Algorithm for finding hosts which provide a UDP-based service on a large network

gene m. stover

created Friday, 2006 January 13
updated Tuesday, 2006 January 17

Contents

1 What is this?

As of Monday, 2006 January 16, this is almost finished. I'd like to do the SNMP & CIFS examples before I declare it complete.

I describe an algorithm which finds hosts which provide a UDP-based service. The algorithm is fast; it can scan the 10.x.y.z network in about 2.5 hours if you have enough memory & there are few providers of the service. (The algorithm's run time increases as the number of providers increases.)

The algorithm allows you to specify the UDP-based service. It does not know anything of the service.

The algorithm produces short bursts of UDP traffice, but there are significant pauses between the bursts.

The algorithm uses just one thread.

The algorithm does not contain any stealth features.

I have implemented the algorithm in a command line program which is licensed under the terms of the Gnu General Public License. The source code is available for download.

2 Inputs

The inputs to the algorithm are:

- The UDP port number for the desired service.
- A message to send to hosts & to which service providers will reply. The algorithm treats the message as opaque data; it does not interpret the message or the replies. You must create a message which is appropriate for the service that interests you.
- An I/O timeout duration T, such as 10 seconds.
- A list of Internet addresses (IP addresses) to search.

3 Output

The algorithm produces a list of the IP addresses of the hosts which responded to the message.

I assume the list of IP addresses is available on standard input (`stdin`).

4 The Algorithm

The algorithm uses the unix `select` (2) system call. In the pseudocode here, I'll express that as "wait for an event on the socket".

1. Create a UDP socket s.
2. Until there are no more addresses to scan
   (a) Load IP addresses into a *list* in memory. Load as many as will fit, but it's okay if they don't all fit.

(b) Until the *list* is empty

i. Wait for an event on socket s with timeout T. If we have not sent to all of the *list*, we are interested in input & output. If we have sent to all of the *list*, we are only ready for input.

ii. If input is waiting on s, read it with `recvfrom`. Extract the sender's IP address & print it. Remove that IP address from the *list*.

iii. If s is ready for output & we have not sent all of the *list*, send the message to the next IP address on the *list*.

iv. If there was a timeout (i.e., we were only interested in input, & there was none), clear the *list*.

3. Exit

5 How it works

Imagine asking the algorithm to scan a bunch of IP addresses, & it just happens that none of the hosts provide the service that interests you. As the algorithm sends the message to the hosts, none of them reply. So, T seconds after sending the final address, the algorithm times out, clears the list, & announces that it's done & none of the hosts are service providers.

Run time for that instance is T seconds plus the number of seconds to send all the UDP messages.

Now imagine that one of the IP addresses is for a host that does provide the service. Then the algorithm will send many of the IP addresses before receiving a reply from that host. It will print that host's IP address, remove it from the list, & begin the process again. But this time, the process is the case in which none of the hosts are providers. So it's the run time from the "no providers" case (a little over T seconds), plus the number of seconds for the one host to reply, which will be one or two seconds in practice, & just under T seconds in the worst case. So the run time will be roughtly $T + 2$ seconds in practice or $2T$ seconds in the words case.

As you imagine cases in which more & more of the hosts are service providers, it just increases the number of iterations through the list. Each iteration removes one IP address from the list, so that it eventually degeneratese to the case in which none of the hosts are providers.

6 Example: echo

The echo service, which isn't used much (probably for security reasons), is on UDP port 7. You send it a line, & it sends you that same line. The message to make echo reply can be just about anything. Here's the one I used:

```
$ cat src/echo.sca
```

```
hello, world
$
```

I have echo running on one of my computers, at 10.1.0.5. The generator I used generates all the 10.1.[01].y addresses, preceeded by 128,000 others (which might be routed over the Internet – it's a test). It finds 10.1.0.5.

7 Example: daytime

The daytime is an old one, like echo. It's on port 13. I used the same generator as for the echo test. I used a file of one empty line for the message. (Just about any file would have worked. As with echo, daytime does not examine the payload it receives.)

