addr_mp             ipsq + down ill         only when ill is up
 * ill_dest_addr                ipsq + down ill         only when ill is up
 *
 * ill_state_flags              ill_lock                ill_lock
 * exclusive bit flags          ipsq_t                  ipsq_t
 * shared bit flags             ill_lock                ill_lock
 *
 * ill_muxid                    ipsq                    Not atomic
 *
 * ill_ipf_gen                  Not atomic
 * ill_frag_count               atomics                 atomics
 * ill_type                     ipsq + down ill         only when ill is up
 * ill_dlpi_multicast_state     ill_lock                ill_lock
```

```
 * ill_dlpi_fastpath_state      ill_lock                ill_lock
 * ill_dlpi_capab_state         ipsq                    ipsq
 * ill_max_hops                 ipsq                    Not atomic
 *
 * ill_mtu                      ill_lock                None
 * ill_mc_mtu                   ill_lock                None
 *
 * ill_user_mtu                 ipsq + ill_lock         ill_lock
 * ill_reachable_time           ipsq + ill_lock         ill_lock
 * ill_reachable_retrans_time   ipsq + ill_lock         ill_lock
 * ill_max_buf                  ipsq + ill_lock         ill_lock
 *
 * Next 2 fields need ill_lock because of the get ioctls. They should not
 * report partially updated results without executing in the ipsq.
 * ill_token                    ipsq + ill_lock         ill_lock
 * ill_token_length             ipsq + ill_lock         ill_lock
 * ill_dest_token               ipsq + down ill         only when ill is up
 * ill_xmit_count               ipsq + down ill         write once
 * ill_ip6_mib                  ipsq + down ill         only when ill is up
 * ill_icmp6_mib                ipsq + down ill         only when ill is up
 *
 * ill_phyint                   ipsq, ill_g_lock, ill_lock      Any of them
 * ill_flags                    ill_lock                ill_lock
 * ill_nd_lla_mp                ipsq + down ill         only when ill is up
 * ill_nd_lla                   ipsq + down ill         only when ill is up
 * ill_nd_lla_len               ipsq + down ill         only when ill is up
 * ill_phys_addr_pend           ipsq + down ill         only when ill is up
 * ill_ifname_pending_err       ipsq                    ipsq
 * ill_avl_byppa                ipsq, ill_g_lock        write once
 *
 * ill_fastpath_list            ill_lock                ill_lock
 * ill_refcnt                   ill_lock                ill_lock
 * ill_ire_cnt                  ill_lock                ill_lock
 * ill_cv                       ill_lock                ill_lock
 * ill_ncec_cnt                 ill_lock                ill_lock
 * ill_nce_cnt                  ill_lock                ill_lock
 * ill_ilm_cnt                  ill_lock                ill_lock
 * ill_src_ipif                 ill_g_lock              ill_g_lock
 * ill_trace                    ill_lock                ill_lock
 * ill_usesrc_grp_next          ill_g_usesrc_lock       ill_g_usesrc_lock
 * ill_dhcpinit                 atomics                 atomics
 * ill_flownotify_mh            write once              write once
 * ill_capab_pending_cnt        ipsq                    ipsq
 * ill_ipallmulti_cnt           ill_lock                ill_lock
 * ill_ipallmulti_ilm           ill_lock                ill_lock
 * ill_saved_ire_mp             ill_saved_ire_lock      ill_saved_ire_lock
 * ill_saved_ire_cnt            ill_saved_ire_lock      ill_saved_ire_lock
 * ill_arl                      ???                     ???
 * ill_ire_multicast            ipsq + quiescent        none
 * ill_bound_ipif               ipsq                    ipsq
 * ill_actnode                  ipsq + ipmp_lock        ipsq OR ipmp_lock
 * ill_grpnode                  ipsq + ill_g_lock       ipsq OR ill_g_lock
 * ill_src_ipif                 ill_g_lock              ill_g_lock
 * ill_move_ipif                ipsq                    ipsq
 * ill_nom_cast                 ipsq                    ipsq OR advisory
 * ill_refresh_tid              ill_lock                ill_lock
 * ill_grp (for IPMP ill)       write once              write once
 * ill_grp (for underlying ill) ipsq + ill_g_lock       ipsq OR ill_g_lock
 * ill_grp_pending              ill_mcast_serializer    ill_mcast_serializer
 * ill_mrouter_cnt              atomics                 atomics
 * ill_mphysaddr_list   ill_lock                ill_lock
 *
 * NOTE: It's OK to make heuristic decisions on an underlying interface
 *       by using IS_UNDER_IPMP() or comparing ill_grp's raw pointer value.
 */
```

```
/*
 * For ioctl restart mechanism see ip_reprocess_ioctl()
 */
struct ip_ioctl_cmd_s;

typedef int (*ifunc_t)(ipif_t *, struct sockaddr_in *, queue_t *, mblk_t *,
    struct ip_ioctl_cmd_s *, void *);

typedef struct ip_ioctl_cmd_s {
        int     ipi_cmd;
        size_t  ipi_copyin_size;
        uint_t  ipi_flags;
        uint_t  ipi_cmd_type;
        ifunc_t ipi_func;
        ifunc_t ipi_func_restart;
} ip_ioctl_cmd_t;

/*
 * ipi_cmd_type:
 *
 * IF_CMD               1       old style ifreq cmd
 * LIF_CMD              2       new style lifreq cmd
 * ARP_CMD              3       arpreq cmd
 * XARP_CMD             4       xarpreq cmd
 * MSFILT_CMD           5       multicast source filter cmd
 * MISC_CMD             6       misc cmd (not a more specific one above)
 */

enum { IF_CMD = 1, LIF_CMD, ARP_CMD, XARP_CMD, MSFILT_CMD, MISC_CMD };

#define IPI_DONTCARE    0       /* For ioctl encoded values that don't matter */

/* Flag values in ipi_flags */
#define IPI_PRIV        0x1     /* Root only command */
#define IPI_MODOK       0x2     /* Permitted on mod instance of IP */
#define IPI_WR          0x4     /* Need to grab writer access */
#define IPI_GET_CMD     0x8     /* branch to mi_copyout on success */
/*      unused          0x10    */
#define IPI_NULL_BCONT  0x20    /* ioctl has not data and hence no b_cont */

extern ip_ioctl_cmd_t   ip_ndx_ioctl_table[];
extern ip_ioctl_cmd_t   ip_misc_ioctl_table[];
extern int ip_ndx_ioctl_count;
extern int ip_misc_ioctl_count;

/* Passed down by ARP to IP during I_PLINK/I_PUNLINK */
typedef struct ipmx_s {
        char    ipmx_name[LIFNAMSIZ];           /* if name */
        uint_t
                ipmx_arpdev_stream : 1,         /* This is the arp stream */
                ipmx_notused : 31;
} ipmx_t;

/*
 * State for detecting if a driver supports certain features.
 * Support for DL_ENABMULTI_REQ uses ill_dlpi_multicast_state.
 * Support for DLPI M_DATA fastpath uses ill_dlpi_fastpath_state.
 */
#define IDS_UNKNOWN     0       /* No DLPI request sent */
#define IDS_INPROGRESS  1       /* DLPI request sent */
#define IDS_OK          2       /* DLPI request completed successfully */
#define IDS_FAILED      3       /* DLPI request failed */

/* Support for DL_CAPABILITY_REQ uses ill_dlpi_capab_state. */
enum {
        IDCS_UNKNOWN,
```

```
        IDCS_PROBE_SENT,
        IDCS_OK,
        IDCS_RESET_SENT,
        IDCS_RENEG,
        IDCS_FAILED
};

/* Extended NDP Management Structure */
typedef struct ipndp_s {
        ndgetf_t        ip_ndp_getf;
        ndsetf_t        ip_ndp_setf;
        caddr_t         ip_ndp_data;
        char            *ip_ndp_name;
} ipndp_t;

/* IXA Notification types */
typedef enum {
        IXAN_LSO,       /* LSO capability change */
        IXAN_PMTU,      /* PMTU change */
        IXAN_ZCOPY      /* ZEROCOPY capability change */
} ixa_notify_type_t;

typedef uint_t ixa_notify_arg_t;

typedef void    (*ixa_notify_t)(void *, ip_xmit_attr_t *ixa, ixa_notify_type_t,
    ixa_notify_arg_t);

/*
 * Attribute flags that are common to the transmit and receive attributes
 */
#define IAF_IS_IPV4             0x80000000      /* ipsec_*_v4 */
#define IAF_TRUSTED_ICMP        0x40000000      /* ipsec_*_icmp_loopback */
#define IAF_NO_LOOP_ZONEID_SET  0x20000000      /* Zone that shouldn't have */
                                                /* a copy */
#define IAF_LOOPBACK_COPY       0x10000000      /* For multi and broadcast */

#define IAF_MASK                0xf0000000      /* Flags that are common */

/*
 * Transmit side attributes used between the transport protocols and IP as
 * well as inside IP. It is also used to cache information in the conn_t i.e.
 * replaces conn_ire and the IPsec caching in the conn_t.
 */
struct ip_xmit_attr_s {
        iaflags_t       ixa_flags;      /* IXAF_*. See below */

        uint32_t        ixa_free_flags; /* IXA_FREE_*. See below */
        uint32_t        ixa_refcnt;     /* Using atomics */

        /*
         * Always initialized independently of ixa_flags settings.
         * Used by ip_xmit so we keep them up front for cache locality.
         */
        uint32_t        ixa_xmit_hint;  /* For ECMP and GLD TX ring fanout */
        uint_t          ixa_pktlen;     /* Always set. For frag and stats */
        zoneid_t        ixa_zoneid;     /* Assumed always set */

        /* Always set for conn_ip_output(); might be stale */
        /*
         * Since TCP keeps the conn_t around past the process going away
         * we need to use the "notr" (e.g, ire_refhold_notr) for ixa_ire,
         * ixa_nce, and ixa_dce.
         */
        ire_t           *ixa_ire;       /* Forwarding table entry */
        uint_t          ixa_ire_generation;
        nce_t           *ixa_nce;       /* Neighbor cache entry */
        dce_t           *ixa_dce;       /* Destination cache entry */
        uint_t          ixa_dce_generation;
        uint_t          ixa_src_generation;     /* If IXAF_VERIFY_SOURCE */

        uint32_t        ixa_src_preferences;    /* prefs for src addr select */
        uint32_t        ixa_pmtu;               /* IXAF_VERIFY_PMTU */

        /* Set by ULP if IXAF_VERIFY_PMTU; otherwise set by IP */
        uint32_t        ixa_fragsize;

        int8_t          ixa_use_min_mtu;        /* IXAF_USE_MIN_MTU values */

        pfirepostfrag_t ixa_postfragfn;         /* Set internally in IP */

        in6_addr_t      ixa_nexthop_v6;         /* IXAF_NEXTHOP_SET */
#define ixa_nexthop_v4  V4_PART_OF_V6(ixa_nexthop_v6)

        zoneid_t        ixa_no_loop_zoneid;     /* IXAF_NO_LOOP_ZONEID_SET */

        uint_t          ixa_scopeid;            /* For IPv6 link-locals */

        uint_t          ixa_broadcast_ttl;      /* IXAF_BROACAST_TTL_SET */

        uint_t          ixa_multicast_ttl;      /* Assumed set for multicast */
        uint_t          ixa_multicast_ifindex;  /* Assumed set for multicast */
        ipaddr_t        ixa_multicast_ifaddr;   /* Assumed set for multicast */

        int             ixa_raw_cksum_offset;   /* If IXAF_SET_RAW_CKSUM */

        uint32_t        ixa_ident;              /* For IPv6 fragment header */

        uint64_t        ixa_conn_id;            /* Used by DTrace */
        /*
         * Cached LSO information.
         */
        ill_lso_capab_t ixa_lso_capab;          /* Valid when IXAF_LSO_CAPAB */

        uint64_t        ixa_ipsec_policy_gen;   /* Generation from iph_gen */
        /*
         * The following IPsec fields are only initialized when
         * IXAF_IPSEC_SECURE is set. Otherwise they contain garbage.
         */
        ipsec_latch_t   *ixa_ipsec_latch;       /* Just the ids */
        struct ipsa_s   *ixa_ipsec_ah_sa;       /* Hard reference SA for AH */
        struct ipsa_s   *ixa_ipsec_esp_sa;      /* Hard reference SA for ESP */
        struct ipsec_policy_s   *ixa_ipsec_policy; /* why are we here? */
        struct ipsec_action_s   *ixa_ipsec_action; /* For reflected packets */
        ipsa_ref_t      ixa_ipsec_ref[2];       /* Soft reference to SA */
                                                /* 0: ESP, 1: AH */

        /*
         * The selectors here are potentially different than the SPD rule’s
         * selectors, and we need to have both available for IKEv2.
         *
         * NOTE: "Source" and "Dest" are w.r.t. outbound datagrams.  Ports can
         *       be zero, and the protocol number is needed to make the ports
         *       significant.
         */
        uint16_t ixa_ipsec_src_port;    /* Source port number of d-gram. */
        uint16_t ixa_ipsec_dst_port;    /* Destination port number of d-gram. */
        uint8_t  ixa_ipsec_icmp_type;   /* ICMP type of d-gram */
        uint8_t  ixa_ipsec_icmp_code;   /* ICMP code of d-gram */

        sa_family_t ixa_ipsec_inaf;     /* Inner address family */
#define IXA_MAX_ADDRLEN 4       /* Max addr len. (in 32-bit words) */
        uint32_t ixa_ipsec_insrc[IXA_MAX_ADDRLEN];      /* Inner src address */
```

```
        uint32_t ixa_ipsec_indst[IXA_MAX_ADDRLEN];      /* Inner dest address */
        uint8_t  ixa_ipsec_insrcpfx;    /* Inner source prefix */
        uint8_t  ixa_ipsec_indstpfx;    /* Inner destination prefix */

        uint8_t ixa_ipsec_proto;        /* IP protocol number for d-gram. */

        /* Always initialized independently of ixa_flags settings */
        uint_t          ixa_ifindex;    /* Assumed always set */
        uint16_t        ixa_ip_hdr_length; /* Points to ULP header */
        uint8_t         ixa_protocol;   /* Protocol number for ULP cksum */
        ts_label_t      *ixa_tsl;       /* Always set. NULL if not TX */
        ip_stack_t      *ixa_ipst;      /* Always set */
        uint32_t        ixa_extra_ident; /* Set if LSO */
        cred_t          *ixa_cred;      /* For getpeerucred */
        pid_t           ixa_cpid;       /* For getpeerucred */

#ifdef DEBUG
        kthread_t       *ixa_curthread; /* For serialization assert */
#endif
        squeue_t        *ixa_sqp;       /* Set from conn_sqp as a hint */
        uintptr_t       ixa_cookie;     /* cookie to use for tx flow control */

        /*
         * Must be set by ULP if any of IXAF_VERIFY_LSO, IXAF_VERIFY_PMTU,
         * or IXAF_VERIFY_ZCOPY is set.
         */
        ixa_notify_t    ixa_notify;     /* Registered upcall notify function */
        void            *ixa_notify_cookie; /* ULP cookie for ixa_notify */
};

/*
 * Flags to indicate which transmit attributes are set.
 * Split into "xxx_SET" ones which indicate that the "xxx" field it set, and
 * single flags.
 */
#define IXAF_REACH_CONF         0x00000001      /* Reachability confirmation */
#define IXAF_BROADCAST_TTL_SET  0x00000002      /* ixa_broadcast_ttl valid */
#define IXAF_SET_SOURCE         0x00000004      /* Replace if broadcast */
#define IXAF_USE_MIN_MTU        0x00000008      /* IPV6_USE_MIN_MTU */

#define IXAF_DONTFRAG           0x00000010      /* IP*_DONTFRAG */
#define IXAF_VERIFY_PMTU        0x00000020      /* ixa_pmtu/ixa_fragsize set */
#define IXAF_PMTU_DISCOVERY     0x00000040      /* Create/use PMTU state */
#define IXAF_MULTICAST_LOOP     0x00000080      /* IP_MULTICAST_LOOP */

#define IXAF_IPSEC_SECURE       0x00000100      /* Need IPsec processing */
#define IXAF_UCRED_TSL          0x00000200      /* ixa_tsl from SCM_UCRED */
#define IXAF_DONTROUTE          0x00000400      /* SO_DONTROUTE */
#define IXAF_NO_IPSEC           0x00000800      /* Ignore policy */

#define IXAF_PMTU_TOO_SMALL     0x00001000      /* PMTU too small */
#define IXAF_SET_ULP_CKSUM      0x00002000      /* Calculate ULP checksum */
#define IXAF_VERIFY_SOURCE      0x00004000      /* Check that source is ok */
#define IXAF_NEXTHOP_SET        0x00008000      /* ixa_nexthop set */

#define IXAF_PMTU_IPV4_DF       0x00010000      /* Set IPv4 DF */
#define IXAF_NO_DEV_FLOW_CTL    0x00020000      /* Protocol needs no flow ctl */
#define IXAF_NO_TTL_CHANGE      0x00040000      /* Internal to IP */
#define IXAF_IPV6_ADD_FRAGHDR   0x00080000      /* Add fragment header */

#define IXAF_IPSEC_TUNNEL       0x00100000      /* Tunnel mode */
#define IXAF_NO_PFHOOK          0x00200000      /* Skip xmit pfhook */
#define IXAF_NO_TRACE           0x00400000      /* When back from ARP/ND */
#define IXAF_SCOPEID_SET        0x00800000      /* ixa_scopeid set */

#define IXAF_MULTIRT_MULTICAST  0x01000000      /* MULTIRT for multicast */
```

```
#define IXAF_NO_HW_CKSUM        0x02000000      /* Force software cksum */
#define IXAF_SET_RAW_CKSUM      0x04000000      /* Use ixa_raw_cksum_offset */
#define IXAF_IPSEC_GLOBAL_POLICY 0x08000000     /* Policy came from global */

/* Note the following uses bits 0x10000000 through 0x80000000 */
#define IXAF_IS_IPV4            IAF_IS_IPV4
#define IXAF_TRUSTED_ICMP       IAF_TRUSTED_ICMP
#define IXAF_NO_LOOP_ZONEID_SET IAF_NO_LOOP_ZONEID_SET
#define IXAF_LOOPBACK_COPY      IAF_LOOPBACK_COPY

/* Note: use the upper 32 bits */
#define IXAF_VERIFY_LSO         0x100000000     /* Check LSO capability */
#define IXAF_LSO_CAPAB          0x200000000     /* Capable of LSO */
#define IXAF_VERIFY_ZCOPY       0x400000000     /* Check Zero Copy capability */
#define IXAF_ZCOPY_CAPAB        0x800000000     /* Capable of ZEROCOPY */

/*
 * The normal flags for sending packets e.g., icmp errors
 */
#define IXAF_BASIC_SIMPLE_V4    \
        (IXAF_SET_ULP_CKSUM | IXAF_IS_IPV4 | IXAF_VERIFY_SOURCE)
#define IXAF_BASIC_SIMPLE_V6    (IXAF_SET_ULP_CKSUM | IXAF_VERIFY_SOURCE)

/*
 * Normally these fields do not have a hold. But in some cases they do, for
 * instance when we've gone through ip_*_attr_to/from_mblk.
 * We use ixa_free_flags to indicate that they have a hold and need to be
 * released on cleanup.
 */
#define IXA_FREE_CRED           0x00000001      /* ixa_cred needs to be rele */
#define IXA_FREE_TSL            0x00000002      /* ixa_tsl needs to be rele */

/*
 * Simplistic way to set the ixa_xmit_hint for locally generated traffic
 * and forwarded traffic. The shift amount are based on the size of the
 * structs to discard the low order bits which don't have much if any variation
 * (coloring in kmem_cache_alloc might provide some variation).
 *
 * Basing the locally generated hint on the address of the conn_t means that
 * the packets from the same socket/connection do not get reordered.
 * Basing the hint for forwarded traffic on the ill_ring_t means that
 * packets from the same NIC+ring are likely to use the same outbound ring
 * hence we get low contention on the ring in the transmitting driver.
 */
#define CONN_TO_XMIT_HINT(connp)        ((uint32_t)(((uintptr_t)connp) >> 11))
#define ILL_RING_TO_XMIT_HINT(ring)     ((uint32_t)(((uintptr_t)ring) >> 7))

/*
 * IP set Destination Flags used by function ip_set_destination,
 * ip_attr_connect, and conn_connect.
 */
#define IPDF_ALLOW_MCBC         0x1     /* Allow multi/broadcast */
#define IPDF_VERIFY_DST         0x2     /* Verify destination addr */
#define IPDF_SELECT_SRC         0x4     /* Select source address */
#define IPDF_LSO                0x8     /* Try LSO */
#define IPDF_IPSEC              0x10    /* Set IPsec policy */
#define IPDF_ZONE_IS_GLOBAL     0x20    /* From conn_zone_is_global */
#define IPDF_ZCOPY              0x40    /* Try ZEROCOPY */
#define IPDF_UNIQUE_DCE         0x80    /* Get a per-destination DCE */

/*
 * Receive side attributes used between the transport protocols and IP as
 * well as inside IP.
 */
struct ip_recv_attr_s {
        iaflags_t       ira_flags;      /* See below */
```

```
        uint32_t        ira_free_flags; /* IRA_FREE_*. See below */

        /*
         * This is a hint for TCP SYN packets.
         * Always initialized independently of ira_flags settings
         */
        squeue_t        *ira_sqp;
        ill_rx_ring_t   *ira_ring;      /* Internal to IP */

        /* For ip_accept_tcp when IRAF_TARGET_SQP is set */
        squeue_t        *ira_target_sqp;
        mblk_t          *ira_target_sqp_mp;

        /* Always initialized independently of ira_flags settings */
        uint32_t        ira_xmit_hint;  /* For ECMP and GLD TX ring fanout */
        zoneid_t        ira_zoneid;     /* ALL_ZONES unless local delivery */
        uint_t          ira_pktlen;     /* Always set. For frag and stats */
        uint16_t        ira_ip_hdr_length; /* Points to ULP header */
        uint8_t         ira_protocol;   /* Protocol number for ULP cksum */
        uint_t          ira_rifindex;   /* Received ifindex */
        uint_t          ira_ruifindex;  /* Received upper ifindex */
        ts_label_t      *ira_tsl;       /* Always set. NULL if not TX */
        /*
         * ira_rill and ira_ill is set inside IP, but not when conn_recv is
         * called; ULPs should use ira_ruifindex instead.
         */
        ill_t           *ira_rill;      /* ill where packet came */
        ill_t           *ira_ill;       /* ill where IP address hosted */
        cred_t          *ira_cred;      /* For getpeerucred */
        pid_t           ira_cpid;       /* For getpeerucred */

        /* Used when IRAF_VERIFIED_SRC is set; this source was ok */
        ipaddr_t        ira_verified_src;

        /*
         * The following IPsec fields are only initialized when
         * IRAF_IPSEC_SECURE is set. Otherwise they contain garbage.
         */
        struct ipsec_action_s *ira_ipsec_action; /* how we made it in.. */
        struct ipsa_s   *ira_ipsec_ah_sa;       /* SA for AH */
        struct ipsa_s   *ira_ipsec_esp_sa;      /* SA for ESP */

        ipaddr_t        ira_mroute_tunnel;      /* IRAF_MROUTE_TUNNEL_SET */

        zoneid_t        ira_no_loop_zoneid;     /* IRAF_NO_LOOP_ZONEID_SET */

        uint32_t        ira_esp_udp_ports;      /* IRAF_ESP_UDP_PORTS */

        /*
         * For IP_RECVSLLA and ip_ndp_conflict/find_solicitation.
         * Same size as max for sockaddr_dl
         */
#define IRA_L2SRC_SIZE  244
        uint8_t         ira_l2src[IRA_L2SRC_SIZE];      /* If IRAF_L2SRC_SET */

        /*
         * Local handle that we use to do lazy setting of ira_l2src.
         * We defer setting l2src until needed but we do before any
         * ip_input pullupmsg or copymsg.
         */
        struct mac_header_info_s *ira_mhip;     /* Could be NULL */
};

