

```
*****
6577 Tue Feb 11 10:48:28 2014
```

new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop\_arp.c

4587 snoop misdecodes DHCPv6 DHCPV6\_DUID\_LL identifiers

\*\*\*\*\*

```
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 2007 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
```

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

```
26 #include <sys/types.h>
27 #include <sys/errno.h>
28 #include <setjmp.h>
29 #include <stdio.h>
30 #include <stdlib.h>
31 #include <string.h>
32 #endif /* ! codereview */
33 #include <sys/socket.h>
34 #include <net/if.h>
35 #include <net/if_arp.h>
36 #include <netinet/in_sysm.h>
37 #include <netinet/in.h>
38 #include <netinet/ip.h>
39 #include <netinet/if_ether.h>
40 #include <netdb.h>
41 #include <net/if_types.h>
42 #include <arpa/inet.h>
43 #endif /* ! codereview */
```

```
45 #include "snoop.h"
```

```
47 extern char *dlc_header;
48 extern jmp_buf xdr_err;
```

```
50 static char *printip(unsigned char *);
51 static char *addrtoname_align(unsigned char *);
```

```
53 static char unarp_addr[] = "Unknown";
54 char *opname[] = {
55     "",
56     "ARP Request",
57     "ARP Reply",
58     "REVARP Request",
59     "REVARP Reply",
```

```
60 };
62 void
63 interpret_arp(int flags, struct arphdr *ap, int alen)
64 {
65     char *line;
66     extern char *src_name, *dst_name;
67     unsigned char *sip, *tip, *sha, *tha;
68     char *smacbuf = NULL, *dmacbuf = NULL;
69     int maclen;
70     ushort_t arpop;
71     boolean_t is_ip = B_FALSE;
73
74     /*
75      * Check that at least the generic ARP header was received.
76      */
77     if (sizeof (struct arphdr) > alen)
78         goto short_packet;
79
80     arpop = ntohs(ap->ar_op);
81     maclen = ap->ar_hln;
82     if (ntohs(ap->ar_pro) == ETHERTYPE_IP)
83         is_ip = B_TRUE;
84
85     sha = (unsigned char *) (ap + 1);
86     sip = sha + maclen;
87     tha = sip + ap->ar_pln;
88     tip = tha + maclen;
89
90     /*
91      * Check that the protocol/hardware addresses were received.
92      */
93     if ((tip + ap->ar_pln) > ((unsigned char *) ap + alen))
94         goto short_packet;
95
96     if (maclen == 0) {
97         smacbuf = dmacbuf = unarp_addr;
98     } else {
99         if (((flags & F_DTAIL) && is_ip) || (arpop == ARPOP_REPLY)) {
100             smacbuf = _link_ntoa(sha, NULL, maclen, IFT_OTHER);
101             if (smacbuf == NULL)
102                 pr_err("Warning: malloc failure");
103         }
104
105         if (((flags & F_DTAIL) && is_ip) || (arpop ==
106             REVARP_REQUEST) || (arpop == REVARP_REPLY)) {
107             dmacbuf = _link_ntoa(tha, NULL, maclen, IFT_OTHER);
108             if (dmacbuf == NULL)
109                 pr_err("Warning: malloc failure");
110         }
111     }
112
113     src_name = addrtoname_align(sip);
114
115     if (flags & F_SUM) {
116         line = get_sum_line();
117
118         switch (arpop) {
119             case ARPOP_REQUEST:
120                 (void) snprintf(line, MAXLINE, "ARP C Who is %s ?",
121                     printip(tip));
122                 break;
123             case ARPOP_REPLY:
124                 (void) snprintf(line, MAXLINE, "ARP R %s is %s",
125                     printip(sip), smacbuf);
```

```

126         dst_name = addrtoname_align(tip);
127         break;
128     case REVARP_REQUEST:
129         (void) snprintf(line, MAXLINE, "RARP C Who is %s ?",
130             dmacbuf);
131         break;
132     case REVARP_REPLY:
133         (void) snprintf(line, MAXLINE, "RARP R %s is %s",
134             dmacbuf, printip(tip));
135         dst_name = addrtoname_align(tip);
136         break;
137     }
138 }

140 if (flags & F_DTAIL) {
141     show_header("ARP: ", "ARP/RARP Frame", alen);
142     show_space();
143     (void) snprintf(get_line(0, 0), get_line_remain(),
144         "Hardware type = %d (%s)", ntohs(ap->ar_hrd),
145         arp_hatype(ntohs(ap->ar_hrd)));
146     (void) snprintf(get_line(0, 0), get_line_remain(),
147         "Protocol type = %04x (%s)", ntohs(ap->ar_pro),
148         print_etherbyte(ntohs(ap->ar_pro)));
149     (void) snprintf(get_line(0, 0), get_line_remain(),
150         "Length of hardware address = %d bytes", ap->ar_hln);
151     (void) snprintf(get_line(0, 0), get_line_remain(),
152         "Length of protocol address = %d bytes", ap->ar_pln);
153     (void) snprintf(get_line(0, 0), get_line_remain(),
154         "Opcode %d (%s)", arpop,
155         (arpop > REVARP_REPLY) ? opname[0] : opname[arpop]);

157     if (is_ip) {
158         (void) snprintf(get_line(0, 0), get_line_remain(),
159             "Sender's hardware address = %s", smacbuf);
160         (void) snprintf(get_line(0, 0), get_line_remain(),
161             "Sender's protocol address = %s",
162             printip(sip));
163         (void) snprintf(get_line(0, 0), get_line_remain(),
164             "Target hardware address = %s",
165             arpop == ARPOP_REQUEST ? "?" : dmacbuf);
166         (void) snprintf(get_line(0, 0), get_line_remain(),
167             "Target protocol address = %s",
168             arpop == REVARP_REQUEST ? "?" :
169             printip(tip));
170     }
171     show_trailer();
172 }

174 if (macflen != 0) {
175     free(smacbuf);
176     free(dmacbuf);
177 }
178 return;

180 short_packet:
181 if (flags & F_SUM) {
182     (void) snprintf(get_sum_line(), MAXLINE,
183         "ARP (short packet)");
184 } else if (flags & F_DTAIL) {
185     show_header("ARP: ", "ARP/RARP Frame", alen);
186     show_space();
187     (void) snprintf(get_line(0, 0), get_line_remain(),
188         "ARP (short packet)");
189 }
190 }