I'm running daytime on 10.1.0.2.

```
$ bin/gen |bin/scan 13 src/daytime.sca >001.txt

MAX_OCTETS_IN_LIST is 1048576.
S_lst_len is 262144.
src/scan.c:159: Message contains 1 octets
src/scan.c:239: S_LoadLst: S_count is 131584. Last is 10.1.1.255.
sendto: Permission denied
src/scan.c:344: sendto (10.1.0.0) failed
sendto: Permission denied
src/scan.c:344: sendto (10.1.0.0) failed
src/scan.c:239: S_LoadLst: S_count is 0. Last is 0.0.0.0.$
$ cat 001.txt
10.1.0.2
$
```

8 Example: ONC RPC portmapper

The portmapper service for ONC RPC is on port 111 (both TCP & UDP, but we're only scanning the UDP ports).

The only trick to scanning for the portmapper is that the message is not obvious. I looked through the `<rpc/*.h>` header files to figure out how to construct a remote call to the portmapper. Then I wrote src/msg-portmapper.c[1] to construct the message & send it to `stdout`. The message does not have to be perfect; it just needs to motivate the portmapper to reply.

Here are the results of the run. I used the same generator as for the previous two examples.

```
$ bin/msg-portmapper >src/portmapper.sca
$ time bin/gen |bin/scan 111 src/portmapper.sca >001.txt
```

[1] `src/msg-portmapper.c`

```
MAX_OCTETS_IN_LIST is 1048576.
S_lst_len is 262144.
src/scan.c:159: Message contains 56 octets
src/scan.c:239: S_LoadLst: S_count is 131584. Last is 10.1.1.255.
sendto: Permission denied
src/scan.c:344: sendto (10.1.0.0) failed
sendto: Permission denied
src/scan.c:344: sendto (10.1.0.0) failed
src/scan.c:239: S_LoadLst: S_count is 0. Last is 0.0.0.0.
real 1m4.705s
user 0m1.010s
sys 0m3.390s
$ cat 001.txt
10.1.0.2
10.1.0.5
$
```

You can see that I'm running the RPC portmapper service at 10.1.0.2 & 10.1.0.5.[2]

9 Example: SNMP

To construct a message for SNMP, I used a remote printer-controlling application to query a printer while I had a network sniffer running. The sniffer showed the forty or so octets that the application sent to the printer in its first transmission. I typed those numbers into a Lisp expression which dumped them as octets into the src/snmp.sca[3] file.

SNMP is on port 161.

After that, I ran the scanner on a LAN which contains a lot of SNMP devices, & the scanner located most of them. It worked better when the timeout was 60 seconds.

I won't show the results here because I'm writing this essay at home, & that network with all the SNMP devices is at a customer's site.

10 Example: CIFS

A Source Code

The program which scans is src/scan.c[4] .

[2] That means that 10.1.0.4 & 10.1.0.9 have problems. Oh well.

[3] `snmp.sca`

[4] `scan.c`

src/gen.c[5] generates IP addresses from some hard-coded data. It's for testing or demo purposes.

src/counter.c[6] is a simple UDP service that can be helpful for tests or demos.

src/msg-portmapper.c[7] generates the message for scanning for the ONC RPC `portmapper` service.

- Makefile[8] , 710 octets
- Makefile.msdos[9] , 2260 octets
- bin/
  - bin/counter[10] , 8979 octets
  - bin/counter.exe[11] , 36864 octets
  - bin/gen[12] , 4840 octets
  - bin/gen.exe[13] , 28672 octets
  - bin/msg-portmapper[14] , 6771 octets
  - bin/scan[15] , 14245 octets
  - bin/scan.exe[16] , 45056 octets
- src/
  - src/counter.c[17] , 6249 octets
  - src/daytime.sca[18] , 1 octets
  - src/debug.c[19] , 1921 octets
  - src/debug.h[20] , 1532 octets
  - src/echo.sca[21] , 13 octets
  - src/gen.c[22] , 1252 octets

[5] `gen.c`
[6] `counter.c`
[7] `msg-portmapper.c`
[8] `Makefile`
[9] `Makefile.msdos`
[10] `counter`
[11] `counter.exe`
[12] `gen`
[13] `gen.exe`
[14] `msg-portmapper`
[15] `scan`
[16] `scan.exe`
[17] `counter.c`
[18] `daytime.sca`
[19] `debug.c`
[20] `debug.h`
[21] `echo.sca`
[22] `gen.c`