/*
 * Flags to indicate which receive attributes are set.
 */
#define IRAF_SYSTEM_LABELED     0x00000001      /* is_system_labeled() */
#define IRAF_IPV4_OPTIONS       0x00000002      /* Performance */
#define IRAF_MULTICAST          0x00000004      /* Was multicast at L3 */
#define IRAF_BROADCAST          0x00000008      /* Was broadcast at L3 */
#define IRAF_MULTIBROADCAST     (IRAF_MULTICAST|IRAF_BROADCAST)

#define IRAF_LOOPBACK           0x00000010      /* Looped back by IP */
#define IRAF_VERIFY_IP_CKSUM    0x00000020      /* Need to verify IP */
#define IRAF_VERIFY_ULP_CKSUM   0x00000040      /* Need to verify TCP,UDP,etc */
#define IRAF_SCTP_CSUM_ERR      0x00000080      /* sctp pkt has failed chksum */

#define IRAF_IPSEC_SECURE       0x00000100      /* Passed AH and/or ESP */
#define IRAF_DHCP_UNICAST       0x00000200
#define IRAF_IPSEC_DECAPS       0x00000400      /* Was packet decapsulated */
                                        /* from a matching inner packet? */
#define IRAF_TARGET_SQP         0x00000800      /* ira_target_sqp is set */
#define IRAF_VERIFIED_SRC       0x00001000      /* ira_verified_src set */
#define IRAF_RSVP               0x00002000      /* RSVP packet for rsvpd */
#define IRAF_MROUTE_TUNNEL_SET  0x00004000      /* From ip_mroute_decap */
#define IRAF_PIM_REGISTER       0x00008000      /* From register_mforward */

#define IRAF_TX_MAC_EXEMPTABLE  0x00010000      /* Allow MAC_EXEMPT readdown */
#define IRAF_TX_SHARED_ADDR     0x00020000      /* Arrived on ALL_ZONES addr */
#define IRAF_ESP_UDP_PORTS      0x00040000      /* NAT-traversal packet */
#define IRAF_NO_HW_CKSUM        0x00080000      /* Force software cksum */

#define IRAF_ICMP_ERROR         0x00100000      /* Send to conn_recvicmp */
#define IRAF_ROUTER_ALERT       0x00200000      /* IPv6 router alert */
#define IRAF_L2SRC_SET          0x00400000      /* ira_l2src has been set */
#define IRAF_L2SRC_LOOPBACK     0x00800000      /* Came from us */

#define IRAF_L2DST_MULTICAST    0x01000000      /* Multicast at L2 */
#define IRAF_L2DST_BROADCAST    0x02000000      /* Broadcast at L2 */
/* Unused 0x04000000 */
/* Unused 0x08000000 */

/* Below starts with 0x10000000 */
#define IRAF_IS_IPV4            IAF_IS_IPV4
#define IRAF_TRUSTED_ICMP       IAF_TRUSTED_ICMP
#define IRAF_NO_LOOP_ZONEID_SET IAF_NO_LOOP_ZONEID_SET
#define IRAF_LOOPBACK_COPY      IAF_LOOPBACK_COPY

/*
 * Normally these fields do not have a hold. But in some cases they do, for
 * instance when we've gone through ip_*_attr_to/from_mblk.
 * We use ira_free_flags to indicate that they have a hold and need to be
 * released on cleanup.
 */
#define IRA_FREE_CRED           0x00000001      /* ira_cred needs to be rele */
#define IRA_FREE_TSL            0x00000002      /* ira_tsl needs to be rele */

/*
 * Optional destination cache entry for path MTU information,
 * and ULP metrics.
 */
struct dce_s {
        uint_t          dce_generation; /* Changed since cached? */
        uint_t          dce_flags;      /* See below */
        uint_t          dce_ipversion;  /* IPv4/IPv6 version */
        uint32_t        dce_pmtu;       /* Path MTU if DCEF_PMTU */
        uint32_t        dce_ident;      /* Per destination IP ident. */
        iulp_t          dce_uinfo;      /* Metrics if DCEF_UINFO */

        struct dce_s    *dce_next;
        struct dce_s    **dce_ptpn;
```

```
        struct dcb_s    *dce_bucket;

        union {
                in6_addr_t      dceu_v6addr;
                ipaddr_t        dceu_v4addr;
        } dce_u;
#define dce_v4addr      dce_u.dceu_v4addr
#define dce_v6addr      dce_u.dceu_v6addr
        /* Note that for IPv6+IPMP we use the ifindex for the upper interface */
        uint_t          dce_ifindex;    /* For IPv6 link-locals */

        kmutex_t        dce_lock;
        uint_t          dce_refcnt;
        uint64_t        dce_last_change_time;   /* Path MTU. In seconds */

        ip_stack_t      *dce_ipst;      /* Does not have a netstack_hold */
};

/*
 * Values for dce_generation.
 *
 * If a DCE has DCE_GENERATION_CONDEMNED, the last dce_refrele should delete
 * it.
 *
 * DCE_GENERATION_VERIFY is never stored in dce_generation but it is
 * stored in places that cache DCE (such as ixa_dce_generation).
 * It is used as a signal that the cache is stale and needs to be reverified.
 */
#define DCE_GENERATION_CONDEMNED        0
#define DCE_GENERATION_VERIFY           1
#define DCE_GENERATION_INITIAL          2
#define DCE_IS_CONDEMNED(dce) \
        ((dce)->dce_generation == DCE_GENERATION_CONDEMNED)


/*
 * Values for ips_src_generation.
 *
 * SRC_GENERATION_VERIFY is never stored in ips_src_generation but it is
 * stored in places that cache IREs (ixa_src_generation). It is used as a
 * signal that the cache is stale and needs to be reverified.
 */
#define SRC_GENERATION_VERIFY           0
#define SRC_GENERATION_INITIAL          1

/*
 * The kernel stores security attributes of all gateways in a database made
 * up of one or more tsol_gcdb_t elements.  Each tsol_gcdb_t contains the
 * security-related credentials of the gateway.  More than one gateways may
 * share entries in the database.
 *
 * The tsol_gc_t structure represents the gateway to credential association,
 * and refers to an entry in the database.  One or more tsol_gc_t entities are
 * grouped together to form one or more tsol_gcgrp_t, each representing the
 * list of security attributes specific to the gateway.  A gateway may be
 * associated with at most one credentials group.
 */
struct tsol_gcgrp_s;

extern uchar_t  ip6opt_ls;      /* TX IPv6 enabler */

/*
 * Gateway security credential record.
 */
typedef struct tsol_gcdb_s {
        uint_t          gcdb_refcnt;    /* reference count */
        struct rtsa_s   gcdb_attr;      /* security attributes */
#define gcdb_mask       gcdb_attr.rtsa_mask
#define gcdb_doi        gcdb_attr.rtsa_doi
#define gcdb_slrange    gcdb_attr.rtsa_slrange
} tsol_gcdb_t;

/*
 * Gateway to credential association.
 */
typedef struct tsol_gc_s {
        uint_t          gc_refcnt;      /* reference count */
        struct tsol_gcgrp_s *gc_grp;    /* pointer to group */
        struct tsol_gc_s *gc_prev;      /* previous in list */
        struct tsol_gc_s *gc_next;      /* next in list */
        tsol_gcdb_t     *gc_db;         /* pointer to actual credentials */
} tsol_gc_t;

/*
 * Gateway credentials group address.
 */
typedef struct tsol_gcgrp_addr_s {
        int             ga_af;          /* address family */
        in6_addr_t      ga_addr;        /* IPv4 mapped or IPv6 address */
} tsol_gcgrp_addr_t;

/*
 * Gateway credentials group.
 */
typedef struct tsol_gcgrp_s {
        uint_t          gcgrp_refcnt;   /* reference count */
        krwlock_t       gcgrp_rwlock;   /* lock to protect following */
        uint_t          gcgrp_count;    /* number of credentials */
        tsol_gc_t       *gcgrp_head;    /* first credential in list */
        tsol_gc_t       *gcgrp_tail;    /* last credential in list */
        tsol_gcgrp_addr_t gcgrp_addr;   /* next-hop gateway address */
} tsol_gcgrp_t;

extern kmutex_t gcgrp_lock;

#define GC_REFRELE(p) {                                 \
        ASSERT((p)->gc_grp != NULL);                    \
        rw_enter(&(p)->gc_grp->gcgrp_rwlock, RW_WRITER); \
        ASSERT((p)->gc_refcnt > 0);                     \
        if (--((p)->gc_refcnt) == 0)                    \
                gc_inactive(p);                         \
        else                                            \
                rw_exit(&(p)->gc_grp->gcgrp_rwlock); \
}

#define GCGRP_REFHOLD(p) {                              \
        mutex_enter(&gcgrp_lock);                       \
        ++((p)->gcgrp_refcnt);                          \
        ASSERT((p)->gcgrp_refcnt != 0);                 \
        mutex_exit(&gcgrp_lock);                        \
}

#define GCGRP_REFRELE(p) {                              \
        mutex_enter(&gcgrp_lock);                       \
        ASSERT((p)->gcgrp_refcnt > 0);                  \
        if (--((p)->gcgrp_refcnt) == 0)                 \
                gcgrp_inactive(p);                      \
        ASSERT(MUTEX_HELD(&gcgrp_lock));                \
        mutex_exit(&gcgrp_lock);                        \
}

/*
```

```
 * IRE gateway security attributes structure, pointed to by tsol_ire_gw_secattr
 */
struct tsol_tnrhc;

struct tsol_ire_gw_secattr_s {
        kmutex_t        igsa_lock;      /* lock to protect following */
        struct tsol_tnrhc *igsa_rhc;    /* host entry for gateway */
        tsol_gc_t       *igsa_gc;       /* for prefix IREs */
};

void irb_refrele_ftable(irb_t *);

extern struct kmem_cache *rt_entry_cache;

typedef struct ire4 {
        ipaddr_t ire4_mask;             /* Mask for matching this IRE. */
        ipaddr_t ire4_addr;             /* Address this IRE represents. */
        ipaddr_t ire4_gateway_addr;     /* Gateway including for IRE_ONLINK */
        ipaddr_t ire4_setsrc_addr;      /* RTF_SETSRC */
} ire4_t;

typedef struct ire6 {
        in6_addr_t ire6_mask;           /* Mask for matching this IRE. */
        in6_addr_t ire6_addr;           /* Address this IRE represents. */
        in6_addr_t ire6_gateway_addr;   /* Gateway including for IRE_ONLINK */
        in6_addr_t ire6_setsrc_addr;    /* RTF_SETSRC */
} ire6_t;

typedef union ire_addr {
        ire6_t  ire6_u;
        ire4_t  ire4_u;
} ire_addr_u_t;

/*
 * Internet Routing Entry
 * When we have multiple identical IREs we logically add them by manipulating
 * ire_identical_ref and ire_delete first decrements
 * that and when it reaches 1 we know it is the last IRE.
 * "identical" is defined as being the same for:
 * ire_addr, ire_netmask, ire_gateway, ire_ill, ire_zoneid, and ire_type
 * For instance, multiple IRE_BROADCASTs for the same subnet number are
 * viewed as identical, and so are the IRE_INTERFACEs when there are
 * multiple logical interfaces (on the same ill) with the same subnet prefix.
 */
struct ire_s {
        struct  ire_s   *ire_next;      /* The hash chain must be first. */
        struct  ire_s   **ire_ptpn;     /* Pointer to previous next. */
        uint32_t        ire_refcnt;     /* Number of references */
        ill_t           *ire_ill;
        uint32_t        ire_identical_ref; /* IRE_INTERFACE, IRE_BROADCAST */
        uchar_t         ire_ipversion;  /* IPv4/IPv6 version */
        ushort_t        ire_type;       /* Type of IRE */
        uint_t          ire_generation; /* Generation including CONDEMNED */
        uint_t  ire_ib_pkt_count;       /* Inbound packets for ire_addr */
        uint_t  ire_ob_pkt_count;       /* Outbound packets to ire_addr */
        time_t  ire_create_time;        /* Time (in secs) IRE was created. */
        uint32_t        ire_flags;      /* flags related to route (RTF_*) */
        /*
         * ire_testhidden is TRUE for INTERFACE IREs of IS_UNDER_IPMP(ill)
         * interfaces
         */
        boolean_t       ire_testhidden;
        pfirerecv_t     ire_recvfn;     /* Receive side handling */
        pfiresend_t     ire_sendfn;     /* Send side handling */
        pfirepostfrag_t ire_postfragfn; /* Bottom end of send handling */
	uint_t		ire_masklen;	/* # bits in ire_mask{,_v6} */
	ire_addr_u_t	ire_u;		/* IPv4/IPv6 address info. */

	irb_t		*ire_bucket;	/* Hash bucket when ire_ptphn is set */
	kmutex_t	ire_lock;
	clock_t		ire_last_used_time;	/* For IRE_LOCAL reception */
	tsol_ire_gw_secattr_t *ire_gw_secattr; /* gateway security attributes */
	zoneid_t	ire_zoneid;

	/*
	 * Cached information of where to send packets that match this route.
	 * The ire_dep_* information is used to determine when ire_nce_cache
	 * needs to be updated.
	 * ire_nce_cache is the fastpath for the Neighbor Cache Entry
	 * for IPv6; arp info for IPv4
	 * Since this is a cache setup and torn down independently of
	 * applications we need to use nce_ref{rele,hold}_notr for it.
	 */
	nce_t		*ire_nce_cache;

	/*
	 * Quick check whether the ire_type and ire_masklen indicates
	 * that the IRE can have ire_nce_cache set i.e., whether it is
	 * IRE_ONLINK and for a single destination.
	 */
	boolean_t	ire_nce_capable;

	/*
	 * Dependency tracking so we can safely cache IRE and NCE pointers
	 * in offlink and onlink IREs.
	 * These are locked under the ips_ire_dep_lock rwlock. Write held
	 * when modifying the linkage.
	 * ire_dep_parent (Also chain towards IRE for nexthop)
	 * ire_dep_parent_generation: ire_generation of ire_dep_parent
	 * ire_dep_children (From parent to first child)
	 * ire_dep_sib_next (linked list of siblings)
	 * ire_dep_sib_ptpn (linked list of siblings)
	 *
	 * The parent has a ire_refhold on each child, and each child has
	 * an ire_refhold on its parent.
	 * Since ire_dep_parent is a cache setup and torn down independently of
	 * applications we need to use ire_ref{rele,hold}_notr for it.
	 */
	ire_t		*ire_dep_parent;
	ire_t		*ire_dep_children;
	ire_t		*ire_dep_sib_next;
	ire_t		**ire_dep_sib_ptpn;	/* Pointer to previous next */
	uint_t		ire_dep_parent_generation;

	uint_t		ire_badcnt;	/* Number of times ND_UNREACHABLE */
	uint64_t	ire_last_badcnt;	/* In seconds */

	/* ire_defense* and ire_last_used_time are only used on IRE_LOCALs */
	uint_t		ire_defense_count;	/* number of ARP conflicts */
	uint_t		ire_defense_time;	/* last time defended (secs) */

	boolean_t	ire_trace_disable;	/* True when alloc fails */
	ip_stack_t	*ire_ipst;	/* Does not have a netstack_hold */
	iulp_t		ire_metrics;
	/*
	 * default and prefix routes that are added without explicitly
	 * specifying the interface are termed "unbound" routes, and will
	 * have ire_unbound set to true.
	 */
	boolean_t	ire_unbound;
};
```

```
/* IPv4 compatibility macros */
#define ire_mask		ire_u.ire4_u.ire4_mask
#define ire_addr		ire_u.ire4_u.ire4_addr
#define ire_gateway_addr	ire_u.ire4_u.ire4_gateway_addr
#define ire_setsrc_addr		ire_u.ire4_u.ire4_setsrc_addr

#define ire_mask_v6		ire_u.ire6_u.ire6_mask
#define ire_addr_v6		ire_u.ire6_u.ire6_addr
#define ire_gateway_addr_v6	ire_u.ire6_u.ire6_gateway_addr
#define ire_setsrc_addr_v6	ire_u.ire6_u.ire6_setsrc_addr

/*
 * Values for ire_generation.
 *
 * If an IRE is marked with IRE_IS_CONDEMNED, the last walker of
 * the bucket should delete this IRE from this bucket.
 *
 * IRE_GENERATION_VERIFY is never stored in ire_generation but it is
 * stored in places that cache IREs (such as ixa_ire_generation and
 * ire_dep_parent_generation). It is used as a signal that the cache is
 * stale and needs to be reverified.
 */
#define IRE_GENERATION_CONDEMNED	0
#define IRE_GENERATION_VERIFY		1
#define IRE_GENERATION_INITIAL		2
#define IRE_IS_CONDEMNED(ire) \
	((ire)->ire_generation == IRE_GENERATION_CONDEMNED)

/* Convenient typedefs for sockaddrs */
typedef struct sockaddr_in	sin_t;
typedef struct sockaddr_in6	sin6_t;

/* Name/Value Descriptor. */
typedef struct nv_s {
	uint64_t nv_value;
	char	*nv_name;
} nv_t;

#define ILL_FRAG_HASH(s, i) \
	((ntohl(s) ^ ((i) ^ ((i) >> 8))) % ILL_FRAG_HASH_TBL_COUNT)

/*
 * The MAX number of allowed fragmented packets per hash bucket
 * calculation is based on the most common mtu size of 1500. This limit
 * will work well for other mtu sizes as well.
 */
#define COMMON_IP_MTU 1500
#define MAX_FRAG_MIN 10
#define MAX_FRAG_PKTS(ipst)	\
	MAX(MAX_FRAG_MIN, (2 * (ipst->ips_ip_reass_queue_bytes / \
	    (COMMON_IP_MTU * ILL_FRAG_HASH_TBL_COUNT))))

/*
 * Maximum dups allowed per packet.
 */
extern uint_t ip_max_frag_dups;

/*
 * Per-packet information for received packets and transmitted.
 * Used by the transport protocols when converting between the packet
 * and ancillary data and socket options.
 *
 * Note: This private data structure and related IPPF_* constant
 * definitions are exposed to enable compilation of some debugging tools
 * like lsof which use struct tcp_t in <inet/tcp.h>. This is intended to be
 * a temporary hack and long term alternate interfaces should be defined
 * to support the needs of such tools and private definitions moved to
 * private headers.
 */
struct ip_pkt_s {
        uint_t          ipp_fields;             /* Which fields are valid */
        in6_addr_t      ipp_addr;               /* pktinfo src/dst addr */
#define ipp_addr_v4     V4_PART_OF_V6(ipp_addr)
        uint_t          ipp_unicast_hops;       /* IPV6_UNICAST_HOPS, IP_TTL */
        uint_t          ipp_hoplimit;           /* IPV6_HOPLIMIT */
        uint_t          ipp_hopoptslen;
        uint_t          ipp_rthdrdstoptslen;
        uint_t          ipp_rthdrlen;
        uint_t          ipp_dstoptslen;
        uint_t          ipp_fraghdrlen;
        ip6_hbh_t       *ipp_hopopts;
        ip6_dest_t      *ipp_rthdrdstopts;
        ip6_rthdr_t     *ipp_rthdr;
        ip6_dest_t      *ipp_dstopts;
        ip6_frag_t      *ipp_fraghdr;
        uint8_t         ipp_tclass;             /* IPV6_TCLASS */
        uint8_t         ipp_type_of_service;    /* IP_TOS */
        uint_t          ipp_ipv4_options_len;   /* Len of IPv4 options */
        uint8_t         *ipp_ipv4_options;      /* Ptr to IPv4 options */
        uint_t          ipp_label_len_v4;       /* Len of TX label for IPv4 */
        uint8_t         *ipp_label_v4;          /* TX label for IPv4 */
        uint_t          ipp_label_len_v6;       /* Len of TX label for IPv6 */
        uint8_t         *ipp_label_v6;          /* TX label for IPv6 */
};
typedef struct ip_pkt_s ip_pkt_t;

extern void ip_pkt_free(ip_pkt_t *);    /* free storage inside ip_pkt_t */
extern ipaddr_t ip_pkt_source_route_v4(const ip_pkt_t *);
extern in6_addr_t *ip_pkt_source_route_v6(const ip_pkt_t *);
extern int ip_pkt_copy(ip_pkt_t *, ip_pkt_t *, int);
extern void ip_pkt_source_route_reverse_v4(ip_pkt_t *);

/* ipp_fields values */
#define IPPF_ADDR               0x0001  /* Part of in6_pktinfo: src/dst addr */
#define IPPF_HOPLIMIT           0x0002  /* Overrides unicast and multicast */
#define IPPF_TCLASS             0x0004  /* Overrides class in sin6_flowinfo */

#define IPPF_HOPOPTS            0x0010  /* ipp_hopopts set */
#define IPPF_RTHDR              0x0020  /* ipp_rthdr set */
#define IPPF_RTHDRDSTOPTS       0x0040  /* ipp_rthdrdstopts set */
#define IPPF_DSTOPTS            0x0080  /* ipp_dstopts set */

#define IPPF_IPV4_OPTIONS       0x0100  /* ipp_ipv4_options set */
#define IPPF_LABEL_V4           0x0200  /* ipp_label_v4 set */
#define IPPF_LABEL_V6           0x0400  /* ipp_label_v6 set */

#define IPPF_FRAGHDR            0x0800  /* Used for IPsec receive side */

/*
 * Data structure which is passed to conn_opt_get/set.
 * The conn_t is included even though it can be inferred from queue_t.
 * setsockopt and getsockopt use conn_ixa and conn_xmit_ipp. However,
 * when handling ancillary data we use separate ixa and ipps.
 */
typedef struct conn_opt_arg_s {
        conn_t          *coa_connp;
        ip_xmit_attr_t  *coa_ixa;
        ip_pkt_t        *coa_ipp;
        boolean_t       coa_ancillary;  /* Ancillary data and not setsockopt */
        uint_t          coa_changed;    /* See below */
} conn_opt_arg_t;
```

```
/*
 * Flags for what changed.
 * If we want to be more efficient in the future we can have more fine
 * grained flags e.g., a flag for just IP_TOS changing.
 * For now we either call ip_set_destination (for "route changed")
 * and/or conn_build_hdr_template/conn_prepend_hdr (for "header changed").
 */
#define COA_HEADER_CHANGED      0x0001
#define COA_ROUTE_CHANGED       0x0002
#define COA_RCVBUF_CHANGED      0x0004  /* SO_RCVBUF */
#define COA_SNDBUF_CHANGED      0x0008  /* SO_SNDBUF */
#define COA_WROFF_CHANGED       0x0010  /* Header size changed */
#define COA_ICMP_BIND_NEEDED    0x0020
#define COA_OOBINLINE_CHANGED   0x0040

#define TCP_PORTS_OFFSET        0
#define UDP_PORTS_OFFSET        0

/*
 * lookups return the ill/ipif only if the flags are clear OR Iam writer.
 * ill / ipif lookup functions increment the refcnt on the ill / ipif only
 * after calling these macros. This ensures that the refcnt on the ipif or
 * ill will eventually drop down to zero.
 */
#define ILL_LOOKUP_FAILED       1       /* Used as error code */
#define IPIF_LOOKUP_FAILED      2       /* Used as error code */

#define ILL_CAN_LOOKUP(ill)                                             \
        (!((ill)->ill_state_flags & ILL_CONDEMNED) ||                   \
        IAM_WRITER_ILL(ill))

#define ILL_IS_CONDEMNED(ill)   \
        ((ill)->ill_state_flags & ILL_CONDEMNED)

#define IPIF_CAN_LOOKUP(ipif)   \
        (!((ipif)->ipif_state_flags & IPIF_CONDEMNED) || \
        IAM_WRITER_IPIF(ipif))

#define IPIF_IS_CONDEMNED(ipif) \
        ((ipif)->ipif_state_flags & IPIF_CONDEMNED)

#define IPIF_IS_CHANGING(ipif)  \
        ((ipif)->ipif_state_flags & IPIF_CHANGING)

/* Macros used to assert that this thread is a writer */
#define IAM_WRITER_IPSQ(ipsq)   ((ipsq)->ipsq_xop->ipx_writer == curthread)
#define IAM_WRITER_ILL(ill)     IAM_WRITER_IPSQ((ill)->ill_phyint->phyint_ipsq)
#define IAM_WRITER_IPIF(ipif)   IAM_WRITER_ILL((ipif)->ipif_ill)