```

```

192 char *
193 printip(unsigned char *p)
194 {
195     static char buff[MAXHOSTNAMELEN + 32];
196     char *ap, *np;
197     struct in_addr a;

199     memcpy(&a, p, 4);
200     ap = (char *)inet_ntoa(a);
201     np = (char *)addrtoname(AF_INET, &a);
202     (void) snprintf(buff, MAXHOSTNAMELEN, "%s, %s", ap, np);
203     return (buff);
204 }

206 char *
207 addrtoname_align(unsigned char *p)
208 {
209     struct in_addr a;

211     memcpy(&a, p, 4);
212     return ((char *)addrtoname(AF_INET, &a));
213 }

215 /*
216  * These numbers are assigned by the IANA. See the arp-parameters registry.
217  * Only those values that are used within Solaris have #defines.
218  */
219 const char *
220 arp_hatype(int t)
221 {
222     switch (t) {
223     case ARPHRD_ETHER:
224         return ("Ethernet (10Mb)");
225     case 2:
226         return ("Experimental Ethernet (3Mb)");
227         return ("Experimental Ethernet (3MB)");
228     case 3:
229         return ("Amateur Radio AX.25");
230     case 4:
231         return ("Proteon ProNET Token Ring");
232     case 5:
233         return ("Chaos");
234     case ARPHRD_IEEE802:
235         return ("IEEE 802");
236     case 7:
237         return ("ARCNET");
238     case 8:
239         return ("Hyperchannel");
240     case 9:
241         return ("Lanstar");
242     case 10:
243         return ("Autonet");
244     case 11:
245         return ("LocalTalk");
246     case 12:
247         return ("LocalNet");
248     case 13:
249         return ("Ultra Link");
250     case 14:
251         return ("SMDS");
252     case ARPHRD_FRAME:
253         return ("Frame Relay");
254     case ARPHRD_ATM:
255         return ("ATM");
256     case ARPHRD_HDLC:
257         return ("HDLC");

```

```
257     case ARPHRD_FC:
258         return ("Fibre Channel");
259     case ARPHRD_IPATM:
260         return ("IP-ATM");
261     case ARPHRD_METRICOM:
262         return ("Metricom");
263 #endif /* ! codereview */
264     case ARPHRD_TUNNEL:
265         return ("Tunnel");
266     case ARPHRD_IB:
267         return ("IPIB");
268     };
269     return ("UNKNOWN");
270 }
```

```

*****
28711 Tue Feb 11 10:48:29 2014
new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop_dhcpv6.c
4587 snoop misdecodes DHCPv6 DHCPV6_DUID_LL identifiers
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

27 /*
28  * Dynamic Host Configuration Protocol version 6, for IPv6. Supports
29  * RFCs 3315, 3319, 3646, 3898, 4075, 4242, 4280, 4580, 4649, and 4704.
30  */

32 #include <ctype.h>
33 #endif /* ! codereview */
34 #include <stdio.h>
35 #include <stdlib.h>
36 #include <string.h>
37 #include <time.h>
38 #include <sys/types.h>
39 #include <sys/socket.h>
40 #include <netinet/in.h>
41 #include <netinet/dhcp6.h>
42 #include <arpa/inet.h>
43 #include <dhcp_impl.h>
44 #include <dhcp_inittab.h>

46 #include "snoop.h"

48 static const char *mtype_to_str(uint8_t);
49 static const char *option_to_str(uint8_t);
50 static const char *duidtype_to_str(uint16_t);
51 static const char *status_to_str(uint16_t);
52 static const char *entr_to_str(uint32_t);
53 static const char *reconf_to_str(uint8_t);
54 static const char *authproto_to_str(uint8_t);
55 static const char *authalg_to_str(uint8_t, uint8_t);
56 static const char *authrdm_to_str(uint8_t);
57 static const char *cwhat_to_str(uint8_t);
58 static const char *catype_to_str(uint8_t);
59 static void show_hex(const uint8_t *, int, const char *);
60 static void show_ascii(const uint8_t *, int, const char *);
61 static void show_address(const char *, const void *);