- src/msg-portmapper.c[23] , 3955 octets
- src/portmapper.sca[24] , 56 octets
- src/scan.c[25] , 12735 octets
- src/snmp.sca[26] , 40 octets
- src/this.h[27] , 1849 octets

```
/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/counter.c,v 1.4 2006/01/18 03:45:54 gene
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */

/*
 * This is a simple service, sort of like "echo" except that it
 * returns a number, not the packet that you send it.
 */

#include "this.h"

/*
 * Port where we are looking for live hosts.
 */
static u_short S_port = 0;

/*
 */
static char *S_progname = "";
```

[23] msg-portmapper.c
[24] portmapper.sca
[25] scan.c
[26] snmp.sca
[27] this.h

```
/*
 */
enum { ACTION_RUN, ACTION_HELP };
static int S_action = ACTION_RUN;

/*
 */
static int
S_CommandLine (int argc, char *argv[])
{
  int rc = 0;
  int optind = 1;

  S_progname = argv[0];
  if (optind < argc) {
    if (sscanf (argv[optind], "%hu", &S_port) == 1) {
      if (0 < S_port && S_port <= 65535) {
        ++optind;
        if (optind < argc) {
          fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
          fprintf (stderr, " Ignoring extra command line arguments.");
        }
      } else {
        fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
        fprintf (stderr, " %d is out of range for a UDP port.", S_port);
        S_action = ACTION_HELP;
      }
    } else {
      fprintf (stderr, "\n%s:%d:",  __FILE__, __LINE__);
      fprintf (stderr, " Cannot parse \"%s\" into a port", argv[optind]);
      fprintf (stderr, " number.");
      S_action = ACTION_HELP;
    }
  } else {
    fprintf (stderr, "\n%s:%d:",  __FILE__, __LINE__);
    fprintf (stderr, " Missing port number & name of message file");
    fprintf (stderr, " on the command line.");
    S_action = ACTION_HELP;
  }
  return rc;
}

/*
 */
static int
S_Help ()
{
  printf ("\nUsage: %s PORT", S_progname);
  printf ("\nPORT is the UDP port number");
  printf ("\n");
```

```
  return 0;
}

/*
 */
static SOCKET
S_CloseSocket (SOCKET s)
{
  if (s != INVALID_SOCKET) {
#if IS_UNIX
    close (s);
#endif
#if IS_WINDERS
    closesocket (s);
#endif
  }
  return INVALID_SOCKET;
}

/*
 */
static SOCKET
S_MakeSocket ()
{
  SOCKET s;
  struct sockaddr_in sin;
  struct sockaddr *sa;

  s = socket(PF_INET, SOCK_DGRAM, 0);
  if (s != INVALID_SOCKET) {
    bzero (&sin, sizeof sin);
    sin.sin_family = AF_INET;
    sin.sin_port = htons (S_port);
    sin.sin_addr.s_addr = INADDR_ANY;
    sa = (struct sockaddr *) &sin;
    if (bind (s, sa, sizeof sin) == 0) {
      /* good */
    } else {
      fprintf (stderr, "\n");
      perror ("bind");
      fprintf (stderr, "%s:%d: bind failed", __FILE__, __LINE__);
      S_CloseSocket (s);
      s = INVALID_SOCKET;
    }
  } else {
    fprintf (stderr, "\n");
    perror ("socket");
    fprintf (stderr, "%s:%d: socket failed", __FILE__, __LINE__);
  }
  fprintf (stderr, "\n%s:%d: trace: s is %d", __FILE__, __LINE__, s);
```

```
  return s;
}

/*
 */
static int
S_DoRecv (struct sockaddr_in *sin, SOCKET s)
{
  int rc = 0, i;
  char buffer[8000];
  struct sockaddr *sa;
  int sin_len;

  fprintf (stderr, "\n%s:%d: trace: s is %d", __FILE__, __LINE__, s);
  bzero (sin, sizeof *sin);
  sin_len = sizeof *sin;
  sa = (struct sockaddr *) sin;
  i = recvfrom (s, buffer, sizeof buffer, 0, sa, &sin_len);
