

```
*****
```

```
79084 Mon May 5 11:11:28 2014
```

```
new/usr/src/uts/common/inet/ipf/ip_fil_solaris.c
```

```
4787 ipf: remove rate_limit_message
```

```
*****
```

```
_____unchanged_portion_omitted_____
```

```
1288 #include <sys/time.h>
1289 #include <sys/varargs.h>
```

```
1291 #ifndef _KERNEL
1292 #include <stdio.h>
1293 #endif
```

```
1295 #define NULLADDR_RATE_LIMIT 10 /* 10 seconds */
```

```
1298 /*
1299  * Print out warning message at rate-limited speed.
1300  */
1301 static void rate_limit_message(ipf_stack_t *ifs,
1302                               int rate, const char *message, ...)
```

```
1303 {
1304     static time_t last_time = 0;
1305     time_t now;
1306     va_list args;
1307     char msg_buf[256];
1308     int need_printed = 0;
1309
1310     now = ddi_get_time();
1311
1312     /* make sure, no multiple entries */
1313     ASSERT(MUTEX_NOT_HELD(&(ifs->ifs_ipf_rw.ipf_lk)));
1314     MUTEX_ENTER(&ifs->ifs_ipf_rw);
1315     if (now - last_time >= rate) {
1316         need_printed = 1;
1317         last_time = now;
1318     }
1319     MUTEX_EXIT(&ifs->ifs_ipf_rw);
```

```
1321     if (need_printed) {
1322         va_start(args, message);
1323         (void)vsnprintf(msg_buf, 255, message, args);
1324         va_end(args);
1325     #ifndef _KERNEL
1326         cmn_err(CE_WARN, msg_buf);
1327     #else
1328         fprintf(std_err, msg_buf);
1329     #endif
1330     }
1331 }
```

```
1295 /*
1296  * Return the first IP Address associated with an interface
1297  * For IPv6, we walk through the list of logical interfaces and return
1298  * the address of the first one that isn't a link-local interface.
1299  * We can't assume that it is :1 because another link-local address
1300  * may have been assigned there.
1301  */
1302 /*ARGSUSED*/
1303 int fr_ifpaddr(v, atype, ifptr, inp, inpmask, ifs)
1304 int v, atype;
1305 void *ifptr;
1306 struct in_addr *inp, *inpmask;
1307 ipf_stack_t *ifs;
1308 {
```

```
1309     struct sockaddr_in6 v6addr[2];
1310     struct sockaddr_in v4addr[2];
1311     net_ifaddr_t type[2];
1312     net_handle_t net_data;
1313     phy_if_t phyif;
1314     void *array;
1315
1316     switch (v)
1317     {
1318     case 4:
1319         net_data = ifs->ifs_ipf_ipv4;
1320         array = v4addr;
1321         break;
1322     case 6:
1323         net_data = ifs->ifs_ipf_ipv6;
1324         array = v6addr;
1325         break;
1326     default:
1327         net_data = NULL;
1328         break;
1329     }
1330
1331     if (net_data == NULL)
1332         return -1;
1333
1334     phyif = (phy_if_t)ifptr;
1335
1336     switch (atype)
1337     {
1338     case FRI_PEERADDR :
1339         type[0] = NA_PEER;
1340         break;
1341
1342     case FRI_BROADCAST :
1343         type[0] = NA_BROADCAST;
1344         break;
1345
1346     default :
1347         type[0] = NA_ADDRESS;
1348         break;
1349     }
1350
1351     type[1] = NA_NETMASK;
1352
1353     if (v == 6) {
1354         lif_if_t idx = 0;
1355
1356         do {
1357             idx = net_lifgetnext(net_data, phyif, idx);
1358             if (net_getlifaddr(net_data, phyif, idx, 2, type,
1359                             array) < 0)
1360                 return -1;
1361             if (!IN6_IS_ADDR_LINKLOCAL(&v6addr[0].sin6_addr) &&
1362                 !IN6_IS_ADDR_MULTICAST(&v6addr[0].sin6_addr))
1363                 break;
1364         } while (idx != 0);
1365
1366         if (idx == 0)
1367             return -1;
1368
1369         return fr_ifpfillv6addr(atype, &v6addr[0], &v6addr[1],
1370                               inp, inpmask);
1371     }
1372
1373     if (net_getlifaddr(net_data, phyif, 0, 2, type, array) < 0)
1374         return -1;
```

new/usr/src/uts/common/inet/ipf/ip_fil_solaris.c

3

```
1376         return fr_ifpfillv4addr(atype, &v4addr[0], &v4addr[1], inp, inpmask);
1377     }
_____unchanged_portion_omitted_____
```