```

```

62 static void show_options(const uint8_t *, int);

64 int
65 interpret_dhcpv6(int flags, const uint8_t *data, int len)
66 {
67     int olen = len;
68     char *line, *lstart;
69     dhcpv6_relay_t d6r;
70     dhcpv6_message_t d6m;
71     uint_t optlen;
72     uint16_t statuscode;

74     if (len <= 0) {
75         (void) strcpy(get_sum_line(), "DHCPv6?", MAXLINE);
76         return (0);
77     }
78     if (flags & F_SUM) {
79         uint_t ias;
80         dhcpv6_option_t *d6o;
81         in6_addr_t link, peer;
82         char linkstr[INET6_ADDRSTRLEN];
83         char peerstr[INET6_ADDRSTRLEN];

85         line = lstart = get_sum_line();
86         line += sprintf(line, MAXLINE, "DHCPv6 %s",
87             mtype_to_str(data[0]));
88         if (data[0] == DHCPV6_MSG_RELAY_FORW ||
89             data[0] == DHCPV6_MSG_RELAY_REPL) {
90             if (len < sizeof (d6r)) {
91                 (void) strcpy(line, "?",
92                     MAXLINE - (line - lstart));
93                 return (olen);
94             }
95             /* Not much in DHCPv6 is aligned. */
96             (void) memcpy(&d6r, data, sizeof (d6r));
97             (void) memcpy(&link, d6r.d6r_linkaddr, sizeof (link));
98             (void) memcpy(&peer, d6r.d6r_peeraddr, sizeof (peer));
99             line += sprintf(line, MAXLINE - (line - lstart),
100                 " HC=%d link=%s peer=%s", d6r.d6r_hop_count,
101                 inet_ntop(AF_INET6, &link, linkstr,
102                     sizeof (linkstr)),
103                 inet_ntop(AF_INET6, &peer, peerstr,
104                     sizeof (peerstr)));
105             data += sizeof (d6r);
106             len -= sizeof (d6r);
107         } else {
108             if (len < sizeof (d6m)) {
109                 (void) strcpy(line, "?",
110                     MAXLINE - (line - lstart));
111                 return (olen);
112             }
113             (void) memcpy(&d6m, data, sizeof (d6m));
114             line += sprintf(line, MAXLINE - (line - lstart),
115                 " xid=%x", DHCPV6_GET_TRANSID(&d6m));
116             data += sizeof (d6m);
117             len -= sizeof (d6m);
118         }
119         ias = 0;
120         d6o = NULL;
121         while ((d6o = dhcpv6_find_option(data, len, d6o,
122             DHCPV6_OPT_IA_NA, NULL)) != NULL)
123             ias++;
124         if (ias > 0)
125             line += sprintf(line, MAXLINE - (line - lstart),
126                 " IAS=%u", ias);
127         d6o = dhcpv6_find_option(data, len, NULL,

```

```

128     DHCPV6_OPT_STATUS_CODE, &optlen);
129     optlen -= sizeof (*d6o);
130     if (d6o != NULL && optlen >= sizeof (statuscode)) {
131         (void) memcpy(&statuscode, d6o + 1,
132             sizeof (statuscode));
133         line += snprintf(line, MAXLINE - (line - lstart),
134             " status=%u", ntohs(statuscode));
135         optlen -= sizeof (statuscode);
136         if (optlen > 0) {
137             line += snprintf(line,
138                 MAXLINE - (line - lstart), " \">%.*s\\"",
139                 optlen, (char *) (d6o + 1) + 2);
140         }
141     }
142     d6o = dhcpv6_find_option(data, len, NULL,
143         DHCPV6_OPT_RELAY_MSG, &optlen);
144     optlen -= sizeof (*d6o);
145     if (d6o != NULL && optlen >= 1) {
146         line += snprintf(line, MAXLINE - (line - lstart),
147             " relay=%s", mtype_to_str(*(uint8_t *) (d6o + 1)));
148     }
149 } else if (flags & F_DTAIL) {
150     show_header("DHCPv6: ",
151         "Dynamic Host Configuration Protocol Version 6", len);
152     show_space();
153     (void) snprintf(get_line(0, 0), get_line_remain(),
154         "Message type (msg-type) = %u (%s)", data[0],
155         mtype_to_str(data[0]));
156     if (data[0] == DHCPV6_MSG_RELAY_FORW ||
157         data[0] == DHCPV6_MSG_RELAY_REPL) {
158         if (len < sizeof (d6r)) {
159             (void) strlcpy(get_line(0, 0), "Truncated",
160                 get_line_remain());
161             return (olen);
162         }
163         (void) memcpy(&d6r, data, sizeof (d6r));
164         (void) snprintf(get_line(0, 0), get_line_remain(),
165             "Hop count = %u", d6r.d6r_hop_count);
166         show_address("Link address", d6r.d6r_linkaddr);
167         show_address("Peer address", d6r.d6r_peeraddr);
168         data += sizeof (d6r);
169         len -= sizeof (d6r);
170     } else {
171         if (len < sizeof (d6m)) {
172             (void) strlcpy(get_line(0, 0), "Truncated",
173                 get_line_remain());
174             return (olen);
175         }
176         (void) memcpy(&d6m, data, sizeof (d6m));
177         (void) snprintf(get_line(0, 0), get_line_remain(),
178             "Transaction ID = %x", DHCPV6_GET_TRANSID(&d6m));
179         data += sizeof (d6m);
180         len -= sizeof (d6m);
181     }
182     show_space();
183     show_options(data, len);
184     show_space();
185 }
186 return (olen);
187 }

189 static const char *
190 mtype_to_str(uint8_t mtype)
191 {
192     switch (mtype) {
193     case DHCPV6_MSG_SOLICIT:

```

```

194         return ("Solicit");
195     case DHCPV6_MSG_ADVERTISE:
196         return ("Advertise");
197     case DHCPV6_MSG_REQUEST:
198         return ("Request");
199     case DHCPV6_MSG_CONFIRM:
200         return ("Confirm");
201     case DHCPV6_MSG_RENEW:
202         return ("Renew");
203     case DHCPV6_MSG_REBIND:
204         return ("Rebind");
205     case DHCPV6_MSG_REPLY:
206         return ("Reply");
207     case DHCPV6_MSG_RELEASE:
208         return ("Release");
209     case DHCPV6_MSG_DECLINE:
210         return ("Decline");
211     case DHCPV6_MSG_RECONFIGURE:
212         return ("Reconfigure");
213     case DHCPV6_MSG_INFO_REQ:
214         return ("Information-Request");
215     case DHCPV6_MSG_RELAY_FORW:
216         return ("Relay-Forward");
217     case DHCPV6_MSG_RELAY_REPL:
218         return ("Relay-Reply");
219     default:
220         return ("Unknown");
221     }
222 }

224 static const char *
225 option_to_str(uint8_t mtype)
226 {
227     switch (mtype) {
228     case DHCPV6_OPT_CLIENTID:
229         return ("Client Identifier");
230     case DHCPV6_OPT_SERVERID:
231         return ("Server Identifier");
232     case DHCPV6_OPT_IA_NA:
233         return ("Identity Association for Non-temporary Addresses");
234     case DHCPV6_OPT_IA_TA:
235         return ("Identity Association for Temporary Addresses");
236     case DHCPV6_OPT_IAADDR:
237         return ("IA Address");
238     case DHCPV6_OPT_ORO:
239         return ("Option Request");
240     case DHCPV6_OPT_PREFERENCE:
241         return ("Preference");
242     case DHCPV6_OPT_ELAPSED_TIME:
243         return ("Elapsed Time");
244     case DHCPV6_OPT_RELAY_MSG:
245         return ("Relay Message");
246     case DHCPV6_OPT_AUTH:
247         return ("Authentication");
248     case DHCPV6_OPT_UNICAST:
249         return ("Server Unicast");
250     case DHCPV6_OPT_STATUS_CODE:
251         return ("Status Code");
252     case DHCPV6_OPT_RAPID_COMMIT:
253         return ("Rapid Commit");
254     case DHCPV6_OPT_USER_CLASS:
255         return ("User Class");
256     case DHCPV6_OPT_VENDOR_CLASS:
257         return ("Vendor Class");
258     case DHCPV6_OPT_VENDOR_OPT:
259         return ("Vendor-specific Information");

```

```

260     case DHCPV6_OPT_INTERFACE_ID:
261         return ("Interface-Id");
262     case DHCPV6_OPT_RECONF_MSG:
263         return ("Reconfigure Message");
264     case DHCPV6_OPT_RECONF_ACC:
265         return ("Reconfigure Accept");
266     case DHCPV6_OPT_SIP_NAMES:
267         return ("SIP Servers Domain Name List");
268     case DHCPV6_OPT_SIP_ADDR:
269         return ("SIP Servers IPv6 Address List");
270     case DHCPV6_OPT_DNS_ADDR:
271         return ("DNS Recursive Name Server");
272     case DHCPV6_OPT_DNS_SEARCH:
273         return ("Domain Search List");
274     case DHCPV6_OPT_IA_PD:
275         return ("Identity Association for Prefix Delegation");
276     case DHCPV6_OPT_IAPREFIX:
277         return ("IA_PD Prefix");
278     case DHCPV6_OPT_NIS_SERVERS:
279         return ("Network Information Service Servers");
280     case DHCPV6_OPT_NIS_DOMAIN:
281         return ("Network Information Service Domain Name");
282     case DHCPV6_OPT_Sntp_SERVERS:
283         return ("Simple Network Time Protocol Servers");
284     case DHCPV6_OPT_INFO_REFTIME:
285         return ("Information Refresh Time");
286     case DHCPV6_OPT_BCMCS_SRV_D:
287         return ("BCMCS Controller Domain Name List");
288     case DHCPV6_OPT_BCMCS_SRV_A:
289         return ("BCMCS Controller IPv6 Address");
290     case DHCPV6_OPT_GEOCONF_CVC:
291         return ("Civic Location");
292     case DHCPV6_OPT_REMOTE_ID:
293         return ("Relay Agent Remote-ID");
294     case DHCPV6_OPT_SUBSCRIBER:
295         return ("Relay Agent Subscriber-ID");
296     case DHCPV6_OPT_CLIENT_FQDN:
297         return ("Client FQDN");
298     default:
299         return ("Unknown");
300 }
301 }
302
303 static const char *
304 duidtype_to_str(uint16_t dtype)
305 {
306     switch (dtype) {
307     case DHCPV6_DUID_LL:
308         return ("Link-layer Address Plus Time");
309     case DHCPV6_DUID_EN:
310         return ("Enterprise Number");
311     case DHCPV6_DUID_LL:
312         return ("Link-layer Address");
313     default:
314         return ("Unknown");
315     }
316 }
317
318 static const char *
319 status_to_str(uint16_t status)
320 {
321     switch (status) {
322     case DHCPV6_STAT_SUCCESS:
323         return ("Success");
324     case DHCPV6_STAT_UNSPECFAIL:
325         return ("Failure, reason unspecified");