  if (i > 0) {
    /* good */
  } else {
    fprintf (stderr, "\n");
    perror ("recvfrom");
    fprintf (stderr, "%s:%d:", __FILE__, __LINE__);
    fprintf (stderr, " recvfrom failed");
    rc = 219;
  }
  return rc;
}

/*
 */
static int
S_DoSend (int count, struct sockaddr_in *sin, SOCKET s)
{
  int rc = 0, i;
  struct sockaddr *sa;
  char buffer[12];

  sprintf (buffer, "%d\n", count);
  sa = (struct sockaddr *) sin;
  i = sendto (s, buffer, strlen (buffer), 0, sa, sizeof *sin);
  if (i > 0) {
    /* good */
  } else {
    fprintf (stderr, "\n");
    perror ("sendto");
    fprintf (stderr, "%s:%d: sendto failed", __FILE__, __LINE__);
    rc = 219;
  }
```

```
  return rc;
}

/*
 */
static int
S_Loop (SOCKET s)
{
  int rc = 0;
  int count = 0;
  struct sockaddr_in addy;

  while (rc == 0) {
    if (S_DoRecv (&addy, s) == 0) {
      printf ("\n%s", inet_ntoa (addy.sin_addr)); fflush (stdout);
      if (S_DoSend (count, &addy, s) == 0) {
        /* good */
        ++count;
      } else {
        fprintf (stderr, "\n%s:%d: S_DoSend failed", __FILE__, __LINE__);
        rc = 3;
      }
    } else {
      fprintf (stderr, "\n%s:%d: S_DoRecv failed", __FILE__, __LINE__);
      rc = -172;
    }
  }
  return rc;
}

/*
 */
static int
S_Run ()
{
  int rc = 0;
  SOCKET s;

  s = S_MakeSocket ();
  if (s != INVALID_SOCKET) {
    if (S_Loop (s) == 0) {
      /* good */
    } else {
      fprintf (stderr, "\n%s:%d: S_Loop failed", __FILE__, __LINE__);
      rc = 134;
    }
    S_CloseSocket (s);
  } else {
    fprintf (stderr, "\n%s:%d: S_MakeSocket failed", __FILE__, __LINE__);
    rc = -116;
```

```
  }
  return rc;
}

/*
 */
static int
S_Dispatch ()
{
  int rc = 0;

  switch (S_action) {
  case ACTION_RUN:
    rc = S_Run ();
    break;
  case ACTION_HELP:
    rc = S_Help ();
    break;
  default:
    fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
    fprintf (stderr, " S_action is %d.", S_action);
    fprintf (stderr, "  This should never happen!");
    rc = 102;
  }
  return rc;
}

int
main (int argc, char *argv[])
{
  int rc = 0;

  if (S_CommandLine (argc, argv) == 0) {
    if (S_Dispatch () == 0) {
      /* good */
    } else {
      fprintf (stderr, "\n%s:%d: S_Dispatchfailed", __FILE__, __LINE__);
      rc = 234;
    }
  } else {
    fprintf (stderr, "\n%s:%d: S_CommandLine failed", __FILE__, __LINE__);
    rc = 234;
  }
  return rc == 0 ? EXIT_SUCCESS : EXIT_FAILURE;
}

/* --- end of file --- */

/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/debug.h,v 1.3 2006/01/18 03:47:30 gene E
```

```
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */

#if __cplusplus
extern "C" {
#endif

  /*
   * SOME DEBUGGING STOOF
   */
#define Trace(moo) DEBUG_Print (__FILE__, __LINE__, moo)

  /*
   */
  Boolean DEBUG_IsVerbose (void);

  /*
   * Print the message if debugging Is Verbose.
   */
  void DEBUG_Print (char file[], int line, char fmt[], ...);

  /*
   * It's appropriate to call this function like this:
   * DEBUG_PrintLastError (__FILE__, __LINE__,
   *                       "name-of-function-which-caused-error");
   */
  void DEBUG_PrintLastError (char file[], int line, char moo[]);

#if __cplusplus
}
#endif

/* --- end of file --- */
```

```
/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/gen.c,v 1.4 2006/01/18 03:46:22 gene Exp
