```
**********************************************************
    4549 Thu Jan 17 14:42:21 2019
new/usr/src/lib/libstmfproxy/common/stmftransport.c
10108 libstmfproxy needs smatch fixes
**********************************************************
/*
 * 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 (c) 2018, Joyent, Inc.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <errno.h>
#include <syslog.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/stat.h>
#include <sys/sdt.h>
#include <signal.h>
#include <fcntl.h>
#include <libstmfproxy.h>

/*
 * NOTE:
 * This is demo code to be used with the existing demo proxy daemon
 * svc-stmfproxy in /usr/demo/comstar.
 */

struct _s_handle {
        int     sockfd;
};
______unchanged_portion_omitted_

static void *
```

```
pt_socket_connect(int server_node, char *server)
{
        int sfd, new_sfd;
        s_handle_t *sh = NULL;
        int on = 1;
        struct sockaddr_in cli_addr, serv_addr;
        struct  sockaddr_in sin;
        int cliLen = sizeof (cli_addr);

        if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) <= 0) {
                syslog(LOG_DAEMON|LOG_WARNING,
                    "socket() call failed: %d", errno);
                return (NULL);
        }

        if (server_node) {

                if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &on,
                    sizeof (on)) < 0) {
                        syslog(LOG_DAEMON|LOG_WARNING,
                            "setsockopt() failed: %d", errno);
                        goto serv_out;
                }

                bzero(&serv_addr, sizeof (serv_addr));
                serv_addr.sin_family = AF_INET;
                serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
                /* XXX get from smf? */
                serv_addr.sin_port = htons(6543);

                if (bind(sfd, (struct sockaddr *)&serv_addr,
                    sizeof (serv_addr)) < 0) {
                        syslog(LOG_DAEMON|LOG_WARNING, "bind() call failed: %d",
                            errno);
                        goto serv_out;
                }

                (void) listen(sfd, 5);

                new_sfd = accept(sfd, (struct sockaddr *)&cli_addr, &cliLen);

                if (new_sfd < 0) {
                        syslog(LOG_DAEMON|LOG_WARNING, "accept failed: %d",
                            errno);
                        goto serv_out;
                }
                sh = malloc(sizeof (*sh));
                sh->sockfd = new_sfd;
serv_out:
                (void) close(sfd);
                close(sfd);
        } else {
                struct  hostent *hp;

                /*
                 * Assume IP dot notation or if that fails, gethostbyname()
                 * If that fails, return
                 */
                if ((inet_aton(server, &sin.sin_addr)) == 0) {
                        if ((hp = gethostbyname(server)) != NULL) {
                                memcpy(&sin.sin_addr.s_addr, hp->h_addr,
                                    hp->h_length);
                        } else {
                                syslog(LOG_DAEMON|LOG_CRIT,
                                    "Cannot get IP address for %s", server);
                                (void) close(sfd);
                                return (NULL);
                        }
                } else {
                        fprintf(stderr,
                            "Sorry, cannot use ip address format\n");
                        (void) close(sfd);
                        return (NULL);
                }
                sin.sin_family = AF_INET;
                /* XXX pass in from smf */
                sin.sin_port = htons(6543);

                while (connect(sfd, (struct sockaddr *)&sin,
                    sizeof (sin)) < 0) {
                        (void) close(sfd);
                        close(sfd);
                        if (errno == ECONNREFUSED) {
                                /* get a fresh socket and retry */
                                sfd = socket(AF_INET, SOCK_STREAM, 0);
                                if (sfd < 0) {
                                        syslog(LOG_DAEMON|LOG_WARNING,
                                            "socket() call failed: %d", errno);
                                        return (NULL);
                                }
                                (void) sleep(2);
                                sleep(2);
                        } else {
                                syslog(LOG_DAEMON|LOG_CRIT,
                                    "Cannot connect %s - %d", server, errno);
                                return (NULL);
                        }
                }
                sh = malloc(sizeof (*sh));
                sh->sockfd = sfd;
        }
        return (sh);
}
```
______***unchanged_portion_omitted_***