/*
 * Grab ill locks in the proper order. The order is highest addressed
 * ill is locked first.
 */
#define GRAB_ILL_LOCKS(ill_1, ill_2)                            \
{                                                               \
        if ((ill_1) > (ill_2)) {                                \
                if (ill_1 != NULL)                              \
                        mutex_enter(&(ill_1)->ill_lock);        \
                if (ill_2 != NULL)                              \
                        mutex_enter(&(ill_2)->ill_lock);        \
        } else {                                                \
                if (ill_2 != NULL)                              \
                        mutex_enter(&(ill_2)->ill_lock);        \
                if (ill_1 != NULL && ill_1 != ill_2)            \
                        mutex_enter(&(ill_1)->ill_lock);        \
        }                                                       \
}

#define RELEASE_ILL_LOCKS(ill_1, ill_2)         \
{                                               \
        if (ill_1 != NULL)                      \
                mutex_exit(&(ill_1)->ill_lock); \
        if (ill_2 != NULL && ill_2 != ill_1)    \
                mutex_exit(&(ill_2)->ill_lock); \
}

/* Get the other protocol instance ill */
#define ILL_OTHER(ill)                                          \
        ((ill)->ill_isv6 ? (ill)->ill_phyint->phyint_illv4 :    \
            (ill)->ill_phyint->phyint_illv6)

/* ioctl command info: Ioctl properties extracted and stored in here */
typedef struct cmd_info_s
{
        ipif_t  *ci_ipif;       /* ipif associated with [l]ifreq ioctl's */
        sin_t   *ci_sin;        /* the sin struct passed down */
        sin6_t  *ci_sin6;       /* the sin6_t struct passed down */
        struct lifreq *ci_lifr; /* the lifreq struct passed down */
} cmd_info_t;

extern struct kmem_cache *ire_cache;

extern ipaddr_t ip_g_all_ones;

extern uint_t   ip_loopback_mtu;        /* /etc/system */
extern uint_t   ip_loopback_mtuplus;
extern uint_t   ip_loopback_mtu_v6plus;

extern vmem_t *ip_minor_arena_sa;
extern vmem_t *ip_minor_arena_la;

/*
 * ip_g_forward controls IP forwarding.  It takes two values:
 *      0: IP_FORWARD_NEVER     Don't forward packets ever.
 *      1: IP_FORWARD_ALWAYS    Forward packets for elsewhere.
 *
 * RFC1122 says there must be a configuration switch to control forwarding,
 * but that the default MUST be to not forward packets ever.  Implicit
 * control based on configuration of multiple interfaces MUST NOT be
 * implemented (Section 3.1).  SunOS 4.1 did provide the "automatic" capability
 * and, in fact, it was the default.  That capability is now provided in the
 * /etc/rc2.d/S69inet script.
 */

#define ips_ip_respond_to_address_mask_broadcast \
                                        ips_propinfo_tbl[0].prop_cur_bval
#define ips_ip_g_resp_to_echo_bcast     ips_propinfo_tbl[1].prop_cur_bval
#define ips_ip_g_resp_to_echo_mcast     ips_propinfo_tbl[2].prop_cur_bval
#define ips_ip_g_resp_to_timestamp      ips_propinfo_tbl[3].prop_cur_bval
#define ips_ip_g_resp_to_timestamp_bcast ips_propinfo_tbl[4].prop_cur_bval
#define ips_ip_g_send_redirects         ips_propinfo_tbl[5].prop_cur_bval
#define ips_ip_g_forward_directed_bcast ips_propinfo_tbl[6].prop_cur_bval
#define ips_ip_mrtdebug                 ips_propinfo_tbl[7].prop_cur_uval
#define ips_ip_ire_reclaim_fraction     ips_propinfo_tbl[8].prop_cur_uval
#define ips_ip_nce_reclaim_fraction     ips_propinfo_tbl[9].prop_cur_uval
#define ips_ip_dce_reclaim_fraction     ips_propinfo_tbl[10].prop_cur_uval
#define ips_ip_def_ttl                  ips_propinfo_tbl[11].prop_cur_uval
#define ips_ip_forward_src_routed       ips_propinfo_tbl[12].prop_cur_bval
#define ips_ip_wroff_extra              ips_propinfo_tbl[13].prop_cur_uval
#define ips_ip_pathmtu_interval         ips_propinfo_tbl[14].prop_cur_uval
#define ips_ip_icmp_return              ips_propinfo_tbl[15].prop_cur_uval
```

```
#define ips_ip_path_mtu_discovery       ips_propinfo_tbl[16].prop_cur_bval
#define ips_ip_pmtu_min                 ips_propinfo_tbl[17].prop_cur_uval
#define ips_ip_ignore_redirect          ips_propinfo_tbl[18].prop_cur_bval
#define ips_ip_arp_icmp_error           ips_propinfo_tbl[19].prop_cur_bval
#define ips_ip_broadcast_ttl            ips_propinfo_tbl[20].prop_cur_uval
#define ips_ip_icmp_err_interval        ips_propinfo_tbl[21].prop_cur_uval
#define ips_ip_icmp_err_burst           ips_propinfo_tbl[22].prop_cur_uval
#define ips_ip_reass_queue_bytes        ips_propinfo_tbl[23].prop_cur_uval
#define ips_ip_strict_dst_multihoming   ips_propinfo_tbl[24].prop_cur_uval
#define ips_ip_addrs_per_if             ips_propinfo_tbl[25].prop_cur_uval
#define ips_ipsec_override_persocket_policy ips_propinfo_tbl[26].prop_cur_bval
#define ips_icmp_accept_clear_messages  ips_propinfo_tbl[27].prop_cur_bval
#define ips_igmp_accept_clear_messages  ips_propinfo_tbl[28].prop_cur_bval

/* IPv6 configuration knobs */
#define ips_delay_first_probe_time      ips_propinfo_tbl[29].prop_cur_uval
#define ips_max_unicast_solicit         ips_propinfo_tbl[30].prop_cur_uval
#define ips_ipv6_def_hops               ips_propinfo_tbl[31].prop_cur_uval
#define ips_ipv6_icmp_return            ips_propinfo_tbl[32].prop_cur_uval
#define ips_ipv6_forward_src_routed     ips_propinfo_tbl[33].prop_cur_bval
#define ips_ipv6_resp_echo_mcast        ips_propinfo_tbl[34].prop_cur_bval
#define ips_ipv6_send_redirects         ips_propinfo_tbl[35].prop_cur_bval
#define ips_ipv6_ignore_redirect        ips_propinfo_tbl[36].prop_cur_bval
#define ips_ipv6_strict_dst_multihoming ips_propinfo_tbl[37].prop_cur_uval
#define ips_src_check                   ips_propinfo_tbl[38].prop_cur_uval
#define ips_ipsec_policy_log_interval   ips_propinfo_tbl[39].prop_cur_uval
#define ips_pim_accept_clear_messages   ips_propinfo_tbl[40].prop_cur_bval
#define ips_ip_ndp_unsolicit_interval   ips_propinfo_tbl[41].prop_cur_uval
#define ips_ip_ndp_unsolicit_count      ips_propinfo_tbl[42].prop_cur_uval
#define ips_ipv6_ignore_home_address_opt ips_propinfo_tbl[43].prop_cur_bval

/* Misc IP configuration knobs */
#define ips_ip_policy_mask              ips_propinfo_tbl[44].prop_cur_uval
#define ips_ip_ecmp_behavior            ips_propinfo_tbl[45].prop_cur_uval
#define ips_ip_multirt_ttl              ips_propinfo_tbl[46].prop_cur_uval
#define ips_ip_ire_badcnt_lifetime      ips_propinfo_tbl[47].prop_cur_uval
#define ips_ip_max_temp_idle            ips_propinfo_tbl[48].prop_cur_uval
#define ips_ip_max_temp_defend          ips_propinfo_tbl[49].prop_cur_uval
#define ips_ip_max_defend               ips_propinfo_tbl[50].prop_cur_uval
#define ips_ip_defend_interval          ips_propinfo_tbl[51].prop_cur_uval
#define ips_ip_dup_recovery             ips_propinfo_tbl[52].prop_cur_uval
#define ips_ip_restrict_interzone_loopback ips_propinfo_tbl[53].prop_cur_bval
#define ips_ip_lso_outbound             ips_propinfo_tbl[54].prop_cur_bval
#define ips_igmp_max_version            ips_propinfo_tbl[55].prop_cur_uval
#define ips_mld_max_version             ips_propinfo_tbl[56].prop_cur_uval
#define ips_ip_forwarding               ips_propinfo_tbl[57].prop_cur_bval
#define ips_ipv6_forwarding             ips_propinfo_tbl[58].prop_cur_bval
#define ips_ip_reassembly_timeout       ips_propinfo_tbl[59].prop_cur_uval
#define ips_ipv6_reassembly_timeout     ips_propinfo_tbl[60].prop_cur_uval
#define ips_ip_cgtp_filter              ips_propinfo_tbl[61].prop_cur_bval
#define ips_arp_probe_delay             ips_propinfo_tbl[62].prop_cur_uval
#define ips_arp_fastprobe_delay         ips_propinfo_tbl[63].prop_cur_uval
#define ips_arp_probe_interval          ips_propinfo_tbl[64].prop_cur_uval
#define ips_arp_fastprobe_interval      ips_propinfo_tbl[65].prop_cur_uval
#define ips_arp_probe_count             ips_propinfo_tbl[66].prop_cur_uval
#define ips_arp_fastprobe_count         ips_propinfo_tbl[67].prop_cur_uval
#define ips_ipv4_dad_announce_interval  ips_propinfo_tbl[68].prop_cur_uval
#define ips_ipv6_dad_announce_interval  ips_propinfo_tbl[69].prop_cur_uval
#define ips_arp_defend_interval         ips_propinfo_tbl[70].prop_cur_uval
#define ips_arp_defend_rate             ips_propinfo_tbl[71].prop_cur_uval
#define ips_ndp_defend_interval         ips_propinfo_tbl[72].prop_cur_uval
#define ips_ndp_defend_rate             ips_propinfo_tbl[73].prop_cur_uval
#define ips_arp_defend_period           ips_propinfo_tbl[74].prop_cur_uval
#define ips_ndp_defend_period           ips_propinfo_tbl[75].prop_cur_uval
#define ips_ipv4_icmp_return_pmtu       ips_propinfo_tbl[76].prop_cur_bval
#define ips_ipv6_icmp_return_pmtu       ips_propinfo_tbl[77].prop_cur_bval
```

```
#define ips_ip_arp_publish_count        ips_propinfo_tbl[78].prop_cur_uval
#define ips_ip_arp_publish_interval     ips_propinfo_tbl[79].prop_cur_uval
#define ips_ip_strict_src_multihoming   ips_propinfo_tbl[80].prop_cur_uval
#define ips_ipv6_strict_src_multihoming ips_propinfo_tbl[81].prop_cur_uval
#define ips_ipv6_drop_inbound_icmpv6    ips_propinfo_tbl[82].prop_cur_bval

extern int      dohwcksum;      /* use h/w cksum if supported by the h/w */
#ifdef ZC_TEST
extern int      noswcksum;
#endif

extern char     ipif_loopback_name[];

extern nv_t     *ire_nv_tbl;

extern struct module_info ip_mod_info;

#define HOOKS4_INTERESTED_PHYSICAL_IN(ipst)     \
        ((ipst)->ips_ip4_physical_in_event.he_interested)
#define HOOKS6_INTERESTED_PHYSICAL_IN(ipst)     \
        ((ipst)->ips_ip6_physical_in_event.he_interested)
#define HOOKS4_INTERESTED_PHYSICAL_OUT(ipst)    \
        ((ipst)->ips_ip4_physical_out_event.he_interested)
#define HOOKS6_INTERESTED_PHYSICAL_OUT(ipst)    \
        ((ipst)->ips_ip6_physical_out_event.he_interested)
#define HOOKS4_INTERESTED_FORWARDING(ipst)      \
        ((ipst)->ips_ip4_forwarding_event.he_interested)
#define HOOKS6_INTERESTED_FORWARDING(ipst)      \
        ((ipst)->ips_ip6_forwarding_event.he_interested)
#define HOOKS4_INTERESTED_LOOPBACK_IN(ipst)     \
        ((ipst)->ips_ip4_loopback_in_event.he_interested)
#define HOOKS6_INTERESTED_LOOPBACK_IN(ipst)     \
        ((ipst)->ips_ip6_loopback_in_event.he_interested)
#define HOOKS4_INTERESTED_LOOPBACK_OUT(ipst)    \
        ((ipst)->ips_ip4_loopback_out_event.he_interested)
#define HOOKS6_INTERESTED_LOOPBACK_OUT(ipst)    \
        ((ipst)->ips_ip6_loopback_out_event.he_interested)
/*
 * Hooks marcos used inside of ip
 * The callers use the above INTERESTED macros first, hence
 * the he_interested check is superflous.
 */
#define FW_HOOKS(_hook, _event, _ilp, _olp, _iph, _fm, _m, _llm, ipst, _err) \
        if ((_hook).he_interested) {                                    \
                hook_pkt_event_t info;                                  \
                                                                        \
                _NOTE(CONSTCOND)                                        \
                ASSERT((_ilp != NULL) || (_olp != NULL));               \
                                                                        \
                FW_SET_ILL_INDEX(info.hpe_ifp, (ill_t *)_ilp);          \
                FW_SET_ILL_INDEX(info.hpe_ofp, (ill_t *)_olp);          \
                info.hpe_protocol = ipst->ips_ipv4_net_data;            \
                info.hpe_hdr = _iph;                                    \
                info.hpe_mp = &(_fm);                                   \
                info.hpe_mb = _m;                                       \
                info.hpe_flags = _llm;                                  \
                _err = hook_run(ipst->ips_ipv4_net_data->netd_hooks,    \
                    _event, (hook_data_t)&info);                        \
                if (_err != 0) {                                        \
                        ip2dbg(("%s hook dropped mblk chain %p hdr %p\n",\
                            (_hook).he_name, (void *)_fm, (void *)_m)); \
                        if (_fm != NULL) {                              \
                                freemsg(_fm);                           \
                                _fm = NULL;                             \
                        }                                               \
                        _iph = NULL;                                    \
```

```
                        _m = NULL;                                      \
                } else {                                                \
                        _iph = info.hpe_hdr;                            \
                        _m = info.hpe_mb;                               \
                }                                                       \
        }

#define FW_HOOKS6(_hook, _event, _ilp, _olp, _iph, _fm, _m, _llm, ipst, _err) \
        if ((_hook).he_interested) {                                    \
                hook_pkt_event_t info;                                  \
                                                                        \
                _NOTE(CONSTCOND)                                        \
                ASSERT((_ilp != NULL) || (_olp != NULL));               \
                                                                        \
                FW_SET_ILL_INDEX(info.hpe_ifp, (ill_t *)_ilp);          \
                FW_SET_ILL_INDEX(info.hpe_ofp, (ill_t *)_olp);          \
                info.hpe_protocol = ipst->ips_ipv6_net_data;            \
                info.hpe_hdr = _iph;                                    \
                info.hpe_mp = &(_fm);                                   \
                info.hpe_mb = _m;                                       \
                info.hpe_flags = _llm;                                  \
                _err = hook_run(ipst->ips_ipv6_net_data->netd_hooks,    \
                    _event, (hook_data_t)&info);                        \
                if (_err != 0) {                                        \
                        ip2dbg(("%s hook dropped mblk chain %p hdr %p\n",\
                            (_hook).he_name, (void *)_fm, (void *)_m)); \
                        if (_fm != NULL) {                              \
                                freemsg(_fm);                           \
                                _fm = NULL;                             \
                        }                                               \
                        _iph = NULL;                                    \
                        _m = NULL;                                      \
                } else {                                                \
                        _iph = info.hpe_hdr;                            \
                        _m = info.hpe_mb;                               \
                }                                                       \
        }

#define FW_SET_ILL_INDEX(fp, ill)                                       \
        _NOTE(CONSTCOND)                                                \
        if ((ill) == NULL || (ill)->ill_phyint == NULL) {               \
                (fp) = 0;                                               \
                _NOTE(CONSTCOND)                                        \
        } else if (IS_UNDER_IPMP(ill)) {                                \
                (fp) = ipmp_ill_get_ipmp_ifindex(ill);                  \
        } else {                                                        \
                (fp) = (ill)->ill_phyint->phyint_ifindex;               \
        }

/*
 * Network byte order macros
 */
#ifdef  _BIG_ENDIAN
#define N_IN_CLASSA_NET         IN_CLASSA_NET
#define N_IN_CLASSD_NET         IN_CLASSD_NET
#define N_INADDR_UNSPEC_GROUP   INADDR_UNSPEC_GROUP
#define N_IN_LOOPBACK_NET       (ipaddr_t)0x7f000000U
#else /* _BIG_ENDIAN */
#define N_IN_CLASSA_NET         (ipaddr_t)0x000000ffU
#define N_IN_CLASSD_NET         (ipaddr_t)0x000000f0U
#define N_INADDR_UNSPEC_GROUP   (ipaddr_t)0x000000e0U
#define N_IN_LOOPBACK_NET       (ipaddr_t)0x0000007fU
#endif /* _BIG_ENDIAN */
#define CLASSD(addr)    (((addr) & N_IN_CLASSD_NET) == N_INADDR_UNSPEC_GROUP)
#define CLASSE(addr)    (((addr) & N_IN_CLASSD_NET) == N_IN_CLASSD_NET)
#define IP_LOOPBACK_ADDR(addr)                  \
        (((addr) & N_IN_CLASSA_NET == N_IN_LOOPBACK_NET))

extern int      ip_debug;
extern uint_t   ip_thread_data;
extern krwlock_t ip_thread_rwlock;
extern list_t   ip_thread_list;

#ifdef IP_DEBUG
#include <sys/debug.h>
#include <sys/promif.h>

#define ip0dbg(a)       printf a
#define ip1dbg(a)       if (ip_debug > 2) printf a
#define ip2dbg(a)       if (ip_debug > 3) printf a
#define ip3dbg(a)       if (ip_debug > 4) printf a
#else
#define ip0dbg(a)       /* */
#define ip1dbg(a)       /* */
#define ip2dbg(a)       /* */
#define ip3dbg(a)       /* */
#endif  /* IP_DEBUG */

/* Default MAC-layer address string length for mac_colon_addr */
#define MAC_STR_LEN     128

struct  mac_header_info_s;

extern void     ill_frag_timer(void *);
extern ill_t    *ill_first(int, int, ill_walk_context_t *, ip_stack_t *);
extern ill_t    *ill_next(ill_walk_context_t *, ill_t *);
extern void     ill_frag_timer_start(ill_t *);
extern void     ill_nic_event_dispatch(ill_t *, lif_if_t, nic_event_t,
    nic_event_data_t, size_t);
extern mblk_t   *ip_carve_mp(mblk_t **, ssize_t);
extern mblk_t   *ip_dlpi_alloc(size_t, t_uscalar_t);
extern mblk_t   *ip_dlnotify_alloc(uint_t, uint_t);
extern mblk_t   *ip_dlnotify_alloc2(uint_t, uint_t, uint_t);
extern char     *ip_dot_addr(ipaddr_t, char *);
extern const char *mac_colon_addr(const uint8_t *, size_t, char *, size_t);
extern void     ip_lwput(queue_t *, mblk_t *);
extern boolean_t icmp_err_rate_limit(ip_stack_t *);
extern void     icmp_frag_needed(mblk_t *, int, ip_recv_attr_t *);
extern mblk_t   *icmp_inbound_v4(mblk_t *, ip_recv_attr_t *);
extern void     icmp_time_exceeded(mblk_t *, uint8_t, ip_recv_attr_t *);
extern void     icmp_unreachable(mblk_t *, uint8_t, ip_recv_attr_t *);
extern boolean_t ip_ipsec_policy_inherit(conn_t *, conn_t *, ip_recv_attr_t *);
extern void     *ip_pullup(mblk_t *, ssize_t, ip_recv_attr_t *);
extern void     ip_setl2src(mblk_t *, ip_recv_attr_t *, ill_t *);
extern mblk_t   *ip_check_and_align_header(mblk_t *, uint_t, ip_recv_attr_t *);
extern mblk_t   *ip_check_length(mblk_t *, uchar_t *, ssize_t, uint_t, uint_t,
    ip_recv_attr_t *);
extern mblk_t   *ip_check_optlen(mblk_t *, ipha_t *, uint_t, uint_t,
    ip_recv_attr_t *);
extern mblk_t   *ip_fix_dbref(mblk_t *, ip_recv_attr_t *);
extern uint_t   ip_cksum(mblk_t *, int, uint32_t);
extern int      ip_close(queue_t *, int);
extern uint16_t ip_csum_hdr(ipha_t *);
extern void     ip_forward_xmit_v4(nce_t *, ill_t *, mblk_t *, ipha_t *,
    ip_recv_attr_t *, uint32_t, uint32_t);
extern boolean_t ip_forward_options(mblk_t *, ipha_t *, ill_t *,
    ip_recv_attr_t *);
extern int      ip_fragment_v4(mblk_t *, nce_t *, iaflags_t, uint_t, uint32_t,
    uint32_t, zoneid_t, zoneid_t, pfirepostfrag_t postfragfn,
    uintptr_t *cookie);
extern void     ip_proto_not_sup(mblk_t *, ip_recv_attr_t *);
extern void     ip_ire_g_fini(void);
```