```

```

326     case DHCPV6_STAT_NOADDRS:
327         return ("No addresses for IAs");
328     case DHCPV6_STAT_NOBINDING:
329         return ("Client binding unavailable");
330     case DHCPV6_STAT_NOTONLINK:
331         return ("Prefix not on link");
332     case DHCPV6_STAT_USEMCAST:
333         return ("Use multicast");
334     case DHCPV6_STAT_NOPREFIX:
335         return ("No prefix available");
336     default:
337         return ("Unknown");
338     }
339 }
340
341 static const char *
342 entr_to_str(uint32_t entr)
343 {
344     switch (entr) {
345     case DHCPV6_SUN_ENT:
346         return ("Sun Microsystems");
347     default:
348         return ("Unknown");
349     }
350 }
351
352 static const char *
353 reconf_to_str(uint8_t msgtype)
354 {
355     switch (msgtype) {
356     case DHCPV6_RECONF_RENEW:
357         return ("Renew");
358     case DHCPV6_RECONF_INFO:
359         return ("Information-request");
360     default:
361         return ("Unknown");
362     }
363 }
364
365 static const char *
366 authproto_to_str(uint8_t aproto)
367 {
368     switch (aproto) {
369     case DHCPV6_PROTO_DELAYED:
370         return ("Delayed");
371     case DHCPV6_PROTO_RECONFIG:
372         return ("Reconfigure Key");
373     default:
374         return ("Unknown");
375     }
376 }
377
378 static const char *
379 authalg_to_str(uint8_t aproto, uint8_t aalg)
380 {
381     switch (aproto) {
382     case DHCPV6_PROTO_DELAYED:
383     case DHCPV6_PROTO_RECONFIG:
384         switch (aalg) {
385         case DHCPV6_ALG_HMAC_MD5:
386             return ("HMAC-MD5 Signature");
387         default:
388             return ("Unknown");
389         }
390     }
391     break;
392     default:

```

```

392         return ("Unknown");
393     }
394 }

396 static const char *
397 authrdm_to_str(uint8_t ardm)
398 {
399     switch (ardm) {
400     case DHCPV6_RDM_MONOCNT:
401         return ("Monotonic Counter");
402     default:
403         return ("Unknown");
404     }
405 }

407 static const char *
408 cwhat_to_str(uint8_t what)
409 {
410     switch (what) {
411     case DHCPV6_CWHAT_SERVER:
412         return ("Server");
413     case DHCPV6_CWHAT_NETWORK:
414         return ("Network");
415     case DHCPV6_CWHAT_CLIENT:
416         return ("Client");
417     default:
418         return ("Unknown");
419     }
420 }

422 static const char *
423 catype_to_str(uint8_t catype)
424 {
425     switch (catype) {
426     case CIVICADDR_LANG:
427         return ("Language; RFC 2277");
428     case CIVICADDR_A1:
429         return ("National division (state)");
430     case CIVICADDR_A2:
431         return ("County");
432     case CIVICADDR_A3:
433         return ("City");
434     case CIVICADDR_A4:
435         return ("City division");
436     case CIVICADDR_A5:
437         return ("Neighborhood");
438     case CIVICADDR_A6:
439         return ("Street group");
440     case CIVICADDR_PRD:
441         return ("Leading street direction");
442     case CIVICADDR_POD:
443         return ("Trailing street suffix");
444     case CIVICADDR_STS:
445         return ("Street suffix or type");
446     case CIVICADDR_HNO:
447         return ("House number");
448     case CIVICADDR_HNS:
449         return ("House number suffix");
450     case CIVICADDR_LMK:
451         return ("Landmark");
452     case CIVICADDR_LOC:
453         return ("Additional location information");
454     case CIVICADDR_NAM:
455         return ("Name/occupant");
456     case CIVICADDR_PC:
457         return ("Postal Code/ZIP");

```

```

458     case CIVICADDR_BLD:
459         return ("Building");
460     case CIVICADDR_UNIT:
461         return ("Unit/apt/suite");
462     case CIVICADDR_FLR:
463         return ("Floor");
464     case CIVICADDR_ROOM:
465         return ("Room number");
466     case CIVICADDR_TYPE:
467         return ("Place type");
468     case CIVICADDR_PCN:
469         return ("Postal community name");
470     case CIVICADDR_POBOX:
471         return ("Post office box");
472     case CIVICADDR_ADDL:
473         return ("Additional code");
474     case CIVICADDR_SEAT:
475         return ("Seat/desk");
476     case CIVICADDR_ROAD:
477         return ("Primary road or street");
478     case CIVICADDR_RSEC:
479         return ("Road section");
480     case CIVICADDR_RBRA:
481         return ("Road branch");
482     case CIVICADDR_RSBR:
483         return ("Road sub-branch");
484     case CIVICADDR_SPRE:
485         return ("Street name pre-modifier");
486     case CIVICADDR_SPOST:
487         return ("Street name post-modifier");
488     case CIVICADDR_SCRIPT:
489         return ("Script");
490     default:
491         return ("Unknown");
492     }
493 }

495 static void
496 show_hex(const uint8_t *data, int len, const char *name)
497 {
498     char buffer[16 * 3 + 1];
499     int nlen;
500     int i;
501     char sep;

503     nlen = strlen(name);
504     sep = '=';
505     while (len > 0) {
506         for (i = 0; i < 16 && i < len; i++)
507             (void) snprintf(buffer + 3 * i, 4, "%02x", *data++);
508         (void) snprintf(get_line(0, 0), get_line_remain(), "%*s %c%s",
509             nlen, name, sep, buffer);
510         name = "";
511         sep = ' ';
512         len -= i;
513     }
514 }

516 static void
517 show_ascii(const uint8_t *data, int len, const char *name)
518 {
519     char buffer[64], *bp;
520     int nlen;
521     int i;
522     char sep;