```
extern void     ip_ire_g_init(void);
extern void     ip_ire_fini(ip_stack_t *);
extern void     ip_ire_init(ip_stack_t *);
extern void     ip_mdata_to_mhi(ill_t *, mblk_t *, struct mac_header_info_s *);
extern int      ip_openv4(queue_t *q, dev_t *devp, int flag, int sflag,
                    cred_t *credp);
extern int      ip_openv6(queue_t *q, dev_t *devp, int flag, int sflag,
                    cred_t *credp);
extern int      ip_reassemble(mblk_t *, ipf_t *, uint_t, boolean_t, ill_t *,
    size_t);
extern void     ip_rput(queue_t *, mblk_t *);
extern void     ip_input(ill_t *, ill_rx_ring_t *, mblk_t *,
    struct mac_header_info_s *);
extern void     ip_input_v6(ill_t *, ill_rx_ring_t *, mblk_t *,
    struct mac_header_info_s *);
extern mblk_t   *ip_input_common_v4(ill_t *, ill_rx_ring_t *, mblk_t *,
    struct mac_header_info_s *, squeue_t *, mblk_t **, uint_t *);
extern mblk_t   *ip_input_common_v6(ill_t *, ill_rx_ring_t *, mblk_t *,
    struct mac_header_info_s *, squeue_t *, mblk_t **, uint_t *);
extern void     ill_input_full_v4(mblk_t *, void *, void *,
    ip_recv_attr_t *, rtc_t *);
extern void     ill_input_short_v4(mblk_t *, void *, void *,
    ip_recv_attr_t *, rtc_t *);
extern void     ill_input_full_v6(mblk_t *, void *, void *,
    ip_recv_attr_t *, rtc_t *);
extern void     ill_input_short_v6(mblk_t *, void *, void *,
    ip_recv_attr_t *, rtc_t *);
extern ipaddr_t ip_input_options(ipha_t *, ipaddr_t, mblk_t *,
    ip_recv_attr_t *, int *);
extern boolean_t ip_input_local_options(mblk_t *, ipha_t *, ip_recv_attr_t *);
extern mblk_t   *ip_input_fragment(mblk_t *, ipha_t *, ip_recv_attr_t *);
extern mblk_t   *ip_input_fragment_v6(mblk_t *, ip6_t *, ip6_frag_t *, uint_t,
    ip_recv_attr_t *);
extern void     ip_input_post_ipsec(mblk_t *, ip_recv_attr_t *);
extern void     ip_fanout_v4(mblk_t *, ipha_t *, ip_recv_attr_t *);
extern void     ip_fanout_v6(mblk_t *, ip6_t *, ip_recv_attr_t *);
extern void     ip_fanout_proto_conn(conn_t *, mblk_t *, ipha_t *, ip6_t *,
    ip_recv_attr_t *);
extern void     ip_fanout_proto_v4(mblk_t *, ipha_t *, ip_recv_attr_t *);
extern void     ip_fanout_send_icmp_v4(mblk_t *, uint_t, uint_t,
    ip_recv_attr_t *);
extern void     ip_fanout_udp_conn(conn_t *, mblk_t *, ipha_t *, ip6_t *,
    ip_recv_attr_t *);
extern void     ip_fanout_udp_multi_v4(mblk_t *, ipha_t *, uint16_t, uint16_t,
    ip_recv_attr_t *);
extern mblk_t   *zero_spi_check(mblk_t *, ip_recv_attr_t *);
extern void     ip_build_hdrs_v4(uchar_t *, uint_t, const ip_pkt_t *, uint8_t);
extern int      ip_find_hdr_v4(ipha_t *, ip_pkt_t *, boolean_t);
extern int      ip_total_hdrs_len_v4(const ip_pkt_t *);

extern mblk_t   *ip_accept_tcp(ill_t *, ill_rx_ring_t *, squeue_t *,
    mblk_t *, mblk_t **, uint_t *cnt);
extern void     ip_rput_dlpi(ill_t *, mblk_t *);
extern void     ip_rput_notdata(ill_t *, mblk_t *);

extern void     ip_mib2_add_ip_stats(mib2_ipIfStatsEntry_t *,
                    mib2_ipIfStatsEntry_t *);
extern void     ip_mib2_add_icmp6_stats(mib2_ipv6IfIcmpEntry_t *,
                    mib2_ipv6IfIcmpEntry_t *);
extern void     ip_rput_other(ipsq_t *, queue_t *, mblk_t *, void *);
extern ire_t    *ip_check_multihome(void *, ire_t *, ill_t *);
extern void     ip_send_potential_redirect_v4(mblk_t *, ipha_t *, ire_t *,
    ip_recv_attr_t *);
extern int      ip_set_destination_v4(ipaddr_t *, ipaddr_t, ipaddr_t,
    ip_xmit_attr_t *, iulp_t *, uint32_t, uint_t);
extern int      ip_set_destination_v6(in6_addr_t *, const in6_addr_t *,
    const in6_addr_t *, ip_xmit_attr_t *, iulp_t *, uint32_t, uint_t);

extern int      ip_output_simple(mblk_t *, ip_xmit_attr_t *);
extern int      ip_output_simple_v4(mblk_t *, ip_xmit_attr_t *);
extern int      ip_output_simple_v6(mblk_t *, ip_xmit_attr_t *);
extern int      ip_output_options(mblk_t *, ipha_t *, ip_xmit_attr_t *,
    ill_t *);
extern void     ip_output_local_options(ipha_t *, ip_stack_t *);

extern ip_xmit_attr_t *conn_get_ixa(conn_t *, boolean_t);
extern ip_xmit_attr_t *conn_get_ixa_tryhard(conn_t *, boolean_t);
extern ip_xmit_attr_t *conn_replace_ixa(conn_t *, ip_xmit_attr_t *);
extern ip_xmit_attr_t *conn_get_ixa_exclusive(conn_t *);
extern ip_xmit_attr_t *ip_xmit_attr_duplicate(ip_xmit_attr_t *);
extern void     ip_xmit_attr_replace_tsl(ip_xmit_attr_t *, ts_label_t *);
extern void     ip_xmit_attr_restore_tsl(ip_xmit_attr_t *, cred_t *);
boolean_t       ip_recv_attr_replace_label(ip_recv_attr_t *, ts_label_t *);
extern void     ixa_inactive(ip_xmit_attr_t *);
extern void     ixa_refrele(ip_xmit_attr_t *);
extern boolean_t ixa_check_drain_insert(conn_t *, ip_xmit_attr_t *);
extern void     ixa_cleanup(ip_xmit_attr_t *);
extern void     ira_cleanup(ip_recv_attr_t *, boolean_t);
extern void     ixa_safe_copy(ip_xmit_attr_t *, ip_xmit_attr_t *);

extern int      conn_ip_output(mblk_t *, ip_xmit_attr_t *);
extern boolean_t ip_output_verify_local(ip_xmit_attr_t *);
extern mblk_t   *ip_output_process_local(mblk_t *, ip_xmit_attr_t *, boolean_t,
    boolean_t, conn_t *);

extern int      conn_opt_get(conn_opt_arg_t *, t_scalar_t, t_scalar_t,
    uchar_t *);
extern int      conn_opt_set(conn_opt_arg_t *, t_scalar_t, t_scalar_t, uint_t,
    uchar_t *, boolean_t, cred_t *);
extern boolean_t        conn_same_as_last_v4(conn_t *, sin_t *);
extern boolean_t        conn_same_as_last_v6(conn_t *, sin6_t *);
extern int      conn_update_label(const conn_t *, const ip_xmit_attr_t *,
    const in6_addr_t *, ip_pkt_t *);

extern int      ip_opt_set_multicast_group(conn_t *, t_scalar_t,
    uchar_t *, boolean_t, boolean_t);
extern int      ip_opt_set_multicast_sources(conn_t *, t_scalar_t,
    uchar_t *, boolean_t, boolean_t);
extern int      conn_getsockname(conn_t *, struct sockaddr *, uint_t *);
extern int      conn_getpeername(conn_t *, struct sockaddr *, uint_t *);

extern int      conn_build_hdr_template(conn_t *, uint_t, uint_t,
    const in6_addr_t *, const in6_addr_t *, uint32_t);
extern mblk_t   *conn_prepend_hdr(ip_xmit_attr_t *, const ip_pkt_t *,
    const in6_addr_t *, const in6_addr_t *, uint8_t, uint32_t, uint_t,
    mblk_t *, uint_t, uint_t, uint32_t *, int *);
extern void     ip_attr_newdst(ip_xmit_attr_t *);
extern void     ip_attr_nexthop(const ip_pkt_t *, const ip_xmit_attr_t *,
    const in6_addr_t *, in6_addr_t *);
extern int      conn_connect(conn_t *, iulp_t *, uint32_t);
extern int      ip_attr_connect(const conn_t *, ip_xmit_attr_t *,
    const in6_addr_t *, const in6_addr_t *, const in6_addr_t *, in_port_t,
    in6_addr_t *, iulp_t *, uint32_t);
extern int      conn_inherit_parent(conn_t *, conn_t *);

extern void     conn_ixa_cleanup(conn_t *connp, void *arg);

extern boolean_t conn_wantpacket(conn_t *, ip_recv_attr_t *, ipha_t *);
extern uint_t   ip_type_v4(ipaddr_t, ip_stack_t *);
extern uint_t   ip_type_v6(const in6_addr_t *, ip_stack_t *);

extern void     ip_wput_nondata(queue_t *, mblk_t *);
```

```
extern void     ip_wsrv(queue_t *);
extern char     *ip_nv_lookup(nv_t *, int);
extern boolean_t ip_local_addr_ok_v6(const in6_addr_t *, const in6_addr_t *);
extern boolean_t ip_remote_addr_ok_v6(const in6_addr_t *, const in6_addr_t *);
extern ipaddr_t ip_massage_options(ipha_t *, netstack_t *);
extern ipaddr_t ip_net_mask(ipaddr_t);
extern void     arp_bringup_done(ill_t *, int);
extern void     arp_replumb_done(ill_t *, int);

extern struct qinit iprinitv6;

extern void     ipmp_init(ip_stack_t *);
extern void     ipmp_destroy(ip_stack_t *);
extern ipmp_grp_t *ipmp_grp_create(const char *, phyint_t *);
extern void     ipmp_grp_destroy(ipmp_grp_t *);
extern void     ipmp_grp_info(const ipmp_grp_t *, lifgroupinfo_t *);
extern int      ipmp_grp_rename(ipmp_grp_t *, const char *);
extern ipmp_grp_t *ipmp_grp_lookup(const char *, ip_stack_t *);
extern int      ipmp_grp_vet_phyint(ipmp_grp_t *, phyint_t *);
extern ipmp_illgrp_t *ipmp_illgrp_create(ill_t *);
extern void     ipmp_illgrp_destroy(ipmp_illgrp_t *);
extern ill_t    *ipmp_illgrp_add_ipif(ipmp_illgrp_t *, ipif_t *);
extern void     ipmp_illgrp_del_ipif(ipmp_illgrp_t *, ipif_t *);
extern ill_t    *ipmp_illgrp_next_ill(ipmp_illgrp_t *);
extern ill_t    *ipmp_illgrp_hold_next_ill(ipmp_illgrp_t *);
extern ill_t    *ipmp_illgrp_hold_cast_ill(ipmp_illgrp_t *);
extern ill_t    *ipmp_illgrp_ipmp_ill(ipmp_illgrp_t *);
extern void     ipmp_illgrp_refresh_mtu(ipmp_illgrp_t *);
extern ipmp_arpent_t *ipmp_illgrp_create_arpent(ipmp_illgrp_t *,
    boolean_t, ipaddr_t, uchar_t *, size_t, uint16_t);
extern void     ipmp_illgrp_destroy_arpent(ipmp_illgrp_t *, ipmp_arpent_t *);
extern ipmp_arpent_t *ipmp_illgrp_lookup_arpent(ipmp_illgrp_t *, ipaddr_t *);
extern void     ipmp_illgrp_refresh_arpent(ipmp_illgrp_t *);
extern void     ipmp_illgrp_mark_arpent(ipmp_illgrp_t *, ipmp_arpent_t *);
extern ill_t    *ipmp_illgrp_find_ill(ipmp_illgrp_t *, uchar_t *, uint_t);
extern void     ipmp_illgrp_link_grp(ipmp_illgrp_t *, ipmp_grp_t *);
extern int      ipmp_illgrp_unlink_grp(ipmp_illgrp_t *);
extern uint_t   ipmp_ill_get_ipmp_ifindex(const ill_t *);
extern void     ipmp_ill_join_illgrp(ill_t *, ipmp_illgrp_t *);
extern void     ipmp_ill_leave_illgrp(ill_t *);
extern ill_t    *ipmp_ill_hold_ipmp_ill(ill_t *);
extern ill_t    *ipmp_ill_hold_xmit_ill(ill_t *, boolean_t);
extern boolean_t ipmp_ill_is_active(ill_t *);
extern void     ipmp_ill_refresh_active(ill_t *);
extern void     ipmp_phyint_join_grp(phyint_t *, ipmp_grp_t *);
extern void     ipmp_phyint_leave_grp(phyint_t *);
extern void     ipmp_phyint_refresh_active(phyint_t *);
extern ill_t    *ipmp_ipif_bound_ill(const ipif_t *);
extern ill_t    *ipmp_ipif_hold_bound_ill(const ipif_t *);
extern boolean_t ipmp_ipif_is_dataaddr(const ipif_t *);
extern boolean_t ipmp_ipif_is_stubaddr(const ipif_t *);
extern boolean_t ipmp_packet_is_probe(mblk_t *, ill_t *);
extern void     ipmp_ncec_delete_nce(ncec_t *);
extern void     ipmp_ncec_refresh_nce(ncec_t *);

extern void     conn_drain_insert(conn_t *, idl_tx_list_t *);
extern void     conn_setqfull(conn_t *, boolean_t *);
extern void     conn_clrqfull(conn_t *, boolean_t *);
extern int      conn_ipsec_length(conn_t *);
extern ipaddr_t ip_get_dst(ipha_t *);
extern uint_t   ip_get_pmtu(ip_xmit_attr_t *);
extern uint_t   ip_get_base_mtu(ill_t *, ire_t *);
extern mblk_t *ip_output_attach_policy(mblk_t *, ipha_t *, ip6_t *,
    const conn_t *, ip_xmit_attr_t *);
extern int      ipsec_out_extra_length(ip_xmit_attr_t *);
extern int      ipsec_out_process(mblk_t *, ip_xmit_attr_t *);
```

```c
extern int      ip_output_post_ipsec(mblk_t *, ip_xmit_attr_t *);
extern void     ipsec_out_to_in(ip_xmit_attr_t *, ill_t *ill,
    ip_recv_attr_t *);

extern void     ire_cleanup(ire_t *);
extern void     ire_inactive(ire_t *);
extern boolean_t irb_inactive(irb_t *);
extern ire_t    *ire_unlink(irb_t *);

#ifdef DEBUG
extern  boolean_t th_trace_ref(const void *, ip_stack_t *);
extern  void    th_trace_unref(const void *);
extern  void    th_trace_cleanup(const void *, boolean_t);
extern  void    ire_trace_ref(ire_t *);
extern  void    ire_untrace_ref(ire_t *);
#endif

extern int      ip_srcid_insert(const in6_addr_t *, zoneid_t, ip_stack_t *);
extern int      ip_srcid_remove(const in6_addr_t *, zoneid_t, ip_stack_t *);
extern void     ip_srcid_find_id(uint_t, in6_addr_t *, zoneid_t, netstack_t *);
extern uint_t   ip_srcid_find_addr(const in6_addr_t *, zoneid_t, netstack_t *);

extern uint8_t  ipoptp_next(ipoptp_t *);
extern uint8_t  ipoptp_first(ipoptp_t *, ipha_t *);
extern int      ip_opt_get_user(conn_t *, uchar_t *);
extern int      ipsec_req_from_conn(conn_t *, ipsec_req_t *, int);
extern int      ip_snmp_get(queue_t *q, mblk_t *mctl, int level, boolean_t);
extern int      ip_snmp_set(queue_t *q, int, int, uchar_t *, int);
extern void     ip_process_ioctl(ipsq_t *, queue_t *, mblk_t *, void *);
extern void     ip_quiesce_conn(conn_t *);
extern  void    ip_reprocess_ioctl(ipsq_t *, queue_t *, mblk_t *, void *);
extern void     ip_ioctl_finish(queue_t *, mblk_t *, int, int, ipsq_t *);

extern boolean_t ip_cmpbuf(const void *, uint_t, boolean_t, const void *,
    uint_t);
extern boolean_t ip_allocbuf(void **, uint_t *, boolean_t, const void *,
    uint_t);
extern void     ip_savebuf(void **, uint_t *, boolean_t, const void *, uint_t);

extern boolean_t        ipsq_pending_mp_cleanup(ill_t *, conn_t *);
extern void     conn_ioctl_cleanup(conn_t *);

extern void     ip_unbind(conn_t *);

extern void tnet_init(void);
extern void tnet_fini(void);

/*
 * Hook functions to enable cluster networking
 * On non-clustered systems these vectors must always be NULL.
 */
extern int (*cl_inet_isclusterwide)(netstackid_t stack_id, uint8_t protocol,
    sa_family_t addr_family, uint8_t *laddrp, void *args);
extern uint32_t (*cl_inet_ipident)(netstackid_t stack_id, uint8_t protocol,
    sa_family_t addr_family, uint8_t *laddrp, uint8_t *faddrp,
    void *args);
extern int (*cl_inet_connect2)(netstackid_t stack_id, uint8_t protocol,
    boolean_t is_outgoing, sa_family_t addr_family, uint8_t *laddrp,
    in_port_t lport, uint8_t *faddrp, in_port_t fport, void *args);
extern void (*cl_inet_getspi)(netstackid_t, uint8_t, uint8_t *, size_t,
    void *);
extern void (*cl_inet_getspi)(netstackid_t stack_id, uint8_t protocol,
    uint8_t *ptr, size_t len, void *args);
extern int (*cl_inet_checkspi)(netstackid_t stack_id, uint8_t protocol,
    uint32_t spi, void *args);
extern void (*cl_inet_deletespi)(netstackid_t stack_id, uint8_t protocol,
```

```c
    uint32_t spi, void *args);
extern void (*cl_inet_idlesa)(netstackid_t, uint8_t, uint32_t,
    sa_family_t, in6_addr_t, in6_addr_t, void *);


/* Hooks for CGTP (multirt routes) filtering module */
#define CGTP_FILTER_REV_1       1
#define CGTP_FILTER_REV_2       2
#define CGTP_FILTER_REV_3       3
#define CGTP_FILTER_REV         CGTP_FILTER_REV_3

/* cfo_filter and cfo_filter_v6 hooks return values */
#define CGTP_IP_PKT_NOT_CGTP    0
#define CGTP_IP_PKT_PREMIUM     1
#define CGTP_IP_PKT_DUPLICATE   2

/* Version 3 of the filter interface */
typedef struct cgtp_filter_ops {
        int     cfo_filter_rev;                 /* CGTP_FILTER_REV_3 */
        int     (*cfo_change_state)(netstackid_t, int);
        int     (*cfo_add_dest_v4)(netstackid_t, ipaddr_t, ipaddr_t,
                    ipaddr_t, ipaddr_t);
        int     (*cfo_del_dest_v4)(netstackid_t, ipaddr_t, ipaddr_t);
        int     (*cfo_add_dest_v6)(netstackid_t, in6_addr_t *, in6_addr_t *,
                    in6_addr_t *, in6_addr_t *);
        int     (*cfo_del_dest_v6)(netstackid_t, in6_addr_t *, in6_addr_t *);
        int     (*cfo_filter)(netstackid_t, uint_t, mblk_t *);
        int     (*cfo_filter_v6)(netstackid_t, uint_t, ip6_t *,
                    ip6_frag_t *);
} cgtp_filter_ops_t;

#define CGTP_MCAST_SUCCESS      1

/*
 * The separate CGTP module needs this global symbol so that it
 * can check the version and determine whether to use the old or the new
 * version of the filtering interface.
 */
extern int      ip_cgtp_filter_rev;

extern int      ip_cgtp_filter_supported(void);
extern int      ip_cgtp_filter_register(netstackid_t, cgtp_filter_ops_t *);
extern int      ip_cgtp_filter_unregister(netstackid_t);
extern int      ip_cgtp_filter_is_registered(netstackid_t);

/*
 * rr_ring_state cycles in the order shown below from RR_FREE through
 * RR_FREE_IN_PROG and  back to RR_FREE.
 */
typedef enum {
        RR_FREE,                        /* Free slot */
        RR_SQUEUE_UNBOUND,              /* Ring's squeue is unbound */
        RR_SQUEUE_BIND_INPROG,          /* Ring's squeue bind in progress */
        RR_SQUEUE_BOUND,                /* Ring's squeue bound to cpu */
        RR_FREE_INPROG                  /* Ring is being freed */
} ip_ring_state_t;

#define ILL_MAX_RINGS           256     /* Max num of rx rings we can manage */
#define ILL_POLLING             0x01    /* Polling in use */

/*
 * These functions pointer types are exported by the mac/dls layer.
 * we need to duplicate the definitions here because we cannot
 * include mac/dls header files here.
 */
typedef boolean_t               (*ip_mac_intr_disable_t)(void *);
typedef void                    (*ip_mac_intr_enable_t)(void *);
typedef ip_mac_tx_cookie_t      (*ip_dld_tx_t)(void *, mblk_t *,
    uint64_t, uint16_t);
typedef void                    (*ip_flow_enable_t)(void *, ip_mac_tx_cookie_t);
typedef void                    *(*ip_dld_callb_t)(void *,
    ip_flow_enable_t, void *);
typedef boolean_t               (*ip_dld_fctl_t)(void *, ip_mac_tx_cookie_t);
typedef int                     (*ip_capab_func_t)(void *, uint_t,
    void *, uint_t);

/*
 * POLLING README
 * sq_get_pkts() is called to pick packets from softring in poll mode. It
 * calls rr_rx to get the chain and process it with rr_ip_accept.
 * rr_rx = mac_soft_ring_poll() to pick packets
 * rr_ip_accept = ip_accept_tcp() to process packets
 */

/*
 * XXX: With protocol, service specific squeues, they will have
 * specific acceptor functions.
 */
typedef mblk_t *(*ip_mac_rx_t)(void *, size_t);
typedef mblk_t *(*ip_accept_t)(ill_t *, ill_rx_ring_t *,
    squeue_t *, mblk_t *, mblk_t **, uint_t *);

/*
 * rr_intr_enable, rr_intr_disable, rr_rx_handle, rr_rx:
 * May be accessed while in the squeue AND after checking that SQS_POLL_CAPAB
 * is set.
 *
 * rr_ring_state: Protected by ill_lock.
 */
struct ill_rx_ring {
        ip_mac_intr_disable_t   rr_intr_disable; /* Interrupt disabling func */
        ip_mac_intr_enable_t    rr_intr_enable; /* Interrupt enabling func */
        void                    *rr_intr_handle; /* Handle interrupt funcs */
        ip_mac_rx_t             rr_rx;          /* Driver receive function */
        ip_accept_t             rr_ip_accept;   /* IP accept function */
        void                    *rr_rx_handle;  /* Handle for Rx ring */
        squeue_t                *rr_sqp; /* Squeue the ring is bound to */
        ill_t                   *rr_ill;        /* back pointer to ill */
        ip_ring_state_t         rr_ring_state;  /* State of this ring */
};

/*
 * IP - DLD direct function call capability
 * Suffixes, df - dld function, dh - dld handle,
 * cf - client (IP) function, ch - client handle
 */
typedef struct ill_dld_direct_s {               /* DLD provided driver Tx */
        ip_dld_tx_t             idd_tx_df;      /* str_mdata_fastpath_put */
        void                    *idd_tx_dh;     /* dld_str_t *dsp */
        ip_dld_callb_t          idd_tx_cb_df;   /* mac_tx_srs_notify */
        void                    *idd_tx_cb_dh;  /* mac_client_handle_t *mch */
        ip_dld_fctl_t           idd_tx_fctl_df; /* mac_tx_is_flow_blocked */
        void                    *idd_tx_fctl_dh;        /* mac_client_handle */
} ill_dld_direct_t;

/* IP - DLD polling capability */
typedef struct ill_dld_poll_s {
        ill_rx_ring_t           idp_ring_tbl[ILL_MAX_RINGS];
} ill_dld_poll_t;

/* Describes ill->ill_dld_capab */
struct ill_dld_capab_s {
```

```
        ip_capab_func_t         idc_capab_df;   /* dld_capab_func */
        void                    *idc_capab_dh;  /* dld_str_t *dsp */
        ill_dld_direct_t        idc_direct;
        ill_dld_poll_t          idc_poll;
};

/*
 * IP squeues exports
 */
extern boolean_t        ip_squeue_fanout;

#define IP_SQUEUE_GET(hint) ip_squeue_random(hint)

extern void ip_squeue_init(void (*)(squeue_t *));
extern squeue_t *ip_squeue_random(uint_t);
extern squeue_t *ip_squeue_get(ill_rx_ring_t *);
extern squeue_t *ip_squeue_getfree(pri_t);
extern int ip_squeue_cpu_move(squeue_t *, processorid_t);
extern void *ip_squeue_add_ring(ill_t *, void *);
extern void ip_squeue_bind_ring(ill_t *, ill_rx_ring_t *, processorid_t);
extern void ip_squeue_clean_ring(ill_t *, ill_rx_ring_t *);
extern void ip_squeue_quiesce_ring(ill_t *, ill_rx_ring_t *);
extern void ip_squeue_restart_ring(ill_t *, ill_rx_ring_t *);
extern void ip_squeue_clean_all(ill_t *);
extern boolean_t        ip_source_routed(ipha_t *, ip_stack_t *);

extern void tcp_wput(queue_t *, mblk_t *);

extern int      ip_fill_mtuinfo(conn_t *, ip_xmit_attr_t *,
    struct ip6_mtuinfo *);
extern hook_t *ipobs_register_hook(netstack_t *, pfv_t);
extern void ipobs_unregister_hook(netstack_t *, hook_t *);
extern void ipobs_hook(mblk_t *, int, zoneid_t, zoneid_t, const ill_t *,
    ip_stack_t *);
typedef void    (*ipsq_func_t)(ipsq_t *, queue_t *, mblk_t *, void *);

extern void     dce_g_init(void);
extern void     dce_g_destroy(void);
extern void     dce_stack_init(ip_stack_t *);
extern void     dce_stack_destroy(ip_stack_t *);
extern void     dce_cleanup(uint_t, ip_stack_t *);
extern dce_t    *dce_get_default(ip_stack_t *);
extern dce_t    *dce_lookup_pkt(mblk_t *, ip_xmit_attr_t *, uint_t *);
extern dce_t    *dce_lookup_v4(ipaddr_t, ip_stack_t *, uint_t *);
extern dce_t    *dce_lookup_v6(const in6_addr_t *, uint_t, ip_stack_t *,
    uint_t *);
extern dce_t    *dce_lookup_and_add_v4(ipaddr_t, ip_stack_t *);
extern dce_t    *dce_lookup_and_add_v6(const in6_addr_t *, uint_t,
    ip_stack_t *);
extern int      dce_update_uinfo_v4(ipaddr_t, iulp_t *, ip_stack_t *);
extern int      dce_update_uinfo_v6(const in6_addr_t *, uint_t, iulp_t *,
    ip_stack_t *);
extern int      dce_update_uinfo(const in6_addr_t *, uint_t, iulp_t *,
    ip_stack_t *);
extern void     dce_increment_generation(dce_t *);
extern void     dce_increment_all_generations(boolean_t, ip_stack_t *);
extern void     dce_refrele(dce_t *);
extern void     dce_refhold(dce_t *);
extern void     dce_refrele_notr(dce_t *);
extern void     dce_refhold_notr(dce_t *);
mblk_t          *ip_snmp_get_mib2_ip_dce(queue_t *, mblk_t *, ip_stack_t *ipst);

extern ip_laddr_t ip_laddr_verify_v4(ipaddr_t, zoneid_t,
    ip_stack_t *, boolean_t);
extern ip_laddr_t ip_laddr_verify_v6(const in6_addr_t *, zoneid_t,
    ip_stack_t *, boolean_t, uint_t);
```