```

```

524     nlen = strlen(name);
525     sep = '=';
526     while (len > 0) {
527         bp = buffer;
528         for (i = 0; i < sizeof (buffer) - 4 && len > 0; len--) {
529             if (!isascii(*data) || !isprint(*data))
530                 bp += snprintf(bp, 5, "\\%03o", *data++);
531             else
532                 *bp++;
533         }
534         *bp = '\0';
535         (void) snprintf(get_line(0, 0), get_line_remain(),
536             "%*s %c \"%s\"", nlen, name, sep, buffer);
537         sep = ' ';
538         name = "";
539     }
540 }

542 static void
543 show_address(const char *addrname, const void *aptr)
544 {
545     char *hname;
546     char addrstr[INET6_ADDRSTRLEN];
547     in6_addr_t addr;

549     (void) memcpy(&addr, aptr, sizeof (in6_addr_t));
550     (void) inet_ntop(AF_INET6, &addr, addrstr, sizeof (addrstr));
551     hname = addrtoname(AF_INET6, &addr);
552     if (strcmp(hname, addrstr) == 0) {
553         (void) snprintf(get_line(0, 0), get_line_remain(), "%s = %s",
554             addrname, addrstr);
555     } else {
556         (void) snprintf(get_line(0, 0), get_line_remain(),
557             "%s = %s (%s)", addrname, addrstr, hname);
558     }
559 }

561 static void
562 nest_options(const uint8_t *data, uint_t olen, char *prefix, char *title)
563 {
564     char *str, *oldnest, *oldprefix;

566     if (olen <= 0)
567         return;
568     oldprefix = prot_prefix;
569     oldnest = prot_nest_prefix;
570     str = malloc(strlen(prot_nest_prefix) + strlen(prot_prefix) + 1);
571     if (str == NULL) {
572         prot_nest_prefix = prot_prefix;
573     } else {
574         (void) sprintf(str, "%s%s", prot_nest_prefix, prot_prefix);
575         prot_nest_prefix = str;
576     }
577     show_header(prefix, title, 0);
578     show_options(data, olen);
579     free(str);
580     prot_prefix = oldprefix;
581     prot_nest_prefix = oldnest;
582 }

584 static void
585 show_options(const uint8_t *data, int len)
586 {
587     dhcpv6_option_t d6o;
588     uint_t olen, retlen;
589     uint16_t val16;

```

```

590     uint16_t type;
591     uint32_t val32;
592     const uint8_t *ostart;
593     char *str, *sp;
594     char *oldnest;

596     /*
597      * Be very careful with negative numbers; ANSI signed/unsigned
598      * comparison doesn't work as expected.
599      */
600     while (len >= (signed)sizeof (d6o)) {
601         (void) memcpy(&d6o, data, sizeof (d6o));
602         d6o.d6o_code = ntohs(d6o.d6o_code);
603         d6o.d6o_len = olen = ntohs(d6o.d6o_len);
604         (void) snprintf(get_line(0, 0), get_line_remain(),
605             "Option Code = %u (%s)", d6o.d6o_code,
606             option_to_str(d6o.d6o_code));
607         ostart = data += sizeof (d6o);
608         len -= sizeof (d6o);
609         if (olen > len) {
610             (void) strcpy(get_line(0, 0), "Option truncated",
611                 get_line_remain());
612             olen = len;
613         }
614         switch (d6o.d6o_code) {
615             case DHCPV6_OPT_CLIENTID:
616             case DHCPV6_OPT_SERVERID:
617                 if (olen < sizeof (vall6))
618                     break;
619                 (void) memcpy(&vall6, data, sizeof (vall6));
620                 data += sizeof (vall6);
621                 olen -= sizeof (vall6);
622                 type = ntohs(vall6);
623                 (void) snprintf(get_line(0, 0), get_line_remain(),
624                     " DUID Type = %u (%s)", type,
625                     duidtype_to_str(type));
626                 if (type == DHCPV6_DUID_LL || type == DHCPV6_DUID_LL) {
627                     if (olen < sizeof (vall6))
628                         break;
629                     (void) memcpy(&vall6, data, sizeof (vall6));
630                     data += sizeof (vall6);
631                     olen -= sizeof (vall6);
632                     vall6 = ntohs(vall6);
633                     (void) snprintf(get_line(0, 0),
634                         get_line_remain(),
635                         " Hardware Type = %u (%s)", vall6,
636                         arp_hatype(vall6));
637                     arp_hatype(vall6);
638                 }
639                 if (type == DHCPV6_DUID_LL) {
640                     time_t timevalue;

641                     if (olen < sizeof (val32))
642                         break;
643                     (void) memcpy(&val32, data, sizeof (val32));
644                     data += sizeof (val32);
645                     olen -= sizeof (val32);
646                     timevalue = ntohl(val32) + DUID_TIME_BASE;
647                     (void) snprintf(get_line(0, 0),
648                         get_line_remain(),
649                         " Time = %lu (%.24s)", ntohl(val32),
650                         ctime(&timevalue));
651                 }
652                 if (type == DHCPV6_DUID_EN) {
653                     if (olen < sizeof (val32))
654                         break;