```
extern int      ip_laddr_fanout_insert(conn_t *);

extern boolean_t ip_verify_src(mblk_t *, ip_xmit_attr_t *, uint_t *);
extern int      ip_verify_ire(mblk_t *, ip_xmit_attr_t *);

extern mblk_t   *ip_xmit_attr_to_mblk(ip_xmit_attr_t *);
extern boolean_t ip_xmit_attr_from_mblk(mblk_t *, ip_xmit_attr_t *);
extern mblk_t   *ip_xmit_attr_free_mblk(mblk_t *);
extern mblk_t   *ip_recv_attr_to_mblk(ip_recv_attr_t *);
extern boolean_t ip_recv_attr_from_mblk(mblk_t *, ip_recv_attr_t *);
extern mblk_t   *ip_recv_attr_free_mblk(mblk_t *);
extern boolean_t ip_recv_attr_is_mblk(mblk_t *);

extern char     *inet_ntop(int, const void *, char *, int);
extern int      _inet_pton(int, char *, void *);
#define inet_pton(x, y, z)      _inet_pton(x, y, z)

#endif /* ! codereview */
/*
 * Squeue tags. Tags only need to be unique when the callback function is the
 * same to distinguish between different calls, but we use unique tags for
 * convenience anyway.
 */
#define SQTAG_IP_INPUT                  1
#define SQTAG_TCP_INPUT_ICMP_ERR        2
#define SQTAG_TCP6_INPUT_ICMP_ERR       3
#define SQTAG_IP_TCP_INPUT              4
#define SQTAG_IP6_TCP_INPUT             5
#define SQTAG_IP_TCP_CLOSE              6
#define SQTAG_TCP_OUTPUT                7
#define SQTAG_TCP_TIMER                 8
#define SQTAG_TCP_TIMEWAIT              9
#define SQTAG_TCP_ACCEPT_FINISH         10
#define SQTAG_TCP_ACCEPT_FINISH_Q0      11
#define SQTAG_TCP_ACCEPT_PENDING        12
#define SQTAG_TCP_LISTEN_DISCON         13
#define SQTAG_TCP_CONN_REQ_1            14
#define SQTAG_TCP_EAGER_BLOWOFF         15
#define SQTAG_TCP_EAGER_CLEANUP         16
#define SQTAG_TCP_EAGER_CLEANUP_Q0      17
#define SQTAG_TCP_CONN_IND              18
#define SQTAG_TCP_RSRV                  19
#define SQTAG_TCP_ABORT_BUCKET          20
#define SQTAG_TCP_REINPUT               21
#define SQTAG_TCP_REINPUT_EAGER         22
#define SQTAG_TCP_INPUT_MCTL            23
#define SQTAG_TCP_RPUTOTHER             24
#define SQTAG_IP_PROTO_AGAIN            25
#define SQTAG_IP_FANOUT_TCP             26
#define SQTAG_IPSQ_CLEAN_RING           27
#define SQTAG_TCP_WPUT_OTHER            28
#define SQTAG_TCP_CONN_REQ_UNBOUND      29
#define SQTAG_TCP_SEND_PENDING          30
#define SQTAG_BIND_RETRY                31
#define SQTAG_UDP_FANOUT                32
#define SQTAG_UDP_INPUT                 33
#define SQTAG_UDP_WPUT                  34
#define SQTAG_UDP_OUTPUT                35
#define SQTAG_TCP_KSSL_INPUT            36
#define SQTAG_TCP_DROP_Q0               37
#define SQTAG_TCP_CONN_REQ_2            38
#define SQTAG_IP_INPUT_RX_RING          39
#define SQTAG_SQUEUE_CHANGE             40
#define SQTAG_CONNECT_FINISH            41
#define SQTAG_SYNCHRONOUS_OP            42
#define SQTAG_TCP_SHUTDOWN_OUTPUT       43
```

```
#define SQTAG_TCP_IXA_CLEANUP           44
#define SQTAG_TCP_SEND_SYNACK           45

extern sin_t    sin_null;       /* Zero address for quick clears */
extern sin6_t   sin6_null;      /* Zero address for quick clears */

#endif  /* _KERNEL */

#ifdef  __cplusplus
}
#endif

#endif  /* _INET_IP_H */
```

```
**********************************************************
   10608 Wed Sep 26 12:51:44 2012
new/usr/src/uts/common/inet/ip/inet_ntop.c
inet_pton
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/cmn_err.h>
#include <sys/systm.h>
#include <sys/socket.h>
#include <sys/sunddi.h>
#include <netinet/in.h>
#include <inet/led.h>

static void     convert2ascii(char *, const in6_addr_t *);
static char     *strchr_w(const char *, int);
static int      str2inet_addr(char *, ipaddr_t *);

/*
 * inet_ntop -- Convert an IPv4 or IPv6 address in binary form into
 * printable form, and return a pointer to that string. Caller should
 * provide a buffer of correct length to store string into.
 * Note: this routine is kernel version of inet_ntop. It has similar
 * format as inet_ntop() defined in rfc2553. But it does not do
 * error handling operations exactly as rfc2553 defines. This function
 * is used by kernel inet directory routines only for debugging.
 * This inet_ntop() function, does not return NULL if third argument
 * is NULL. The reason is simple that we don't want kernel to panic
 * as the output of this function is directly fed to ip<n>dbg macro.
 * Instead it uses a local buffer for destination address for
 * those calls which purposely pass NULL ptr for the destination
 * buffer. This function is thread-safe when the caller passes a non-
 * null buffer with the third argument.
 */
/* ARGSUSED */
char *
```

```
inet_ntop(int af, const void *addr, char *buf, int addrlen)
{
        static char local_buf[INET6_ADDRSTRLEN];
        static char *err_buf1 = "<badaddr>";
        static char *err_buf2 = "<badfamily>";
        in6_addr_t      *v6addr;
        uchar_t         *v4addr;
        char            *caddr;

        /*
         * We don't allow thread unsafe inet_ntop calls, they
         * must pass a non-null buffer pointer. For DEBUG mode
         * we use the ASSERT() and for non-debug kernel it will
         * silently allow it for now. Someday we should remove
         * the static buffer from this function.
         */

        ASSERT(buf != NULL);
        if (buf == NULL)
                buf = local_buf;
        buf[0] = '\0';

        /* Let user know politely not to send NULL or unaligned addr */
        if (addr == NULL || !(OK_32PTR(addr))) {
#ifdef DEBUG
                cmn_err(CE_WARN, "inet_ntop: addr is <null> or unaligned");
#endif
                return (err_buf1);
        }


#define UC(b)   (((int)b) & 0xff)
        switch (af) {
        case AF_INET:
                ASSERT(addrlen >= INET_ADDRSTRLEN);
                v4addr = (uchar_t *)addr;
                (void) sprintf(buf, "%03d.%03d.%03d.%03d",
                    UC(v4addr[0]), UC(v4addr[1]), UC(v4addr[2]), UC(v4addr[3]));
                return (buf);

        case AF_INET6:
                ASSERT(addrlen >= INET6_ADDRSTRLEN);
                v6addr = (in6_addr_t *)addr;
                if (IN6_IS_ADDR_V4MAPPED(v6addr)) {
                        caddr = (char *)addr;
                        (void) sprintf(buf, "::ffff:%d.%d.%d.%d",
                            UC(caddr[12]), UC(caddr[13]),
                            UC(caddr[14]), UC(caddr[15]));
                } else if (IN6_IS_ADDR_V4COMPAT(v6addr)) {
                        caddr = (char *)addr;
                        (void) sprintf(buf, "::%d.%d.%d.%d",
                            UC(caddr[12]), UC(caddr[13]), UC(caddr[14]),
                            UC(caddr[15]));
                } else if (IN6_IS_ADDR_UNSPECIFIED(v6addr)) {
                        (void) sprintf(buf, "::");
                } else {
                        convert2ascii(buf, v6addr);
                }
                return (buf);

        default:
                return (err_buf2);
        }
#undef UC
}
______unchanged_portion_omitted_
```

```
static int
str2inet_addr(char *cp, ipaddr_t *addrp)
{
        char *end;
        long byte;
        int i;
        uint8_t *addr = (uint8_t *)addrp;

        *addrp = 0;
        ipaddr_t addr = 0;

        for (i = 0; i < 4; i++) {
                if (ddi_strtol(cp, &end, 10, &byte) != 0 || byte < 0 ||
                    byte > 255) {
                        return (0);
                }
                addr[i] = (uint8_t)byte;
                addr = (addr << 8) | (uint8_t)byte;
                if (i < 3) {
                        if (*end != '.') {
                                return (0);
                        } else {
                                cp = end + 1;
                        }
                } else {
                        cp = end;
                }
        }

        *addrp = addr;
        return (1);
}

/*
 * inet_pton: This function takes string format IPv4 or IPv6 address and
 * converts it to binary form. The format of this function corresponds to
 * inet_pton() in the socket library.
 * It returns 0 for invalid IPv4 and IPv6 address
 *            1 when successfully converts ascii to binary
 *            -1 when af is not AF_INET or AF_INET6
 */
int
m_inet_pton(int af, char *inp, void *outp, int revert)
inet_pton(int af, char *inp, void *outp)
{
        int i;
        long byte;
        char *end;

        switch (af) {
        case AF_INET:
                if (str2inet_addr(inp, (ipaddr_t *)outp)) {
                        if (! revert)
                                *(uint32_t *)outp = ntohl(*(uint32_t *)outp);
                        return (1);
                } else {
                        return (0);
                }
                return (str2inet_addr(inp, (ipaddr_t *)outp));
        case AF_INET6: {
                union v6buf_u {
                        uint16_t v6words_u[8];
                        in6_addr_t v6addr_u;
                } v6buf, *v6outp;
                uint16_t        *dbl_col = NULL;
```

```
                char lastbyte = NULL;

                v6outp = (union v6buf_u *)outp;

                if (strchr_w(inp, '.') != NULL) {
                        /* v4 mapped or v4 compatable */
                        if (strncmp(inp, "::ffff:", 7) == 0) {
                                ipaddr_t ipv4_all_zeroes = 0;
                                /* mapped - first init prefix and then fill */
                                IN6_IPADDR_TO_V4MAPPED(ipv4_all_zeroes,
                                    &v6outp->v6addr_u);
                                return (str2inet_addr(inp + 7,
                                    &(v6outp->v6addr_u.s6_addr32[3])));
                        } else if (strncmp(inp, "::", 2) == 0) {
                                /* v4 compatable - prefix all zeroes */
                                bzero(&v6outp->v6addr_u, sizeof (in6_addr_t));
                                return (str2inet_addr(inp + 2,
                                    &(v6outp->v6addr_u.s6_addr32[3])));
                        }
                        return (0);
                }
                for (i = 0; i < 8; i++) {
                        int error;
                        /*
                         * if ddi_strtol() fails it could be because
                         * the string is "::".  That is valid and
                         * checked for below so just set the value to
                         * 0 and continue.
                         */
                        if ((error = ddi_strtol(inp, &end, 16, &byte)) != 0) {
                                if (error == ERANGE)
                                        return (0);
                                byte = 0;
                        }
                        if (byte < 0 || byte > 0x0ffff) {
                                return (0);
                        }
                        if (revert) {
                                v6buf.v6words_u[i] = htons((uint16_t)byte);
                        } else {
#endif /* ! codereview */
                                v6buf.v6words_u[i] = (uint16_t)byte;
                        }
#endif /* ! codereview */
                        if (*end == NULL || i == 7) {
                                inp = end;
                                break;
                        }
                        if (inp == end) {       /* not a number must be */
                                if (*inp == ':' &&
                                    ((i == 0 && *(inp + 1) == ':') ||
                                    lastbyte == ':')) {
                                        if (dbl_col) {
                                                return (0);
                                        }
                                        if (byte != 0)
                                                i++;
                                        dbl_col = &v6buf.v6words_u[i];
                                        if (i == 0)
                                                inp++;
                                } else if (*inp == NULL || *inp == ' ' ||
                                    *inp == '\t') {
                                        break;
                                } else {
                                        return (0);
                                }
                        } else {
                                inp = end;
                        }
                        if (*inp != ’:’) {
                                return (0);
                        }
                        inp++;
                        if (*inp == NULL || *inp == ’ ’ || *inp == ’\t’) {
                                break;
                        }
                        lastbyte = *inp;
                }
                if (*inp != NULL && *inp != ’ ’ && *inp != ’\t’) {
                        return (0);
                }
                /*
                 * v6words now contains the bytes we could translate
                 * dbl_col points to the word (should be 0) where
                 * a double colon was found
                 */
                if (i == 7) {
                        v6outp->v6addr_u = v6buf.v6addr_u;
                } else {
                        int rem;
                        int word;
                        int next;
                        if (dbl_col == NULL) {
                                return (0);
                        }
                        bzero(&v6outp->v6addr_u, sizeof (in6_addr_t));
                        rem = dbl_col - &v6buf.v6words_u[0];
                        for (next = 0; next < rem; next++) {
                                v6outp->v6words_u[next] = v6buf.v6words_u[next];
                        }
                        next++; /* skip dbl_col 0 */
                        rem = i - rem;
                        word = 8 - rem;
                        while (rem > 0) {
                                v6outp->v6words_u[word] = v6buf.v6words_u[next];
                                word++;
                                rem--;
                                next++;
                        }
                }
                return (1);     /* Success */
        }
        }       /* switch */
        return (-1);    /* return -1 for default case */
}


int
_inet_pton(int af, char *inp, void *outp)
{
        return (m_inet_pton(af, inp, outp, 1));
}

/*
 * We need this inet_pton to preserve compatibility with old closed binaries.
 * Earlier, inet_pton returned address in hardware native order,
 * not in network one. (See http://www.illumos.org/issue/3105).
 * Having fixed that, we still need to support binaries, that use bad inet_pton
 * and reverse returned address manually. All new inet_pton calls will be
 * redirected to _inet_pton with #define in the header file.
 */
int
```

```
inet_pton(int af, char *inp, void *outp)
{
        return (m_inet_pton(af, inp, outp, 0));
}
#endif /* ! codereview */
```

```
**********************************************************
   10051 Wed Sep 26 12:51:46 2012
new/usr/src/uts/common/inet/ip6.h
inet_pton
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */
#endif /* ! codereview */

#ifndef _INET_IP6_H
#define _INET_IP6_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/isa_defs.h>

#ifdef  _KERNEL
/* icmp6_t is used in the prototype of icmp_inbound_error_fanout_v6() */
#include <netinet/icmp6.h>
#endif  /* _KERNEL */

/* version number for IPv6 - hard to get this one wrong! */
#define IPV6_VERSION            6

#define IPV6_HDR_LEN            40

#define IPV6_ADDR_LEN           16

/*
 * IPv6 address scopes.  The values of these enums also match the scope
 * field of multicast addresses.
 */
typedef enum {
        IP6_SCOPE_INTFLOCAL = 1,        /* Multicast addresses only */
        IP6_SCOPE_LINKLOCAL,
        IP6_SCOPE_SUBNETLOCAL,          /* Multicast addresses only */
        IP6_SCOPE_ADMINLOCAL,           /* Multicast addresses only */
        IP6_SCOPE_SITELOCAL,
        IP6_SCOPE_GLOBAL
```

```
} in6addr_scope_t;

/* From RFC 3542 - setting for IPV6_USE_MIN_MTU socket option */
#define IPV6_USE_MIN_MTU_MULTICAST      -1      /* Default */
#define IPV6_USE_MIN_MTU_NEVER          0
#define IPV6_USE_MIN_MTU_ALWAYS         1

#ifdef  _KERNEL

/* Extract the scope from a multicast address */
#ifdef _BIG_ENDIAN
#define IN6_ADDR_MC_SCOPE(addr) \
        (((addr)->s6_addr32[0] & 0x000f0000) >> 16)
#else
#define IN6_ADDR_MC_SCOPE(addr) \
        (((addr)->s6_addr32[0] & 0x00000f00) >> 8)
#endif

/* Default IPv4 TTL for IPv6-in-IPv4 encapsulated packets */
#define IPV6_DEFAULT_HOPS       60      /* XXX What should it be? */

/* Max IPv6 TTL */
#define IPV6_MAX_HOPS   255

/* Minimum IPv6 MTU from rfc2460 */
#define IPV6_MIN_MTU            1280

/* EUI-64 based token length */
#define IPV6_TOKEN_LEN          64

/* Length of an advertised IPv6 prefix */
#define IPV6_PREFIX_LEN         64

/* Default and maximum tunnel encapsulation limits.  See RFC 2473. */
#define IPV6_DEFAULT_ENCAPLIMIT 4
#define IPV6_MAX_ENCAPLIMIT     255

/*
 * Minimum and maximum extension header lengths for IPv6.  The 8-bit
 * length field of each extension header (see rfc2460) specifies the
 * number of 8 octet units of data in the header not including the
 * first 8 octets.  A value of 0 would indicate 8 bytes (0 * 8 + 8),
 * and 255 would indicate 2048 bytes (255 * 8 + 8).
 */
#define MIN_EHDR_LEN            8
#define MAX_EHDR_LEN            2048

#ifdef _BIG_ENDIAN
#define IPV6_DEFAULT_VERS_AND_FLOW      0x60000000
#define IPV6_VERS_AND_FLOW_MASK         0xF0000000
#define V6_MCAST                        0xFF000000
#define V6_LINKLOCAL                    0xFE800000

#define IPV6_FLOW_TCLASS(x)             (((x) & IPV6_FLOWINFO_TCLASS) >> 20)
#define IPV6_TCLASS_FLOW(f, c)          (((f) & ~IPV6_FLOWINFO_TCLASS) |\
                                        ((c) << 20))
#else
#define IPV6_DEFAULT_VERS_AND_FLOW      0x00000060
#define IPV6_VERS_AND_FLOW_MASK         0x000000F0
#define V6_MCAST                        0x000000FF
#define V6_LINKLOCAL                    0x000080FE

#define IPV6_FLOW_TCLASS(x)             ((((x) & 0xf000U) >> 12) |\
                                        (((x) & 0xf) << 4))
#define IPV6_TCLASS_FLOW(f, c)          (((f) & ~IPV6_FLOWINFO_TCLASS) |\
                                        ((((c) & 0xf) << 12) |\
```

```c
                                        (((c) & 0xf0) >> 4)))
#endif

/*
 * UTILITY MACROS FOR ADDRESSES.
 */

/*
 * Convert an IPv4 address mask to an IPv6 mask.   Pad with 1-bits.
 */
#define V4MASK_TO_V6(v4, v6)    ((v6).s6_addr32[0] = 0xffffffffUL,      \
                                (v6).s6_addr32[1] = 0xffffffffUL,       \
                                (v6).s6_addr32[2] = 0xffffffffUL,       \
                                (v6).s6_addr32[3] = (v4))

/*
 * Convert aligned IPv4-mapped IPv6 address into an IPv4 address.
 * Note: We use "v6" here in definition of macro instead of "(v6)"
 * Not possible to use "(v6)" here since macro is used with struct
 * field names as arguments.
 */
#define V4_PART_OF_V6(v6)       v6.s6_addr32[3]

#ifdef _BIG_ENDIAN
#define V6_OR_V4_INADDR_ANY(a)  ((a).s6_addr32[3] == 0 &&               \
                                ((a).s6_addr32[2] == 0xffffU || \
                                (a).s6_addr32[2] == 0) &&               \
                                (a).s6_addr32[1] == 0 &&                \
                                (a).s6_addr32[0] == 0)

#else
#define V6_OR_V4_INADDR_ANY(a)  ((a).s6_addr32[3] == 0 &&               \
                                ((a).s6_addr32[2] == 0xffff0000U ||     \
                                (a).s6_addr32[2] == 0) &&               \
                                (a).s6_addr32[1] == 0 &&                \
                                (a).s6_addr32[0] == 0)
#endif /* _BIG_ENDIAN */

/* IPv4-mapped CLASSD addresses */
#ifdef _BIG_ENDIAN
#define IN6_IS_ADDR_V4MAPPED_CLASSD(addr) \
        (((addr)->_S6_un._S6_u32[2] == 0x0000ffff) && \
        (CLASSD((addr)->_S6_un._S6_u32[3])) && \
        ((addr)->_S6_un._S6_u32[1] == 0) && \
        ((addr)->_S6_un._S6_u32[0] == 0))
#else  /* _BIG_ENDIAN */
#define IN6_IS_ADDR_V4MAPPED_CLASSD(addr) \
        (((addr)->_S6_un._S6_u32[2] == 0xffff0000U) && \
        (CLASSD((addr)->_S6_un._S6_u32[3])) && \
        ((addr)->_S6_un._S6_u32[1] == 0) && \
        ((addr)->_S6_un._S6_u32[0] == 0))
#endif /* _BIG_ENDIAN */

/* Clear an IPv6 addr */
#define V6_SET_ZERO(a)          ((a).s6_addr32[0] = 0,                  \
                                (a).s6_addr32[1] = 0,                   \
                                (a).s6_addr32[2] = 0,                   \
                                (a).s6_addr32[3] = 0)

/* Mask comparison: is IPv6 addr a, and'ed with mask m, equal to addr b? */
#define V6_MASK_EQ(a, m, b)                                             \
        ((((a).s6_addr32[0] & (m).s6_addr32[0]) == (b).s6_addr32[0]) && \
        (((a).s6_addr32[1] & (m).s6_addr32[1]) == (b).s6_addr32[1]) &&  \
        (((a).s6_addr32[2] & (m).s6_addr32[2]) == (b).s6_addr32[2]) &&  \
        (((a).s6_addr32[3] & (m).s6_addr32[3]) == (b).s6_addr32[3]))
```

```c
#define V6_MASK_EQ_2(a, m, b)                                           \
        ((((a).s6_addr32[0] & (m).s6_addr32[0]) ==                      \
            ((b).s6_addr32[0]  & (m).s6_addr32[0])) &&                  \
        (((a).s6_addr32[1] & (m).s6_addr32[1]) ==                       \
            ((b).s6_addr32[1]  & (m).s6_addr32[1])) &&                  \
        (((a).s6_addr32[2] & (m).s6_addr32[2]) ==                       \
            ((b).s6_addr32[2]  & (m).s6_addr32[2])) &&                  \
        (((a).s6_addr32[3] & (m).s6_addr32[3]) ==                       \
            ((b).s6_addr32[3]  & (m).s6_addr32[3])))

/* Copy IPv6 address (s), logically and'ed with mask (m), into (d) */
#define V6_MASK_COPY(s, m, d)                                           \
        ((d).s6_addr32[0] = (s).s6_addr32[0] & (m).s6_addr32[0],        \
        (d).s6_addr32[1] = (s).s6_addr32[1] & (m).s6_addr32[1],         \
        (d).s6_addr32[2] = (s).s6_addr32[2] & (m).s6_addr32[2],         \
        (d).s6_addr32[3] = (s).s6_addr32[3] & (m).s6_addr32[3])

#define ILL_FRAG_HASH_V6(v6addr, i)                                     \
        ((ntohl((v6addr).s6_addr32[3]) ^ (i ^ (i >> 8))) %              \
                                                ILL_FRAG_HASH_TBL_COUNT)


/*
 * GLOBAL EXTERNALS
 */
extern const in6_addr_t ipv6_all_ones;
extern const in6_addr_t ipv6_all_zeros;
extern const in6_addr_t ipv6_loopback;
extern const in6_addr_t ipv6_all_hosts_mcast;
extern const in6_addr_t ipv6_all_rtrs_mcast;
extern const in6_addr_t ipv6_all_v2rtrs_mcast;
extern const in6_addr_t ipv6_solicited_node_mcast;
extern const in6_addr_t ipv6_unspecified_group;

/*
 * FUNCTION PROTOTYPES
 */

extern void     convert2ascii(char *buf, const in6_addr_t *addr);
extern char     *inet_ntop(int, const void *, char *, int);
extern int      inet_pton(int, char *, void *);
extern void     icmp_param_problem_nexthdr_v6(mblk_t *, boolean_t,
    ip_recv_attr_t *);
extern void     icmp_pkt2big_v6(mblk_t *, uint32_t, boolean_t,
    ip_recv_attr_t *);
extern void     icmp_time_exceeded_v6(mblk_t *, uint8_t, boolean_t,
    ip_recv_attr_t *);
extern void     icmp_unreachable_v6(mblk_t *, uint8_t, boolean_t,
    ip_recv_attr_t *);
extern mblk_t   *icmp_inbound_v6(mblk_t *, ip_recv_attr_t *);
extern void     icmp_inbound_error_fanout_v6(mblk_t *, icmp6_t *,
    ip_recv_attr_t *);
extern void     icmp_update_out_mib_v6(ill_t *, icmp6_t *);

extern boolean_t conn_wantpacket_v6(conn_t *, ip_recv_attr_t *, ip6_t *);

extern in6addr_scope_t  ip_addr_scope_v6(const in6_addr_t *);
extern void     ip_build_hdrs_v6(uchar_t *, uint_t, const ip_pkt_t *, uint8_t,
    uint32_t);
extern void     ip_fanout_udp_multi_v6(mblk_t *, ip6_t *, uint16_t, uint16_t,
    ip_recv_attr_t *);
extern void     ip_fanout_send_icmp_v6(mblk_t *, uint_t, uint8_t,
    ip_recv_attr_t *);
extern void     ip_fanout_proto_v6(mblk_t *, ip6_t *, ip_recv_attr_t *);
extern int      ip_find_hdr_v6(mblk_t *, ip6_t *, boolean_t, ip_pkt_t *,
    uint8_t *);
```

```
extern in6_addr_t ip_get_dst_v6(ip6_t *, const mblk_t *, boolean_t *);
extern ip6_rthdr_t      *ip_find_rthdr_v6(ip6_t *, uint8_t *);
extern boolean_t        ip_hdr_length_nexthdr_v6(mblk_t *, ip6_t *,
    uint16_t *, uint8_t **);
extern int      ip_hdr_length_v6(mblk_t *, ip6_t *);
extern uint32_t ip_massage_options_v6(ip6_t *, ip6_rthdr_t *, netstack_t *);
extern void     ip_forward_xmit_v6(nce_t *, mblk_t *, ip6_t *, ip_recv_attr_t *,
    uint32_t, uint32_t);
extern mblk_t   *ip_fraghdr_add_v6(mblk_t *, uint32_t, ip_xmit_attr_t *);
extern int      ip_fragment_v6(mblk_t *, nce_t *, iaflags_t, uint_t, uint32_t,
    uint32_t, zoneid_t, zoneid_t, pfirepostfrag_t postfragfn,
    uintptr_t *ixa_cookie);
extern int      ip_process_options_v6(mblk_t *, ip6_t *,
    uint8_t *, uint_t, uint8_t, ip_recv_attr_t *);
extern void     ip_process_rthdr(mblk_t *, ip6_t *, ip6_rthdr_t *,
    ip_recv_attr_t *);
extern int      ip_total_hdrs_len_v6(const ip_pkt_t *);
extern mblk_t   *ipsec_early_ah_v6(mblk_t *, ip_recv_attr_t *);
extern int      ipsec_ah_get_hdr_size_v6(mblk_t *, boolean_t);
extern void     ip_send_potential_redirect_v6(mblk_t *, ip6_t *, ire_t *,
    ip_recv_attr_t *);
extern void     ip_rput_v6(queue_t *, mblk_t *);
extern mblk_t   *mld_input(mblk_t *, ip_recv_attr_t *);
extern void     mld_joingroup(ilm_t *);
extern void     mld_leavegroup(ilm_t *);
extern void     mld_timeout_handler(void *);

extern void     pr_addr_dbg(char *, int, const void *);
extern void     *ip6_kstat_init(netstackid_t, ip6_stat_t *);
extern void     ip6_kstat_fini(netstackid_t, kstat_t *);
extern size_t   ip6_get_src_preferences(ip_xmit_attr_t *, uint32_t *);
extern int      ip6_set_src_preferences(ip_xmit_attr_t *, uint32_t);

#endif  /* _KERNEL */

#ifdef  __cplusplus
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   42357 Wed Sep 26 12:51:48 2012
new/usr/src/uts/common/io/scsi/adapters/iscsi/iscsi.h
inet_pton
**********************************************************
/*
 * 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 2000 by Cisco Systems, Inc.  All rights reserved.
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
#endif /* ! codereview */
 */

#ifndef _ISCSI_H
#define _ISCSI_H

/*
 * Block comment which describes the contents of this file.
 */

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/scsi/scsi.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/socket.h>
#include <sys/kstat.h>
#include <sys/sunddi.h>
#include <sys/sunmdi.h>
#include <sys/mdi_impldefs.h>
#include <sys/time.h>
#include <sys/nvpair.h>
#include <sys/sdt.h>

#include <sys/iscsi_protocol.h>
#include <sys/scsi/adapters/iscsi_if.h>
#include <iscsiAuthClient.h>
#include <iscsi_stats.h>
#include <iscsi_thread.h>
#include <sys/idm/idm.h>
#include <sys/idm/idm_conn_sm.h>
#include <nvfile.h>
#include <inet/ip.h>
#endif /* ! codereview */
```

```
#ifndef MIN
#define MIN(a, b) ((a) < (b) ? (a) : (b))
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define LOGIN_PDU_BUFFER_SIZE   (16 * 1024)     /* move somewhere else */

extern boolean_t iscsi_conn_logging;
extern boolean_t iscsi_io_logging;
extern boolean_t iscsi_login_logging;
extern boolean_t iscsi_logging;
extern boolean_t iscsi_sess_logging;
#define ISCSI_CONN_LOG  if (iscsi_conn_logging) cmn_err
#define ISCSI_IO_LOG    if (iscsi_io_logging) cmn_err
#define ISCSI_LOGIN_LOG if (iscsi_login_logging) cmn_err
#define ISCSI_LOG       if (iscsi_logging) cmn_err
#define ISCSI_SESS_LOG  if (iscsi_sess_logging) cmn_err

/*
 * Name Format of the different Task Queues
 */
#define ISCSI_SESS_IOTH_NAME_FORMAT             "io_thrd_%d.%d"
#define ISCSI_SESS_WD_NAME_FORMAT               "wd_thrd_%d.%d"
#define ISCSI_SESS_LOGIN_TASKQ_NAME_FORMAT      "login_taskq_%d.%d"
#define ISCSI_SESS_ENUM_TASKQ_NAME_FORMAT       "enum_taskq_%d.%d"
#define ISCSI_CONN_CN_TASKQ_NAME_FORMAT         "conn_cn_taskq_%d.%d.%d"
#define ISCSI_CONN_RXTH_NAME_FORMAT             "rx_thrd_%d.%d.%d"
#define ISCSI_CONN_TXTH_NAME_FORMAT             "tx_thrd_%d.%d.%d"

/*
 * The iSCSI driver will not build scatter/gather lists (iovec) longer
 * than the value defined here. Asserts have been include in the code
 * to check.
 */
#define ISCSI_MAX_IOVEC         5

#define ISCSI_DEFAULT_MAX_STORM_DELAY           32

/*
 * The SNDBUF and RCVBUF size parameters for the sockets are just a
 * guess for the time being (I think it is the values used by CISCO
 * or UNH).  Testing will have to be done to figure * out the impact
 * of these values on performance.
 */
#define ISCSI_SOCKET_SNDBUF_SIZE                (256 * 1024)
#define ISCSI_SOCKET_RCVBUF_SIZE                (256 * 1024)
#define ISCSI_TCP_NODELAY_DEFAULT               0
#define ISCSI_TCP_CNOTIFY_THRESHOLD_DEFAULT     2000
#define ISCSI_TCP_CABORT_THRESHOLD_DEFAULT      10000
#define ISCSI_TCP_ABORT_THRESHOLD_DEFAULT       (30 * 1000) /* milliseconds */
#define ISNS_TCP_ABORT_THRESHOLD_DEFAULT        (3 * 1000) /* milliseconds */

/* Default values for tunable parameters */
#define ISCSI_DEFAULT_RX_TIMEOUT_VALUE          60
#define ISCSI_DEFAULT_CONN_DEFAULT_LOGIN_MAX    180
#define ISCSI_DEFAULT_LOGIN_POLLING_DELAY       60

/*
 * Convenient short hand defines
 */
#define TARGET_PROP     "target"
#define LUN_PROP        "lun"
#define MDI_GUID        "wwn"
#define NDI_GUID        "client-guid"

#define ISCSI_SIG_CMD   0x11111111
#define ISCSI_SIG_LUN   0x22222222
#define ISCSI_SIG_CONN  0x33333333
#define ISCSI_SIG_SESS  0x44444444
#define ISCSI_SIG_HBA   0x55555555

#define SENDTARGETS_DISCOVERY   "SENDTARGETS_DISCOVERY"

#define ISCSI_LUN_MASK_MSB      0x00003f00
#define ISCSI_LUN_MASK_LSB      0x000000ff
#define ISCSI_LUN_MASK          (ISCSI_LUN_MASK_MSB | ISCSI_LUN_MASK_LSB)
#define ISCSI_LUN_BYTE_COPY(lun, report_lun_data) \
        lun[0] = (report_lun_data & ISCSI_LUN_MASK_MSB) >> 8; \
        lun[1] = (report_lun_data & ISCSI_LUN_MASK_LSB);
/*
 * Not defined by iSCSI, but used in the login code to
 * determine when to send the initial Login PDU
 */
#define ISCSI_INITIAL_LOGIN_STAGE       -1

typedef enum iscsi_status {
        /* Success */
        ISCSI_STATUS_SUCCESS = 0,
        /* Driver / Kernel / Code error */
        ISCSI_STATUS_INTERNAL_ERROR,
        /* ITT table is already full, unable to reserve slot */
        ISCSI_STATUS_ITT_TABLE_FULL,
        /* Login on connection failed */
        ISCSI_STATUS_LOGIN_FAILED,
        /* No connections are in the LOGGED_IN state */
        ISCSI_STATUS_NO_CONN_LOGGED_IN,
        /* TCP Transfer Error */
        ISCSI_STATUS_TCP_TX_ERROR,
        /* TCP Receive Error */
        ISCSI_STATUS_TCP_RX_ERROR,
        /* iSCSI packet RCV timeout */
        ISCSI_STATUS_RX_TIMEOUT,
        /* iSCSI Header Digest CRC error */
        ISCSI_STATUS_HEADER_DIGEST_ERROR,
        /* iSCSI Data Digest CRC error */
        ISCSI_STATUS_DATA_DIGEST_ERROR,
        /* kmem_alloc failure */
        ISCSI_STATUS_ALLOC_FAILURE,
        /* cmd (tran_abort/reset) failed */
        ISCSI_STATUS_CMD_FAILED,
        /* iSCSI protocol error */
        ISCSI_STATUS_PROTOCOL_ERROR,
        /* iSCSI protocol version mismatch */
        ISCSI_STATUS_VERSION_MISMATCH,
        /* iSCSI login negotiation failed */
        ISCSI_STATUS_NEGO_FAIL,
        /* iSCSI login authentication failed */
        ISCSI_STATUS_AUTHENTICATION_FAILED,
        /* iSCSI login redirection failed */
        ISCSI_STATUS_REDIRECTION_FAILED,
        /* iSCSI uscsi status failure */
        ISCSI_STATUS_USCSI_FAILED,
        /* data received would have overflowed given buffer */
        ISCSI_STATUS_DATA_OVERFLOW,
        /* session/connection needs to shutdown */
```

```
        ISCSI_STATUS_SHUTDOWN,
        /* logical unit in use */
        ISCSI_STATUS_BUSY,
        /* Login on connection failed, retries exceeded */
        ISCSI_STATUS_LOGIN_TIMED_OUT,
        /* iSCSI login tpgt negotiation failed */
        ISCSI_STATUS_LOGIN_TPGT_NEGO_FAIL
} iscsi_status_t;
#define ISCSI_SUCCESS(status) (status == ISCSI_STATUS_SUCCESS)

/* SNA32 check value used on increment of CmdSn values */
#define ISCSI_SNA32_CHECK 2147483648UL /* 2**31 */

/*
 * This is the maximum number of commands that can be outstanding
 * on a iSCSI session at anyone point in time.
 */
#define ISCSI_CMD_TABLE_SIZE            1024

/* Used on connections thread create of receiver thread */
extern pri_t minclsyspri;

/*
 * Callers of iscsid_config_one/all must hold this
 * semaphore across the calls.  Otherwise a ndi_devi_enter()
 * deadlock in the DDI layer may occur.
 */
extern ksema_t iscsid_config_semaphore;

extern kmutex_t iscsi_oid_mutex;
extern uint32_t iscsi_oid;
extern void *iscsi_state;

/*
 * NOP delay is used to send a iSCSI NOP (ie. ping) across the
 * wire to see if the target is still alive.  NOPs are only
 * sent when the RX thread hasn't received anything for the
 * below amount of time.
 */
#define ISCSI_DEFAULT_NOP_DELAY                 5 /* seconds */
extern int      iscsi_nop_delay;
/*
 * If we haven't received anything in a specified period of time
 * we will stop accepting IO via tran start.  This will enable
 * upper level drivers to see we might be having a problem and
 * in the case of scsi_vhci will start to route IO down a better
 * path.
 */
#define ISCSI_DEFAULT_RX_WINDOW                 20 /* seconds */
extern int      iscsi_rx_window;
/*
 * If we haven't received anything in a specified period of time
 * we will stop accepting IO via tran start.  This the max limit
 * when encountered we will start returning a fatal error.
 */
#define ISCSI_DEFAULT_RX_MAX_WINDOW             180 /* seconds */
extern int      iscsi_rx_max_window;

/*
 * During iscsi boot, if the boot session has been created, the
 * initiator hasn't changed the boot lun to be online, we will wait
 * 180s here for lun online by default.
 */
#define ISCSI_BOOT_DEFAULT_MAX_DELAY            180 /* seconds */
/*
 * +-----------------------------------------------------------------+
 * | iSCSI Driver Structures                                          |
 * +------------------------------------------------------------------+
 */

/*
 * iSCSI Auth Information
 */
typedef struct iscsi_auth {
        IscsiAuthStringBlock    auth_recv_string_block;
        IscsiAuthStringBlock    auth_send_string_block;
        IscsiAuthLargeBinary    auth_recv_binary_block;
        IscsiAuthLargeBinary    auth_send_binary_block;
        IscsiAuthClient         auth_client_block;
        int                     num_auth_buffers;
        IscsiAuthBufferDesc     auth_buffers[5];

        /*
         * To indicate if bi-directional authentication is enabled.
         * 0 means uni-directional authentication.
         * 1 means bi-directional authentication.
         */
        int                     bidirectional_auth;

        /* Initiator’s authentication information. */
        char                    username[iscsiAuthStringMaxLength];
        uint8_t                 password[iscsiAuthStringMaxLength];
        int                     password_length;

        /* Target’s authentication information. */
        char                    username_in[iscsiAuthStringMaxLength];
        uint8_t                 password_in[iscsiAuthStringMaxLength];
        int                     password_length_in;
} iscsi_auth_t;

/*
 * iSCSI Task
 */
typedef struct iscsi_task {
        void                    *t_arg;
        boolean_t               t_blocking;
        uint32_t                t_event_count;
} iscsi_task_t;

/*
 * These are all the iscsi_cmd types that we use to track our
 * commands between queues and actions.
 */
typedef enum iscsi_cmd_type {
        ISCSI_CMD_TYPE_SCSI = 1,        /* scsi cmd */
        ISCSI_CMD_TYPE_NOP,             /* nop / ping */
        ISCSI_CMD_TYPE_ABORT,           /* abort */
        ISCSI_CMD_TYPE_RESET,           /* reset */
        ISCSI_CMD_TYPE_LOGOUT,          /* logout */
        ISCSI_CMD_TYPE_LOGIN,           /* login */
        ISCSI_CMD_TYPE_TEXT             /* text */
} iscsi_cmd_type_t;

/*
 * iscsi_cmd_state - (reference iscsi_cmd.c for state diagram)
 */
typedef enum iscsi_cmd_state {
        ISCSI_CMD_STATE_FREE = 0,
        ISCSI_CMD_STATE_PENDING,
        ISCSI_CMD_STATE_ACTIVE,
        ISCSI_CMD_STATE_ABORTING,
        ISCSI_CMD_STATE_IDM_ABORTING,
```

```
        ISCSI_CMD_STATE_COMPLETED,
        ISCSI_CMD_STATE_MAX
} iscsi_cmd_state_t;

#ifdef ISCSI_CMD_SM_STRINGS
static const char *iscsi_cmd_state_names[ISCSI_CMD_STATE_MAX+1] = {
        "ISCSI_CMD_STATE_FREE",
        "ISCSI_CMD_STATE_PENDING",
        "ISCSI_CMD_STATE_ACTIVE",
        "ISCSI_CMD_STATE_ABORTING",
        "ISCSI_CMD_STATE_IDM_ABORTING",
        "ISCSI_CMD_STATE_COMPLETED",
        "ISCSI_CMD_STATE_MAX"
};
#endif

/*
 * iscsi command events
 */
typedef enum iscsi_cmd_event {
        ISCSI_CMD_EVENT_E1 = 0,
        ISCSI_CMD_EVENT_E2,
        ISCSI_CMD_EVENT_E3,
        ISCSI_CMD_EVENT_E4,
        ISCSI_CMD_EVENT_E6,
        ISCSI_CMD_EVENT_E7,
        ISCSI_CMD_EVENT_E8,
        ISCSI_CMD_EVENT_E9,
        ISCSI_CMD_EVENT_E10,
        ISCSI_CMD_EVENT_MAX
} iscsi_cmd_event_t;

#ifdef ISCSI_CMD_SM_STRINGS
static const char *iscsi_cmd_event_names[ISCSI_CMD_EVENT_MAX+1] = {
        "ISCSI_CMD_EVENT_E1",
        "ISCSI_CMD_EVENT_E2",
        "ISCSI_CMD_EVENT_E3",
        "ISCSI_CMD_EVENT_E4",
        "ISCSI_CMD_EVENT_E6",
        "ISCSI_CMD_EVENT_E7",
        "ISCSI_CMD_EVENT_E8",
        "ISCSI_CMD_EVENT_E9",
        "ISCSI_CMD_EVENT_E10",
        "ISCSI_CMD_EVENT_MAX"
};
#endif

/*
 * iscsi text command stages - these stages are used by iSCSI text
 * processing to manage long resonses.
 */
typedef enum iscsi_cmd_text_stage {
        ISCSI_CMD_TEXT_INITIAL_REQ = 0,
        ISCSI_CMD_TEXT_CONTINUATION,
        ISCSI_CMD_TEXT_FINAL_RSP
} iscsi_cmd_text_stage_t;

/*
 * iscsi cmd misc flags - bitwise applicable
 */
#define ISCSI_CMD_MISCFLAG_INTERNAL     0x1
#define ISCSI_CMD_MISCFLAG_FREE         0x2
#define ISCSI_CMD_MISCFLAG_STUCK        0x4
#define ISCSI_CMD_MISCFLAG_XARQ         0x8
#define ISCSI_CMD_MISCFLAG_SENT         0x10
#define ISCSI_CMD_MISCFLAG_FLUSH        0x20
```

```
/*
 * 1/2 of a 32 bit number, used for checking CmdSN
 * wrapped.
 */
#define ISCSI_CMD_SN_WRAP               0x80000000

#define ISCSI_CMD_PKT_STAT_INIT         0

/*
 * iSCSI cmd/pkt Structure
 */
typedef struct iscsi_cmd {
        uint32_t                cmd_sig;
        struct iscsi_cmd        *cmd_prev;
        struct iscsi_cmd        *cmd_next;
        struct iscsi_conn       *cmd_conn;

        iscsi_cmd_type_t        cmd_type;
        iscsi_cmd_state_t       cmd_state;
        iscsi_cmd_state_t       cmd_prev_state;
        clock_t                 cmd_lbolt_pending;
        clock_t                 cmd_lbolt_active;
        clock_t                 cmd_lbolt_aborting;
        clock_t                 cmd_lbolt_idm_aborting;
        clock_t                 cmd_lbolt_timeout;
        uint8_t                 cmd_misc_flags;
        idm_task_t              *cmd_itp;

        union {
                /* ISCSI_CMD_TYPE_SCSI */
                struct {
                        idm_buf_t               *ibp_ibuf;
                        idm_buf_t               *ibp_obuf;
                        struct scsi_pkt         *pkt;
                        struct buf              *bp;
                        int                     cmdlen;
                        int                     statuslen;
                        size_t                  data_transferred;

                        uint32_t                lun;

                        /*
                         * If SCSI_CMD_TYPE is in ABORTING_STATE
                         * then the abort_icmdp field will be a pointer
                         * to the abort command chasing this one.
                         */
                        struct iscsi_cmd        *abort_icmdp;
                        /*
                         * pointer to the r2t associated with this
                         * command (if any)
                         */
                        struct iscsi_cmd        *r2t_icmdp;
                        /*
                         * It will be true if this command has
                         * another R2T to handle.
                         */
                        boolean_t               r2t_more;
                        /*
                         * It is used to record pkt_statistics temporarily.
                         */
                        uint_t                  pkt_stat;
                } scsi;
                /* ISCSI_CMD_TYPE_ABORT */
                struct {
                        /* pointer to original iscsi_cmd, for abort */
```

```
                        struct iscsi_cmd        *icmdp;
                } abort;
                /* ISCSI_CMD_TYPE_RESET */
                struct {
                        int                     level;
                        uint8_t                 response;
                } reset;
                /* ISCSI_CMD_TYPE_NOP */
                struct {
                        int rsvd;
                } nop;
                /* ISCSI_CMD_TYPE_R2T */
                struct {
                        struct iscsi_cmd        *icmdp;
                        uint32_t                offset;
                        uint32_t                length;
                } r2t;
                /* ISCSI_CMD_TYPE_LOGIN */
                struct {
                        int rvsd;
                } login;
                /* ISCSI_CMD_TYPE_LOGOUT */
                struct {
                        int rvsd;
                } logout;
                /* ISCSI_CMD_TYPE_TEXT */
                struct {
                        char                    *buf;
                        int                     buf_len;
                        uint32_t                offset;
                        uint32_t                data_len;
                        uint32_t                total_rx_len;
                        uint32_t                ttt;
                        uint8_t                 lun[8];
                        iscsi_cmd_text_stage_t  stage;
                } text;
        } cmd_un;

        struct iscsi_lun        *cmd_lun; /* associated lun */

        uint32_t                cmd_itt;
        uint32_t                cmd_ttt;

        /*
         * If a data digest error is seem on a data pdu.  This flag
         * will get set.  We don't abort the cmd immediately because
         * we want to read in all the data to get it out of the
         * stream.  Once the completion for the cmd is received we
         * we will abort the cmd and state no sense data was available.
         */
        boolean_t               cmd_crc_error_seen;

        /*
         * Used to block and wake up caller until action is completed.
         * This is for ABORT, RESET, and PASSTHRU cmds.
         */
        int                     cmd_result;
        int                     cmd_completed;
        kmutex_t                cmd_mutex;
        kcondvar_t              cmd_completion;

        idm_pdu_t               cmd_pdu;

        sm_audit_buf_t          cmd_state_audit;

        uint32_t                cmd_sn;
```

```
} iscsi_cmd_t;