```



```

787     (void) snprintf(get_line(0, 0), get_line_remain(),
788                  " Algorithm = %u (%s)", d6a.d6a_alg,
789                  authalg_to_str(d6a.d6a_proto, d6a.d6a_alg));
790     (void) snprintf(get_line(0, 0), get_line_remain(),
791                  " Replay Detection Method = %u (%s)", d6a.d6a_rdm,
792                  authrdm_to_str(d6a.d6a_rdm));
793     show_hex(d6a.d6a_replay, sizeof (d6a.d6a_replay),
794            " RDM Data");
795     if (olen > 0)
796         show_hex(data, olen, " Auth Info");
797     break;
798 }
799 case DHCPV6_OPT_UNICAST:
800     if (olen >= sizeof (in6_addr_t))
801         show_address(" Server Address", data);
802     break;
803 case DHCPV6_OPT_STATUS_CODE:
804     if (olen < sizeof (vall6))
805         break;
806     (void) memcpy(&vall6, data, sizeof (vall6));
807     vall6 = ntohs(vall6);
808     (void) snprintf(get_line(0, 0), get_line_remain(),
809                  " Status Code = %u (%s)", vall6,
810                  status_to_str(vall6));
811     data += sizeof (vall6);
812     olen -= sizeof (vall6);
813     if (olen > 0)
814         (void) snprintf(get_line(0, 0),
815                      get_line_remain(), " Text = \"%.*s\"",
816                      olen, data);
817     break;
818 case DHCPV6_OPT_VENDOR_CLASS:
819     if (olen < sizeof (val32))
820         break;
821     (void) memcpy(&val32, data, sizeof (val32));
822     data += sizeof (val32);
823     olen -= sizeof (val32);
824     val32 = ntohl(val32);
825     (void) snprintf(get_line(0, 0), get_line_remain(),
826                  " Enterprise Number = %lu (%s)", val32,
827                  entr_to_str(val32));
828     /* FALLTHROUGH */
829 case DHCPV6_OPT_USER_CLASS:
830     while (olen >= sizeof (vall6)) {
831         (void) memcpy(&vall6, data, sizeof (vall6));
832         data += sizeof (vall6);
833         olen -= sizeof (vall6);
834         vall6 = ntohs(vall6);
835         if (vall6 > olen) {
836             (void) strcpy(get_line(0, 0),
837                          " Truncated class",
838                          get_line_remain());
839             vall6 = olen;
840         }
841         show_hex(data, olen, " Class");
842         data += vall6;
843         olen -= vall6;
844     }
845     break;
846 case DHCPV6_OPT_VENDOR_OPT: {
847     dhcpv6_option_t sd6o;
849     if (olen < sizeof (val32))
850         break;
851     (void) memcpy(&val32, data, sizeof (val32));
852     data += sizeof (val32);

```

```

853     olen -= sizeof (val32);
854     val32 = ntohl(val32);
855     (void) snprintf(get_line(0, 0), get_line_remain(),
856                  " Enterprise Number = %lu (%s)", val32,
857                  entr_to_str(val32));
858     while (olen >= sizeof (sd6o)) {
859         (void) memcpy(&sd6o, data, sizeof (sd6o));
860         sd6o.d6o_code = ntohs(sd6o.d6o_code);
861         sd6o.d6o_len = ntohs(sd6o.d6o_len);
862         (void) snprintf(get_line(0, 0),
863                      get_line_remain(),
864                      " Vendor Option Code = %u", d6o.d6o_code);
865         data += sizeof (d6o);
866         olen -= sizeof (d6o);
867         if (sd6o.d6o_len > olen) {
868             (void) strcpy(get_line(0, 0),
869                          " Vendor Option truncated",
870                          get_line_remain());
871             sd6o.d6o_len = olen;
872         }
873         if (sd6o.d6o_len > 0) {
874             show_hex(data, sd6o.d6o_len,
875                    " Data");
876             data += sd6o.d6o_len;
877             olen -= sd6o.d6o_len;
878         }
879     }
880     break;
881 }
882 case DHCPV6_OPT_REMOTE_ID:
883     if (olen < sizeof (val32))
884         break;
885     (void) memcpy(&val32, data, sizeof (val32));
886     data += sizeof (val32);
887     olen -= sizeof (val32);
888     val32 = ntohl(val32);
889     (void) snprintf(get_line(0, 0), get_line_remain(),
890                  " Enterprise Number = %lu (%s)", val32,
891                  entr_to_str(val32));
892     /* FALLTHROUGH */
893 case DHCPV6_OPT_INTERFACE_ID:
894 case DHCPV6_OPT_SUBSCRIBER:
895     if (olen > 0)
896         show_hex(data, olen, " ID");
897     break;
898 case DHCPV6_OPT_RECONF_MSG:
899     if (olen > 0) {
900         (void) snprintf(get_line(0, 0),
901                      get_line_remain(),
902                      " Message Type = %u (%s)", *data,
903                      reconf_to_str(*data));
904     }
905     break;
906 case DHCPV6_OPT_SIP_NAMES:
907 case DHCPV6_OPT_DNS_SEARCH:
908 case DHCPV6_OPT_NIS_DOMAIN:
909 case DHCPV6_OPT_BCMCS_SRV_D: {
910     dhcp_symbol_t *symp;
911     char *sp2;
913     symp = inittab_getbycode(
914         ITAB_CAT_STANDARD | ITAB_CAT_V6, ITAB_CONS_SNOOP,
915         d6o.d6o_code);
916     if (symp != NULL) {
917         str = inittab_decode(symp, data, olen, B_TRUE);
918         if (str != NULL) {