/*
 * iSCSI LUN Structure
 */
typedef struct iscsi_lun {
        uint32_t                lun_sig;
        int                     lun_state;

        struct iscsi_lun        *lun_next;      /* next lun on this sess. */
        struct iscsi_sess       *lun_sess;      /* parent sess. for lun */
        dev_info_t              *lun_dip;
        mdi_pathinfo_t          *lun_pip;

        uint16_t                lun_num;        /* LUN */
        uint8_t                 lun_addr_type;  /* LUN addressing type */
        uint32_t                lun_oid;        /* OID */
        char                    *lun_guid;      /* GUID */
        int                     lun_guid_size;  /* GUID allocation size */
        char                    *lun_addr;      /* sess,lun */
        time_t                  lun_time_online;

        uchar_t                 lun_cap;        /* bitmap of scsi caps */

        uchar_t                 lun_vid[ISCSI_INQ_VID_BUF_LEN]; /* Vendor ID */
        uchar_t                 lun_pid[ISCSI_INQ_PID_BUF_LEN]; /* Product ID */

        uchar_t                 lun_type;
} iscsi_lun_t;

#define ISCSI_LUN_STATE_CLEAR       0           /* used to clear all states */
#define ISCSI_LUN_STATE_OFFLINE     1
#define ISCSI_LUN_STATE_ONLINE      2
#define ISCSI_LUN_STATE_INVALID     4           /* offline failed */
#define ISCSI_LUN_STATE_BUSY        8           /* logic unit is in reset */

#define ISCSI_LUN_CAP_RESET         0x01

#define ISCSI_SCSI_RESET_SENSE_CODE 0x29
#define ISCSI_SCSI_LUNCHANGED_CODE      0x3f

#define ISCSI_SCSI_LUNCHANGED_ASCQ      0x0e

/*
 *
 *
 */
typedef struct iscsi_queue {
        iscsi_cmd_t     *head;
        iscsi_cmd_t     *tail;
        int             count;
        kmutex_t        mutex;
} iscsi_queue_t;

#define ISCSI_CONN_DEFAULT_LOGIN_MIN            0
#define ISCSI_CONN_DEFAULT_LOGIN_REDIRECT       10

/* iSCSI tunable Parameters */
typedef struct iscsi_tunable_params {
        int             recv_login_rsp_timeout; /* range: 0 - 60*60 */
        int             conn_login_max;         /* range: 0 - 60*60 */
        int             polling_login_delay;    /* range: 0 - 60*60 */
} iscsi_tunable_params_t;

typedef union iscsi_sockaddr {
```

```
        struct sockaddr         sin;
        struct sockaddr_in      sin4;
        struct sockaddr_in6     sin6;
} iscsi_sockaddr_t;

#define SIZEOF_SOCKADDR(so)     ((so)->sa_family == AF_INET ? \
        sizeof (struct sockaddr_in) : sizeof (struct sockaddr_in6))

typedef enum {
        LOGIN_START,
        LOGIN_READY,
        LOGIN_TX,
        LOGIN_RX,
        LOGIN_ERROR,
        LOGIN_DONE,
        LOGIN_FFP,
        LOGIN_MAX
} iscsi_login_state_t;

#ifdef ISCSI_LOGIN_STATE_NAMES
static const char *iscsi_login_state_names[LOGIN_MAX+1] = {
        "LOGIN_START",
        "LOGIN_READY",
        "LOGIN_TX",
        "LOGIN_RX",
        "LOGIN_ERROR",
        "LOGIN_DONE",
        "LOGIN_FFP",
        "LOGIN_MAX"
};
#endif

/*
 * iscsi_conn_state
 */
typedef enum iscsi_conn_state {
        ISCSI_CONN_STATE_UNDEFINED = 0,
        ISCSI_CONN_STATE_FREE,
        ISCSI_CONN_STATE_IN_LOGIN,
        ISCSI_CONN_STATE_LOGGED_IN,
        ISCSI_CONN_STATE_IN_LOGOUT,
        ISCSI_CONN_STATE_FAILED,
        ISCSI_CONN_STATE_POLLING,
        ISCSI_CONN_STATE_MAX
} iscsi_conn_state_t;

#ifdef ISCSI_ICS_NAMES
static const char *iscsi_ics_name[ISCSI_CONN_STATE_MAX+1] = {
        "ISCSI_CONN_STATE_UNDEFINED",
        "ISCSI_CONN_STATE_FREE",
        "ISCSI_CONN_STATE_IN_LOGIN",
        "ISCSI_CONN_STATE_LOGGED_IN",
        "ISCSI_CONN_STATE_IN_LOGOUT",
        "ISCSI_CONN_STATE_FAILED",
        "ISCSI_CONN_STATE_POLLING",
        "ISCSI_CONN_STATE_MAX"
};
#endif

#define ISCSI_CONN_STATE_FULL_FEATURE(state) \
        ((state == ISCSI_CONN_STATE_LOGGED_IN) || \
        (state == ISCSI_CONN_STATE_IN_LOGOUT))

/*
 * iSCSI Connection Structure
 */
```

```
typedef struct iscsi_conn {
        uint32_t                conn_sig;
        struct iscsi_conn       *conn_next;     /* next conn on this sess. */
        struct iscsi_sess       *conn_sess;     /* parent sess. for conn. */

        iscsi_conn_state_t      conn_state;     /* cur. conn. driver state */
        iscsi_conn_state_t      conn_prev_state; /* prev. conn. driver state */
        /* protects the session state and synchronizes the state machine */
        kmutex_t                conn_state_mutex;
        kcondvar_t              conn_state_change;
        boolean_t               conn_state_destroy;
        boolean_t               conn_state_ffp;
        boolean_t               conn_state_idm_connected;
        boolean_t               conn_async_logout;
        ddi_taskq_t             *conn_cn_taskq;

        idm_conn_t              *conn_ic;

        /* base connection information, may have been redirected */
        iscsi_sockaddr_t        conn_base_addr;

        /* current connection information, may have been redirected */
        iscsi_sockaddr_t        conn_curr_addr;

        boolean_t               conn_bound;
        iscsi_sockaddr_t        conn_bound_addr;

        uint32_t                conn_cid;       /* CID */
        uint32_t                conn_oid;       /* OID */

        int                     conn_current_stage;     /* iSCSI login stage */
        int                     conn_next_stage;        /* iSCSI login stage */
        int                     conn_partial_response;

        /*
         * The active queue contains iscsi_cmds that have already
         * been sent on this connection.  Any future responses to
         * these cmds require alligence to this connection.  If there
         * are issues with these cmds the command may need aborted
         * depending on the command type, and must be put back into
         * the session's pending queue or aborted.
         */
        iscsi_queue_t           conn_queue_active;
        iscsi_queue_t           conn_queue_idm_aborting;

        /* lbolt from the last receive, used for nop processing */
        clock_t                 conn_rx_lbolt;
        clock_t                 conn_nop_lbolt;

        iscsi_thread_t          *conn_tx_thread;

        /*
         * The expstatsn is the command status sn that is expected
         * next from the target.  Command status is carried on a number
         * of iSCSI PDUs (ex.  SCSI Cmd Response, SCSI Data IN with
         * S-Bit set, ...), not all PDUs.  If our expstatsn is different
         * than the received statsn.  Something got out of sync we need to
         * recover.
         */
        uint32_t                conn_expstatsn;
        uint32_t                conn_laststatsn;

        /* active login parameters */
        iscsi_login_params_t    conn_params;

        /* Statistics */
```

```
        struct {
                kstat_t                 *ks;
                iscsi_conn_stats_t      ks_data;
        } stats;

        /*
         * These fields are used to coordinate the asynchronous IDM
         * PDU operations with the synchronous login code.
         */
        kmutex_t                conn_login_mutex;
        kcondvar_t              conn_login_cv;
        iscsi_login_state_t     conn_login_state;
        iscsi_status_t          conn_login_status;
        iscsi_hdr_t             conn_login_resp_hdr;
        char                    *conn_login_data;
        int                     conn_login_datalen;
        int                     conn_login_max_data_length;

        /*
         * login min and max identify the amount of time
         * in lbolt that iscsi_start_login() should attempt
         * to log into a target portal.  The login will
         * delay until the min lbolt has been reached and
         * will end once max time has been reached.  These
         * values are normally set to the default but can
         * are also altered by async commands received from
         * the targetlogin.
         */
        clock_t                 conn_login_min;
        clock_t                 conn_login_max;
        sm_audit_buf_t          conn_state_audit;

        /* active tunable parameters */
        iscsi_tunable_params_t  conn_tunable_params;
        boolean_t               conn_timeout;
} iscsi_conn_t;


/*
 * iscsi_sess_state - (reference iscsi_sess.c for state diagram)
 */
typedef enum iscsi_sess_state {
        ISCSI_SESS_STATE_FREE = 0,
        ISCSI_SESS_STATE_LOGGED_IN,
        ISCSI_SESS_STATE_FAILED,
        ISCSI_SESS_STATE_IN_FLUSH,
        ISCSI_SESS_STATE_FLUSHED,
        ISCSI_SESS_STATE_MAX
} iscsi_sess_state_t;

#ifdef ISCSI_SESS_SM_STRINGS
static const char *iscsi_sess_state_names[ISCSI_SESS_STATE_MAX+1] = {
        "ISCSI_SESS_STATE_FREE",
        "ISCSI_SESS_STATE_LOGGED_IN",
        "ISCSI_SESS_STATE_FAILED",
        "ISCSI_SESS_STATE_IN_FLUSH",
        "ISCSI_SESS_STATE_FLUSHED",
        "ISCSI_SESS_STATE_MAX"
};
#endif

#define ISCSI_SESS_STATE_FULL_FEATURE(state) \
        ((state == ISCSI_SESS_STATE_LOGGED_IN) || \
        (state == ISCSI_SESS_STATE_IN_FLUSH))
```

```
typedef enum iscsi_sess_event {
        ISCSI_SESS_EVENT_N1 = 0,
        ISCSI_SESS_EVENT_N3,
        ISCSI_SESS_EVENT_N5,
        ISCSI_SESS_EVENT_N6,
        ISCSI_SESS_EVENT_N7,
        ISCSI_SESS_EVENT_MAX
} iscsi_sess_event_t;

#ifdef ISCSI_SESS_SM_STRINGS
static const char *iscsi_sess_event_names[ISCSI_SESS_EVENT_MAX+1] = {
        "ISCSI_SESS_EVENT_N1",
        "ISCSI_SESS_EVENT_N3",
        "ISCSI_SESS_EVENT_N5",
        "ISCSI_SESS_EVENT_N6",
        "ISCSI_SESS_EVENT_N7",
        "ISCSI_SESS_EVENT_MAX"
};
#endif

typedef enum iscsi_sess_type {
        ISCSI_SESS_TYPE_NORMAL = 0,
        ISCSI_SESS_TYPE_DISCOVERY
} iscsi_sess_type_t;

#define SESS_ABORT_TASK_MAX_THREADS     1

/* Sun’s initiator session ID */
#define ISCSI_SUN_ISID_0    0x40    /* ISID - EN format */
#define ISCSI_SUN_ISID_1    0x00    /* Sec B */
#define ISCSI_SUN_ISID_2    0x00    /* Sec B */
#define ISCSI_SUN_ISID_3    0x2A    /* Sec C - 42 = Sun’s EN */
/*
 * defines 4-5 are the reserved values.  These reserved values
 * are used as the ISID for an initiator-port in MP-API and used
 * for the send targets discovery sessions.  Byte 5 is overridden
 * for full feature sessions.  The default values of byte 5 for a
 * full feature session is 0.  When MS/T is enabled with more than
 * one session this byte 5 will increment > 0 up to
 * ISCSI_MAX_CONFIG_SESSIONS.
 */
#define ISCSI_SUN_ISID_4    0x00
#define ISCSI_SUN_ISID_5    0xFF

#define ISCSI_DEFAULT_SESS_BOUND        B_FALSE
#define ISCSI_DEFAULT_SESS_NUM          1

typedef enum iscsi_enum_status {
        ISCSI_SESS_ENUM_FREE            =       0,
        ISCSI_SESS_ENUM_INPROG,
        ISCSI_SESS_ENUM_DONE
} iscsi_enum_status_t;

typedef enum iscsi_enum_result {
        ISCSI_SESS_ENUM_COMPLETE        =       0,
        ISCSI_SESS_ENUM_PARTIAL,
        ISCSI_SESS_ENUM_IOFAIL,
        ISCSI_SESS_ENUM_SUBMITTED,
        ISCSI_SESS_ENUM_SUBFAIL,
        ISCSI_SESS_ENUM_GONE,
        ISCSI_SESS_ENUM_TUR_FAIL
} iscsi_enum_result_t;

/*
 * iSCSI Session(Target) Structure
 */
```

```
typedef struct iscsi_sess {
        uint32_t                sess_sig;

        iscsi_sess_state_t      sess_state;
        iscsi_sess_state_t      sess_prev_state;
        clock_t                 sess_state_lbolt;
        /* protects the session state and synchronizes the state machine */
        krwlock_t               sess_state_rwlock;

        /*
         * Associated target OID.
         */
        uint32_t                sess_target_oid;

        /*
         * Session OID.  Used by IMA, interfaces and exported as
         * TARGET_PROP which is checked by the NDI.  In addition
         * this is used in our tran_lun_init function.
         */
        uint32_t                sess_oid;

        struct iscsi_sess       *sess_next;
        struct iscsi_hba        *sess_hba;

        /* list of all luns relating to session */
        struct iscsi_lun        *sess_lun_list;
        krwlock_t               sess_lun_list_rwlock;

        /* list of all connections relating to session */
        struct iscsi_conn       *sess_conn_list;
        struct iscsi_conn       *sess_conn_list_last_ptr;
        /* pointer to active connection in session */
        struct iscsi_conn       *sess_conn_act;
        krwlock_t               sess_conn_list_rwlock;

        /* Connection ID for next connection to be added to session */
        uint32_t                sess_conn_next_cid;

        /*
         * last time any connection on this session received
         * data from the target.
         */
        clock_t                 sess_rx_lbolt;

        clock_t                 sess_failure_lbolt;

        int                     sess_storm_delay;

        /*
         * sess_cmdsn_mutex protects the cmdsn and itt table/values
         * Cmdsn isn’t that big of a problem yet since we only have
         * one connection but in the future we will need to ensure
         * this locking is working so keep the sequence numbers in
         * sync on the wire.
         *
         * We also use this lock to protect the ITT table and it’s
         * values.  We need to make sure someone doesn’t assign
         * a duplicate ITT value or cell to a command.  Also we
         * need to make sure when someone is looking up an ITT
         * that the command is still in that correct queue location.
         */
        kmutex_t                sess_cmdsn_mutex;

        /*
         * iSCSI command sequencing / windowing.  The next
         * command to be sent via the pending queue will
```

```
	 * get the sess_cmdsn.  If the maxcmdsn is less
	 * than the next cmdsn then the iSCSI window is
	 * closed and this command cannot be sent yet.
	 * Most iscsi cmd responses from the target carry
	 * a new maxcmdsn.  If this new maxcmdsn is greater
	 * than the sess_maxcmdsn we will update it's value
	 * and set a timer to fire in one tick and reprocess
	 * the pending queue.
	 *
	 * The expcmdsn.   Is the value the target expects
	 * to be sent for my next cmdsn.  If the expcmdsn
	 * and the cmdsn get out of sync this could denote
	 * a communication problem.
	 */
	uint32_t		sess_cmdsn;
	uint32_t		sess_expcmdsn;
	uint32_t		sess_maxcmdsn;

	/* Next Initiator Task Tag (ITT) to use */
	uint32_t		sess_itt;
	/*
	 * The session iscsi_cmd table is used to a fast performance
	 * lookup of an ITT to a iscsi_cmd when we receive an iSCSI
	 * PDU from the wire.  To reserve a location in the sess_cmd_table
	 * we try the sess_itt % ISCSI_CMD_TABLE_SIZE if this cmd table
	 * cell is already full.  Then increament the sess_itt and
	 * try to get the cell position again, repeat until an empty
	 * cell is found.  Once an empty cell is found place your
	 * scsi_cmd point into the cell to reserve the location.  This
	 * selection process should be done while holding the session's
	 * mutex.
	 */
	struct iscsi_cmd	*sess_cmd_table[ISCSI_CMD_TABLE_SIZE];
	int			sess_cmd_table_count;

	/*
	 * The pending queue contains all iscsi_cmds that require an
	 * open MaxCmdSn window to be put on the wire and haven't
	 * been placed on the wire.  Once placed on the wire they
	 * will be moved to a connections specific active queue.
	 */
	iscsi_queue_t		sess_queue_pending;

	iscsi_error_t		sess_last_err;

	iscsi_queue_t		sess_queue_completion;
	/* configured login parameters */
	iscsi_login_params_t	sess_params;

	/* general iSCSI protocol/session info */
	uchar_t			sess_name[ISCSI_MAX_NAME_LEN];
	int			sess_name_length;
	char			sess_alias[ISCSI_MAX_NAME_LEN];
	int			sess_alias_length;
	iSCSIDiscoveryMethod_t	sess_discovered_by;
	iscsi_sockaddr_t	sess_discovered_addr;
	uchar_t			sess_isid[ISCSI_ISID_LEN]; /* Session ID */
	uint16_t		sess_tsid; /* Target ID */
	/*
	 * If the target portal group tag(TPGT) is equal to ISCSI_DEFAULT_TPGT
	 * then the initiator will accept a successful login with any TPGT
	 * specified by the target.  If a none default TPGT is configured
	 * then we will only successfully accept a login with that matching
	 * TPGT value.
	 */
	int			sess_tpgt_conf;
```

```
	/* This field records the negotiated TPGT value, preserved for dtrace */
	int			sess_tpgt_nego;

	/*
	 * Authentication information.
	 *
	 * DCW: Again IMA seems to take a session view at this
	 * information.
	 */
	iscsi_auth_t		sess_auth;

	/* Statistics */
	struct {
		kstat_t			*ks;
		iscsi_sess_stats_t	ks_data;
		kstat_t			*ks_io;
		kstat_io_t		ks_io_data;
		kmutex_t		ks_io_lock;
	} stats;

	iscsi_thread_t		*sess_ic_thread;
	boolean_t		sess_window_open;
	boolean_t		sess_boot;
	iscsi_sess_type_t	sess_type;

	ddi_taskq_t		*sess_login_taskq;

	iscsi_thread_t		*sess_wd_thread;

	sm_audit_buf_t		sess_state_audit;

	kmutex_t		sess_reset_mutex;

	boolean_t		sess_reset_in_progress;

	boolean_t		sess_boot_nic_reset;
	kmutex_t		sess_enum_lock;
	kcondvar_t		sess_enum_cv;
	iscsi_enum_status_t	sess_enum_status;
	iscsi_enum_result_t	sess_enum_result;
	uint32_t		sess_enum_result_count;
	ddi_taskq_t		*sess_enum_taskq;

	kmutex_t		sess_state_wmutex;
	kcondvar_t		sess_state_wcv;
	boolean_t		sess_state_hasw;

	/* to accelerate the state change in case of new event */
	volatile uint32_t	sess_state_event_count;
} iscsi_sess_t;

/*
 * This structure will be used to store sessions to be online
 * during normal login operation.
 */
typedef struct iscsi_sess_list {
	iscsi_sess_t		*session;
	struct iscsi_sess_list	*next;
} iscsi_sess_list_t;

/*
 * iSCSI client notify task context for deferred IDM notifications processing
 */
typedef struct iscsi_cn_task {
	idm_conn_t		*ct_ic;
	idm_client_notify_t	ct_icn;
```

```
        uintptr_t               ct_data;
} iscsi_cn_task_t;

/*
 * iscsi_network
 */
typedef struct iscsi_network {
        void* (*socket)(int domain, int, int);
        int (*bind)(void *, struct sockaddr *, int, int, int);
        int (*connect)(void *, struct sockaddr *, int, int, int);
        int (*listen)(void *, int);
        void* (*accept)(void *, struct sockaddr *, int *);
        int (*getsockname)(void *, struct sockaddr *, socklen_t *);
        int (*getsockopt)(void *, int, int, void *, int *, int);
        int (*setsockopt)(void *, int, int, void *, int);
        int (*shutdown)(void *, int);
        void (*close)(void *);

        size_t (*poll)(void *, clock_t);
        size_t (*sendmsg)(void *, struct msghdr *);
        size_t (*recvmsg)(void *, struct msghdr *, int);

        iscsi_status_t (*sendpdu)(void *, iscsi_hdr_t *, char *, int);
        iscsi_status_t (*recvdata)(void *, iscsi_hdr_t *, char *,
            int, int, int);
        iscsi_status_t (*recvhdr)(void *, iscsi_hdr_t *, int, int, int);

        struct {
                int                     sndbuf;
                int                     rcvbuf;
                int                     nodelay;
                int                     conn_notify_threshold;
                int                     conn_abort_threshold;
                int                     abort_threshold;
        } tweaks;
} iscsi_network_t;

#define ISCSI_NET_HEADER_DIGEST 0x00000001
#define ISCSI_NET_DATA_DIGEST   0x00000002

extern iscsi_network_t *iscsi_net;

/*
 * If we get bus_config requests in less than 5 seconds
 * apart skip the name services re-discovery and just
 * complete the requested logins.  This protects against
 * bus_config storms from stale /dev links.
 */
#define ISCSI_CONFIG_STORM_DELAY_DEFAULT    5

/*
 * iSCSI HBA Structure
 */
typedef struct iscsi_hba {
        uint32_t                hba_sig;
        dev_info_t              *hba_dip;       /* dev info ptr */
        scsi_hba_tran_t         *hba_tran;      /* scsi tran ptr */
        ldi_ident_t             hba_li;

        struct iscsi_sess       *hba_sess_list; /* sess. list for hba */
        krwlock_t               hba_sess_list_rwlock; /* protect sess. list */

        /* lbolt of the last time we received a config request */
        clock_t                 hba_config_lbolt;
        /* current number of seconds to protect against bus config storms */
        int                     hba_config_storm_delay;
```

```

        /* general iSCSI protocol hba/initiator info */
        uchar_t                 hba_name[ISCSI_MAX_NAME_LEN];
        int                     hba_name_length;
        uchar_t                 hba_alias[ISCSI_MAX_NAME_LEN];
        int                     hba_alias_length;

        /* Default SessionID for HBA */
        uchar_t                 hba_isid[ISCSI_ISID_LEN];

        /* Default HBA wide settings */
        iscsi_login_params_t    hba_params;

        /*
         * There's only one HBA and it's set to ISCSI_INITIATOR_OID
         * (value of 1) at the beginning of time.
         */
        uint32_t                hba_oid;

        /*
         * Keep track of which events have been sent. User daemons request
         * this information so they don't wait for events which they won't
         * see.
         */
        kmutex_t                hba_discovery_events_mutex;
        iSCSIDiscoveryMethod_t  hba_discovery_events;
        boolean_t               hba_discovery_in_progress;

        boolean_t               hba_mpxio_enabled; /* mpxio-enabled */
        /* if the persistent store is loaded */
        boolean_t               hba_persistent_loaded;

        /*
         * Ensures only one SendTargets operation occurs at a time
         */
        ksema_t                 hba_sendtgts_semaphore;

        /*
         * Statistics
         */
        struct {
                kstat_t                 *ks;
                iscsi_hba_stats_t       ks_data;
        } stats;

        /*
         * track/control the service status and client
         *
         * service- service online ensures the operational of cli
         *        - and the availability of iSCSI discovery/devices
         *        - so obviously offline means the unusable of cli
         *        - , disabling of all discovery methods and to offline
         *        - all discovered devices
         *
         * client - here the client actually means 'exclusive client'
         *        - for operations these clients take may conflict
         *        - with the changing of service status and therefore
         *        - need to be exclusive
         *
         * The service has three status:
         *      ISCSI_SERVICE_ENABLED    -      client is permitted to
         *                               -      request service
         *
         *      ISCSI_SERVICE_DISABLED   -      client is not permitted to
         *                               -      request service
         *
         *      ISCSI_SERVICE_TRANSITION -      client must wait for
         *                               -      one of above two statuses
         *
         * The hba_service_client_count tracks the number of
         * current clients, it increases with new clients and decreases
         * with leaving clients. It stops to increase once the
         * ISCSI_SERVICE_TRANSITION is set, and causes later clients be
         * blocked there.
         *
         * The status of the service can only be changed when the number
         * of current clients reaches zero.
         *
         * Clients include:
         *      iscsi_ioctl
         *      iscsi_tran_bus_config
         *      iscsi_tran_bus_unconfig
         *      isns_scn_callback
         */
        kmutex_t                hba_service_lock;
        kcondvar_t              hba_service_cv;
        uint32_t                hba_service_status;
        uint32_t                hba_service_client_count;