```

```

919         sp = str;
920         do {
921             sp2 = strchr(sp, ' ');
922             if (sp2 != NULL)
923                 *sp2++ = '\0';
924             (void) snprintf(get_line(0, 0),
925                 get_line_remain(),
926                 " Name = %s", sp);
927         } while ((sp = sp2) != NULL);
928         free(str);
929     }
930     free(symp);
931 }
932 break;
933 }
934 case DHCPV6_OPT_SIP_ADDR:
935 case DHCPV6_OPT_DNS_ADDR:
936 case DHCPV6_OPT_NIS_SERVERS:
937 case DHCPV6_OPT_SNTP_SERVERS:
938 case DHCPV6_OPT_BCMLS_SRV_A:
939     while (olen >= sizeof (in6_addr_t)) {
940         show_address(" Address", data);
941         data += sizeof (in6_addr_t);
942         olen -= sizeof (in6_addr_t);
943     }
944     break;
945 case DHCPV6_OPT_IAPREFIX: {
946     dhcpv6_iaprefix_t d6ip;
947
948     if (olen < DHCPV6_IAPREFIX_SIZE - sizeof (d6o))
949         break;
950     (void) memcpy(&d6ip, data - sizeof (d6o),
951         DHCPV6_IAPREFIX_SIZE);
952     data += DHCPV6_IAPREFIX_SIZE - sizeof (d6o);
953     olen -= DHCPV6_IAPREFIX_SIZE - sizeof (d6o);
954     show_address(" Prefix", d6ip.d6ip_addr);
955     (void) snprintf(get_line(0, 0), get_line_remain(),
956         " Preferred lifetime = %u seconds",
957         ntohl(d6ip.d6ip_preftime));
958     (void) snprintf(get_line(0, 0), get_line_remain(),
959         " Valid lifetime = %u seconds",
960         ntohl(d6ip.d6ip_vallife));
961     (void) snprintf(get_line(0, 0), get_line_remain(),
962         " Prefix length = %u", d6ip.d6ip_preflen);
963     nest_options(data, olen, "ADDR: ", "Address");
964     break;
965 }
966 case DHCPV6_OPT_INFO_REFTIME:
967     if (olen < sizeof (val32))
968         break;
969     (void) memcpy(&val32, data, sizeof (val32));
970     (void) snprintf(get_line(0, 0), get_line_remain(),
971         " Refresh Time = %lu seconds", ntohl(val32));
972     break;
973 case DHCPV6_OPT_GEOCONF_CVC: {
974     dhcpv6_civic_t d6c;
975     int solen;
976
977     if (olen < DHCPV6_CIVIC_SIZE - sizeof (d6o))
978         break;
979     (void) memcpy(&d6c, data - sizeof (d6o),
980         DHCPV6_CIVIC_SIZE);
981     data += DHCPV6_CIVIC_SIZE - sizeof (d6o);
982     olen -= DHCPV6_CIVIC_SIZE - sizeof (d6o);
983     (void) snprintf(get_line(0, 0), get_line_remain(),
984         " What Location = %u (%s)", d6c.d6c_what,

```

```

985         cwhat_to_str(d6c.d6c_what));
986     (void) snprintf(get_line(0, 0), get_line_remain(),
987         " Country Code = %.*s", sizeof (d6c.d6c_cc),
988         d6c.d6c_cc);
989     while (olen >= 2) {
990         (void) snprintf(get_line(0, 0),
991             get_line_remain(),
992             " CA Element = %u (%s)", *data,
993             catype_to_str(*data));
994         solen = data[1];
995         data += 2;
996         olen -= 2;
997         if (solen > olen) {
998             (void) strcpy(get_line(0, 0),
999                 " CA Element truncated",
1000                 get_line_remain());
1001             solen = olen;
1002         }
1003         if (solen > 0) {
1004             show_ascii(data, solen, " CA Data");
1005             data += solen;
1006             olen -= solen;
1007         }
1008     }
1009     break;
1010 }
1011 case DHCPV6_OPT_CLIENT_FQDN: {
1012     dhcp_symbol_t *symp;
1013
1014     if (olen == 0)
1015         break;
1016     (void) snprintf(get_line(0, 0), get_line_remain(),
1017         " Flags = %02x", *data);
1018     (void) snprintf(get_line(0, 0), get_line_remain(),
1019         " %s", getflag(*data, DHCPV6_FQDNF_S,
1020             "Perform AAAA RR updates", "No AAAA RR updates"));
1021     (void) snprintf(get_line(0, 0), get_line_remain(),
1022         " %s", getflag(*data, DHCPV6_FQDNF_O,
1023             "Server override updates",
1024             "No server override updates"));
1025     (void) snprintf(get_line(0, 0), get_line_remain(),
1026         " %s", getflag(*data, DHCPV6_FQDNF_N,
1027             "Server performs no updates",
1028             "Server performs updates"));
1029     symp = inittab_getbycode(
1030         ITAB_CAT_STANDARD | ITAB_CAT_V6, ITAB_CONS_SNOOP,
1031         d6o.d6o_code);
1032     if (symp != NULL) {
1033         str = inittab_decode(symp, data, olen, B_TRUE);
1034         if (str != NULL) {
1035             (void) snprintf(get_line(0, 0),
1036                 get_line_remain(),
1037                 " FQDN = %s", str);
1038             free(str);
1039         }
1040         free(symp);
1041     }
1042     break;
1043 }
1044 }
1045 data = ostart + d6o.d6o_len;
1046 len -= d6o.d6o_len;
1047 }
1048 if (len != 0) {
1049     (void) strcpy(get_line(0, 0), "Option entry truncated",
1050         get_line_remain());

```

new/usr/src/cmd/cmd-inet/usr.sbin/snoop/snoop\_dhcpv6.c

17

```
1051     }  
1052 }  
_____unchanged_portion_omitted_____
```