        /* Default HBA tunable settings */
        iscsi_tunable_params_t  hba_tunable_params;
        boolean_t               hba_service_status_overwrite;
} iscsi_hba_t;

/*
 * +--------------------------------------------------------------------+
 * | iSCSI prototypes                                                   |
 * +--------------------------------------------------------------------+
 */

/* IDM client callback entry points */
idm_rx_pdu_cb_t iscsi_rx_scsi_rsp;
idm_rx_pdu_cb_t iscsi_rx_misc_pdu;
idm_rx_pdu_error_cb_t iscsi_rx_error_pdu;
idm_build_hdr_cb_t iscsi_build_hdr;
idm_task_cb_t iscsi_task_aborted;
idm_client_notify_cb_t iscsi_client_notify;

/* iscsi_io.c */
int iscsi_sna_lte(uint32_t n1, uint32_t n2);
char *iscsi_get_next_text(char *data, int data_length, char *curr_text);

void iscsi_ic_thread(iscsi_thread_t *thread, void *arg);
void iscsi_tx_thread(iscsi_thread_t *thread, void *arg);
void iscsi_wd_thread(iscsi_thread_t *thread, void *arg);

iscsi_status_t iscsi_tx_cmd(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);

void iscsi_task_cleanup(int opcode, iscsi_cmd_t *icmdp);

void iscsi_handle_abort(void *arg);
iscsi_status_t iscsi_handle_reset(iscsi_sess_t *isp, int level,
    iscsi_lun_t *ilp);
iscsi_status_t iscsi_handle_logout(iscsi_conn_t *icp);
iscsi_status_t iscsi_handle_passthru(iscsi_sess_t *isp, uint16_t lun,
    struct uscsi_cmd *ucmdp);
iscsi_status_t iscsi_handle_text(iscsi_conn_t *icp,
    char *buf, uint32_t buf_len, uint32_t data_len, uint32_t *rx_data_len);

void iscsi_iodone(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);

/* iscsi_crc.c */
```

```
uint32_t iscsi_crc32c(void *address, unsigned long length);
uint32_t iscsi_crc32c_continued(void *address, unsigned long length,
    uint32_t crc);

/* iscsi_queue.c */
void iscsi_init_queue(iscsi_queue_t *queue);
void iscsi_destroy_queue(iscsi_queue_t *queue);
void iscsi_enqueue_pending_cmd(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);
void iscsi_dequeue_pending_cmd(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);
void iscsi_enqueue_active_cmd(iscsi_conn_t *icp, iscsi_cmd_t *icmdp);
void iscsi_dequeue_active_cmd(iscsi_conn_t *icp, iscsi_cmd_t *icmdp);
void iscsi_enqueue_idm_aborting_cmd(iscsi_conn_t *icp, iscsi_cmd_t *icmdp);
void iscsi_dequeue_idm_aborting_cmd(iscsi_conn_t *icp, iscsi_cmd_t *icmdp);
void iscsi_enqueue_completed_cmd(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);
iscsi_status_t iscsi_dequeue_cmd(iscsi_cmd_t **, iscsi_cmd_t **, iscsi_cmd_t *);
void iscsi_move_queue(iscsi_queue_t *src_queue, iscsi_queue_t *dst_queue);
void iscsi_enqueue_cmd_head(iscsi_cmd_t **, iscsi_cmd_t **,
    iscsi_cmd_t *);

/* iscsi_login.c */
iscsi_status_t iscsi_login_start(void *arg);
void iscsi_login_update_state(iscsi_conn_t *icp,
    iscsi_login_state_t next_state);
void iscsi_login_update_state_locked(iscsi_conn_t *icp,
    iscsi_login_state_t next_state);


/* iscsi_stats.c */
boolean_t iscsi_hba_kstat_init(struct iscsi_hba *ihp);
boolean_t iscsi_hba_kstat_term(struct iscsi_hba *ihp);
boolean_t iscsi_sess_kstat_init(struct iscsi_sess *isp);
boolean_t iscsi_sess_kstat_term(struct iscsi_sess *isp);
boolean_t iscsi_conn_kstat_init(struct iscsi_conn       *icp);
void iscsi_conn_kstat_term(struct iscsi_conn *icp);

/* iscsi_net.c */
void iscsi_net_init();
void iscsi_net_fini();
iscsi_status_t iscsi_net_interface(boolean_t reset);

/* iscsi_sess.c */
iscsi_sess_t *iscsi_sess_create(iscsi_hba_t *ihp,
    iSCSIDiscoveryMethod_t method, struct sockaddr *addr_dsc,
    char *target_name, int tpgt, uchar_t isid_lsb,
    iscsi_sess_type_t type, uint32_t *oid);
void iscsi_sess_online(void *arg);
int iscsi_sess_get(uint32_t oid, iscsi_hba_t *ihp, iscsi_sess_t **ispp);
iscsi_status_t iscsi_sess_destroy(iscsi_sess_t *isp);
void iscsi_sess_state_machine(iscsi_sess_t *isp, iscsi_sess_event_t event,
    uint32_t event_count);
char *iscsi_sess_state_str(iscsi_sess_state_t state);
boolean_t iscsi_sess_set_auth(iscsi_sess_t *isp);
iscsi_status_t iscsi_sess_reserve_scsi_itt(iscsi_cmd_t *icmdp);
void iscsi_sess_release_scsi_itt(iscsi_cmd_t *icmdp);
iscsi_status_t iscsi_sess_reserve_itt(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);
void iscsi_sess_release_itt(iscsi_sess_t *isp, iscsi_cmd_t *icmdp);
void iscsi_sess_redrive_io(iscsi_sess_t *isp);
int iscsi_sess_get_by_target(uint32_t target_oid, iscsi_hba_t *ihp,
    iscsi_sess_t **ispp);
iscsi_enum_result_t iscsi_sess_enum_request(iscsi_sess_t *isp,
    boolean_t wait, uint32_t event_count);
iscsi_enum_result_t iscsi_sess_enum_query(iscsi_sess_t *isp);
void iscsi_sess_enter_state_zone(iscsi_sess_t *isp);
void iscsi_sess_exit_state_zone(iscsi_sess_t *isp);

/* iscsi_conn.c */
```

```
iscsi_status_t iscsi_conn_create(struct sockaddr *addr, iscsi_sess_t *isp,
    iscsi_conn_t **icpp);
iscsi_status_t iscsi_conn_online(iscsi_conn_t *icp);
iscsi_status_t iscsi_conn_offline(iscsi_conn_t *icp);
iscsi_status_t iscsi_conn_destroy(iscsi_conn_t *icp);
void iscsi_conn_set_login_min_max(iscsi_conn_t *icp, int min, int max);
iscsi_status_t iscsi_conn_sync_params(iscsi_conn_t *icp);
void iscsi_conn_retry(iscsi_sess_t *isp, iscsi_conn_t *icp);
void iscsi_conn_update_state(iscsi_conn_t *icp, iscsi_conn_state_t next_state);
void iscsi_conn_update_state_locked(iscsi_conn_t *icp,
                        iscsi_conn_state_t next_state);

/* iscsi_lun.c */
iscsi_status_t iscsi_lun_create(iscsi_sess_t *isp, uint16_t lun_num,
    uint8_t lun_addr_type, struct scsi_inquiry *inq, char *guid);
iscsi_status_t iscsi_lun_destroy(iscsi_hba_t *ihp,
    iscsi_lun_t *ilp);
void iscsi_lun_online(iscsi_hba_t *ihp,
    iscsi_lun_t *ilp);
iscsi_status_t iscsi_lun_offline(iscsi_hba_t *ihp,
    iscsi_lun_t *ilp, boolean_t lun_free);

/* iscsi_cmd.c */
void iscsi_cmd_state_machine(iscsi_cmd_t *icmdp,
    iscsi_cmd_event_t event, void *arg);
iscsi_cmd_t     *iscsi_cmd_alloc(iscsi_conn_t *icp, int km_flags);
void            iscsi_cmd_free(iscsi_cmd_t *icmdp);

/* iscsi_ioctl.c */
void * iscsi_ioctl_copyin(caddr_t arg, int mode, size_t size);
int iscsi_ioctl_copyout(void *data, size_t size, caddr_t arg, int mode);
iscsi_conn_list_t *iscsi_ioctl_conn_oid_list_get_copyin(caddr_t, int);
int iscsi_ioctl_conn_oid_list_get_copyout(iscsi_conn_list_t *, caddr_t, int);
boolean_t iscsi_ioctl_conn_oid_list_get(iscsi_hba_t *ihp,
    iscsi_conn_list_t *cl);
boolean_t iscsi_ioctl_conn_props_get(iscsi_hba_t *ihp, iscsi_conn_props_t *cp);
int iscsi_ioctl_sendtgts_get(iscsi_hba_t *ihp, iscsi_sendtgts_list_t *stl);
int iscsi_target_prop_mod(iscsi_hba_t *, iscsi_property_t *, int cmd);
int iscsi_set_params(iscsi_param_set_t *, iscsi_hba_t *, boolean_t);
int iscsi_get_persisted_param(uchar_t *, iscsi_param_get_t *,
    iscsi_login_params_t *);
void iscsi_set_default_login_params(iscsi_login_params_t *params);
int iscsi_ioctl_get_config_sess(iscsi_hba_t *ihp,
    iscsi_config_sess_t *ics);
int iscsi_ioctl_set_config_sess(iscsi_hba_t *ihp,
    iscsi_config_sess_t *ics);
int iscsi_ioctl_set_tunable_param(iscsi_hba_t *ihp,
    iscsi_tunable_object_t *tpss);
/* ioctls  prototypes */
int iscsi_get_param(iscsi_login_params_t *params,
    boolean_t valid_flag,
    iscsi_param_get_t *ipgp);

/* iscsid.c */
boolean_t iscsid_init(iscsi_hba_t *ihp);
boolean_t iscsid_start(iscsi_hba_t *ihp);
boolean_t iscsid_stop(iscsi_hba_t *ihp);
void iscsid_fini();
void iscsid_props(iSCSIDiscoveryProperties_t *props);
boolean_t iscsid_enable_discovery(iscsi_hba_t *ihp,
    iSCSIDiscoveryMethod_t idm, boolean_t poke);
boolean_t iscsid_disable_discovery(iscsi_hba_t *ihp,
    iSCSIDiscoveryMethod_t idm);
void iscsid_poke_discovery(iscsi_hba_t *ihp, iSCSIDiscoveryMethod_t method);
void iscsid_do_sendtgts(entry_t *discovery_addr);
void iscsid_do_isns_query_one_server(
```

```
    iscsi_hba_t *ihp, entry_t *isns_addr);
void iscsid_do_isns_query(iscsi_hba_t *ihp);
void iscsid_config_one(iscsi_hba_t *ihp,
    char *name, boolean_t protect);
void iscsid_config_all(iscsi_hba_t *ihp, boolean_t protect);
void iscsid_unconfig_one(iscsi_hba_t *ihp, char *name);
void iscsid_unconfig_all(iscsi_hba_t *ihp);
void isns_scn_callback(void *arg);
boolean_t iscsid_del(iscsi_hba_t *ihp, char *target_name,
    iSCSIDiscoveryMethod_t method, struct sockaddr *addr_dsc);
boolean_t iscsid_login_tgt(iscsi_hba_t *ihp, char *target_name,
    iSCSIDiscoveryMethod_t method, struct sockaddr *addr_dsc);
void iscsid_addr_to_sockaddr(int src_insize, void *src_addr, int src_port,
    struct sockaddr *dst_addr);
void iscsid_set_default_initiator_node_settings(iscsi_hba_t *ihp,
    boolean_t minimal);

void iscsi_send_sysevent(iscsi_hba_t *ihp, char *eventcalss,
    char *subclass, nvlist_t *np);
boolean_t iscsi_reconfig_boot_sess(iscsi_hba_t *ihp);
boolean_t iscsi_chk_bootlun_mpxio(iscsi_hba_t *ihp);
boolean_t iscsi_cmp_boot_ini_name(char *name);
boolean_t iscsi_cmp_boot_tgt_name(char *name);
boolean_t iscsi_client_request_service(iscsi_hba_t *ihp);
void iscsi_client_release_service(iscsi_hba_t *ihp);

extern void bcopy(const void *s1, void *s2, size_t n);
extern void bzero(void *s, size_t n);
/*
 * Here we need a contract for inet_ntop() and inet_pton()
 * in uts/common/inet/ip/inet_ntop.c
 */
extern char *inet_ntop(int af, const void *addr, char *buf, int addrlen);
extern int inet_pton(int af, char *inp, void *outp);

#ifdef __cplusplus
}
```

______***unchanged_portion_omitted_***

****
    9400 Wed Sep 26 12:51:50 2012
new/usr/src/uts/common/io/scsi/adapters/iscsi/iscsi_doorclt.c
inet_pton
****

```
/*
 * 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 2000 by Cisco Systems, Inc.  All rights reserved.
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
#endif /* ! codereview */
 *
 * iSCSI Software Initiator
 */

#include <sys/types.h>
#include <sys/errno.h>
#include <sys/conf.h>
#include <sys/cmn_err.h>
#include <sys/stat.h>
#include <sys/pathname.h>
#include <sys/door.h>
#include <sys/kmem.h>
#include <sys/socket.h>
#include <sys/fs/snode.h>
#include <netinet/in.h>

#include <sys/scsi/adapters/iscsi_door.h>
#include "iscsi.h"

#define ISCSI_DOOR_MAX_SEMA_VALUE       16

static boolean_t        iscsi_door_init = B_FALSE;
static ksema_t          iscsi_door_sema;
static krwlock_t        iscsi_door_lock;
static door_handle_t    iscsi_door_handle;

typedef struct _mybuffer {
        size_t          signature;
        size_t          size;
} mybuffer_t;

/*
 * iscsi_door_ini
 *
 * This function initializes the variables needed to handle the door upcall.
```

```
 */
boolean_t
iscsi_door_ini(void)
{
        ASSERT(!iscsi_door_init);
        if (!iscsi_door_init) {
                rw_init(
                    &iscsi_door_lock,
                    NULL,
                    RW_DRIVER,
                    NULL);

                sema_init(
                    &iscsi_door_sema,
                    ISCSI_DOOR_MAX_SEMA_VALUE,
                    NULL,
                    SEMA_DRIVER,
                    NULL);

                iscsi_door_handle = NULL;
                iscsi_door_init = B_TRUE;
                return (B_TRUE);
        }
        return (B_FALSE);
}

/*
 * iscsi_door_term
 *
 * This function releases the resources allocated to handle the door
 * upcall.  It disconnects from the door if currently connected.
 */
boolean_t
iscsi_door_term(void)
{
        ASSERT(iscsi_door_init);
        if (iscsi_door_init) {
                iscsi_door_init = B_FALSE;
                iscsi_door_unbind();
                rw_destroy(&iscsi_door_lock);
                sema_destroy(&iscsi_door_sema);
                return (B_TRUE);
        }
        return (B_FALSE);
}

/*
 * iscsi_door_bind
 *
 * This function tries to connect the iscsi_door.  If it succeeds
 * it keeps the vnode.
 */
boolean_t
iscsi_door_bind(
        int             did
)
{
        door_handle_t   new_handle;

        new_handle = door_ki_lookup(did);
        if (new_handle == NULL) {
                /* The lookup failed. */
                return (B_FALSE);
        }

        /* The new handle is stored.  If we had one, it is released. */
        rw_enter(&iscsi_door_lock, RW_WRITER);
        if (iscsi_door_handle != NULL) {
                door_ki_rele(iscsi_door_handle);
        }
        iscsi_door_handle = new_handle;
        rw_exit(&iscsi_door_lock);

        return (B_TRUE);
}

/*
 * iscsi_door_unbind
 *
 * This function releases the current door handle.
 */
void
iscsi_door_unbind(void)
{
        rw_enter(&iscsi_door_lock, RW_WRITER);
        if (iscsi_door_handle != NULL) {
                door_ki_rele(iscsi_door_handle);
                iscsi_door_handle = NULL;
        }
        rw_exit(&iscsi_door_lock);
}

/*
 * iscsi_door_upcall
 *
 * This function tries to call the iscsi_door.
 */
static
boolean_t
iscsi_door_upcall(door_arg_t *arg)
{
        int     error;

        /*
         * This semaphore limits the number of simultaneous calls
         * to the door.
         */
        sema_p(&iscsi_door_sema);
        /*
         * The mutex protecting the iscsi_door_handle is entered.
         */
        rw_enter(&iscsi_door_lock, RW_READER);

        if (iscsi_door_handle == NULL) {
                /* There’s no door handle. */
                rw_exit(&iscsi_door_lock);
                sema_v(&iscsi_door_sema);
                return (B_FALSE);
        }
        error = door_ki_upcall(iscsi_door_handle, arg);

        rw_exit(&iscsi_door_lock);
        sema_v(&iscsi_door_sema);

        if (error != 0) {
                return (B_FALSE);
        } else {
                return (B_TRUE);
        }
}

/*
```

```
 * kfreehostent
 *
 * This function frees the memory returned by kgetipnodebyname.
 */
void
kfreehostent(
        struct hostent          *hptr
)
{
        mybuffer_t              *buffer;

        ASSERT(hptr != NULL);
        if (hptr) {
                buffer = (mybuffer_t *)((char *)hptr - sizeof (mybuffer_t));
                ASSERT(buffer->signature == ISCSI_DOOR_REQ_SIGNATURE);
                if (buffer->signature == ISCSI_DOOR_REQ_SIGNATURE) {
                        kmem_free((void *)buffer, buffer->size);
                        return;
                }
        }
        /* A message should be logged here. */
}

/*
 * kgetipnodebyname
 *
 * This function builds a request that will be sent to the iscsi_door.
 * The iSCSI door after receiving the request calls getipnodebyaddr().
 * for more information on the input, output parameter and return value,
 * consult the man page for getipnodebyname().
 *
 * Before calling the iscsi door this function tries to do the conversion
 * locally.  If a name resolution is needed the iscsi door is called.
 *
 * There’s some limitations to the information returned by this function.
 * Only one address of the address list returned by getipnodebyname() is
 * returned.  The other parameters of the structure should be ignored.
 */
struct hostent *
kgetipnodebyname(
        const char      *name,
        int             af,
        int             flags,
        int             *error_num
)
{
        door_arg_t              arg;
        mybuffer_t              *buffer;
        size_t                  msg_size = ISCSI_DOOR_MAX_DATA_SIZE;
        size_t                  hostent_size = ISCSI_DOOR_MAX_DATA_SIZE;
        size_t                  buffer_size;
        getipnodebyname_req_t   *req;
        getipnodebyname_cnf_t   *cnf;
        struct hostent          *hptr;
        int                     i;
        uint16_t                *swap;


        buffer_size = msg_size + hostent_size + sizeof (mybuffer_t);
        buffer = (mybuffer_t *)kmem_zalloc(buffer_size, KM_SLEEP);

        if (buffer) {

                /*
                 * The buffer was successfully allocated.
                 *
```

```
	 *	  Buffer
	 *
	 * +--------------------+ <--- buffer
	 * |    mybuffer_t      |
	 * +--------------------+ <--- hptr
	 * |                    |
	 * |                    |
	 * |    hostent_size    |
	 * |                    |
	 * |                    |
	 * |                    |
	 * +--------------------+ <--- req, cnf
	 * |                    |
	 * |                    |
	 * |                    |
	 * |    msg_size        |
	 * |                    |
	 * |                    |
	 * |                    |
	 * +--------------------+
	 */
	buffer->signature = ISCSI_DOOR_REQ_SIGNATURE;
	buffer->size = buffer_size;

	hptr = (struct hostent *)((char *)buffer + sizeof (mybuffer_t));
	req = (getipnodebyname_req_t *)((char *)hptr + hostent_size);
	cnf = (getipnodebyname_cnf_t *)((char *)hptr + hostent_size);

	hostent_size -= sizeof (struct hostent);

	/*
	 * We try first locally.  If the conversion cannot be done
	 * by inet_pton the door is called.
	 * The cnf address is used as output buffer.
	 * inet_pton returns '1' if the conversion was successful.
	 */
	switch (af) {
	case AF_INET:
		hptr->h_length = sizeof (struct in_addr);
		break;
	case AF_INET6:
		hptr->h_length = sizeof (struct in6_addr);
		break;
	default:
		kfreehostent(hptr);
		*error_num = NO_RECOVERY;
		return (NULL);
	}
	if ((msg_size < hptr->h_length) ||
	    (hostent_size < sizeof (char *))) {
		kfreehostent(hptr);
		*error_num = NO_RECOVERY;
		return (NULL);
	}
	if (inet_pton(af, (char *)name, cnf) == 1) {
		/*
		 * inet_pton converted the string successfully.
		 * reset to network order.  swaps based on nfs code
		 */
		if (af == AF_INET) {
			*((uint32_t *)cnf) = htonl(*((uint32_t *)cnf));
		} else {
			for (swap = ((void *)cnf), i = 0;
			    i < hptr->h_length / sizeof (uint16_t);
			    i++) {
				swap[i] = htons(swap[i]);
```

```
			}
		}
		hptr->h_addrtype = af;
		hptr->h_addr_list = (char **)((char *)hptr +
		    sizeof (struct hostent));
		*hptr->h_addr_list = (char *)cnf;
		return (hptr);
	}

	/*
	 * The name couldn't ne converted by inet_pton.  The door is
	 * called.
	 */

	/* Header initialization. */
	req->hdr.signature = ISCSI_DOOR_REQ_SIGNATURE;
	req->hdr.version = ISCSI_DOOR_REQ_VERSION_1;
	req->hdr.opcode = ISCSI_DOOR_GETIPNODEBYNAME_REQ;

	/* Body initialization. */
	req->name_length = strlen(name);
	if (req->name_length >
	    (msg_size - sizeof (getipnodebyname_req_t) - 1)) {
		kfreehostent(hptr);
		*error_num = NO_RECOVERY;
		return (NULL);
	}

	req->name_offset = sizeof (getipnodebyname_req_t);
	req->af = af;
	req->flags = flags;
	bcopy(
	    name,
	    ((char *)req + req->name_offset),
	    req->name_length);

	/* Door argument initialization. */
	arg.data_ptr = (char *)req;
	arg.data_size = msg_size;
	arg.desc_num = 0;
	arg.desc_ptr = NULL;
	arg.rbuf = (char *)cnf;
	arg.rsize = msg_size;

	if (iscsi_door_upcall(&arg) == B_FALSE) {
		/* The door call failed */
		kfreehostent(hptr);
		*error_num = NO_RECOVERY;
		return (NULL);
	}

	/*
	 * The door call itself was successful.  The value returned
	 * in arg.rbuf should be cnf, but we never know.
	 */
	cnf = (getipnodebyname_cnf_t *)arg.rbuf;

	if ((cnf == NULL) ||
	    (arg.rsize < sizeof (getipnodebyname_cnf_t)) ||
	    (cnf->hdr.signature != ISCSI_DOOR_REQ_SIGNATURE) ||
	    (cnf->hdr.version != ISCSI_DOOR_REQ_VERSION_1) ||
	    (cnf->hdr.opcode != ISCSI_DOOR_GETIPNODEBYNAME_CNF) ||
	    ((cnf->hdr.status != ISCSI_DOOR_STATUS_SUCCESS) &&
	    (cnf->hdr.status != ISCSI_DOOR_STATUS_MORE))) {
		/* The door didn't like the request */
		kfreehostent(hptr);
```

```
                        *error_num = NO_RECOVERY;
                        return (NULL);
                }

                if (cnf->h_addr_list_length == 0) {
                        kfreehostent(hptr);
                        *error_num = HOST_NOT_FOUND;
                        return (NULL);
                }

                hptr->h_addrtype = cnf->h_addrtype;
                hptr->h_length = cnf->h_addrlen;
                hptr->h_addr_list = (char **)((char *)hptr +
                    sizeof (struct hostent));
                *hptr->h_addr_list = ((char *)cnf + cnf->h_addr_list_offset);
                return (hptr);
        } else {
                *error_num = NO_RECOVERY;
                return (NULL);
        }
}
```
______***unchanged_portion_omitted_***