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */

#include "this.h"

int
main ()
{
  int w, x, y, z;

  /* 10.1.y.z */
  for (w = 10; w <= 10; ++w) {
    for (x = 1; x <= 1; ++x) {
      for (y = 0; y <= 10; ++y) {
        for (z = 0; z <= 255; ++z) {
          printf ("%d.%d.%d.%d\n", w, x, y, z);
        }
      }
    }
  }
  return 0;
}

/* --- end of file --- */

/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/msg-portmapper.c,v 1.2 2006/01/18 03:46:
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
```

```
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */

#include "this.h"

/*
 */
static char *S_progname = "";

/*
 */
enum { ACTION_RUN, ACTION_HELP };
static int S_action = ACTION_RUN;

/*
 */
static int
S_CommandLine (int argc, char *argv[])
{
  int rc = 0;
#if IS_UNIX
  int c;
#endif
#if IS_WINDERS
  int optind = 1;
#endif

  S_progname = argv[0];
#if IS_UNIX
  while (rc == 0 && (c = getopt (argc, argv, "h")) != -1) {
    switch (c) {
    case 'h':
      S_action = ACTION_HELP;
      break;
    default:
      fprintf (stderr, "\n%s:%d: warning:", __FILE__, __LINE__);
      fprintf (stderr, " Ignoring unexpected command line");
      fprintf (stderr, " option \"%c\".", c);
    }
```

```
  }
#endif
#if IS_WINDERS
  /* Winders doesn't have getopt. */
  while (rc == 0 && optind < argc && argv[optind][0] == '-') {
    if (strcmp (argv[optind], "-h") == 0) {
      S_action = ACTION_HELP;
    } else {
      fprintf (stderr, "\n%s:%d: Ignoring unexpected switch, \"%s\".",
               __FILE__, __LINE__, argv[optind]);
    }
    ++optind;
  }
#endif
  if (optind < argc) {
    fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
    fprintf (stderr, " Ignoring extra command line arguments.");
  }
  return rc;
}

/*
 */
static int
S_Help ()
{
  printf ("\nUsage: %s [-h]", S_progname);
  printf ("\n");
  return 0;
}

/*
 */
static int
S_Run ()
{
  int rc = 0;
  XDR xdrs;
  struct rpc_msg msg;
  struct pmap spmap;

  xdrstdio_create (&xdrs, stdout, XDR_ENCODE);
  bzero (&msg, sizeof msg);
  msg.rm_xid = 17;                      /* msg id, pulled from an orifice */
  msg.rm_direction = CALL;
  msg.ru.RM_cmb.cb_rpcvers = 2;
  msg.ru.RM_cmb.cb_prog = PMAPPROG;
  msg.ru.RM_cmb.cb_vers = PMAPVERS;
  msg.ru.RM_cmb.cb_proc = PMAPPROC_NULL;
  if (xdr_callmsg (&xdrs, &msg)) {
```

```
    bzero (&spmap, sizeof spmap);
    /* We'll ask the portmapper for the portmapper. */
    spmap.pm_prog = PMAPPROG;
    spmap.pm_vers = PMAPVERS;
    spmap.pm_prot = IPPROTO_UDP;
    /* spmap_pm_port = ?; */
    if (xdr_pmap (&xdrs, &spmap)) {
      /* good */
    } else {
      fprintf (stderr, "\n%s:%d: xdr_pmap failed", __FILE__, __LINE__);
      rc = 234;
    }
  } else {
    fprintf (stderr, "\n%s:%d: xdr_callmsg failed", __FILE__, __LINE__);
    rc = 333;
  }
  return rc;
}

/*
 */
static int
S_Dispatch ()
{
  int rc = 0;

  switch (S_action) {
  case ACTION_RUN:
    rc = S_Run ();
    break;
  case ACTION_HELP:
    rc = S_Help ();
    break;
  default:
    fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
    fprintf (stderr, " S_action is %d.", S_action);
    fprintf (stderr, "  This should never happen!");
    rc = 102;
  }
  return rc;
}

int
main (int argc, char *argv[])
{
  int rc = 0;

  if (S_CommandLine (argc, argv) == 0) {
    if (S_Dispatch () == 0) {
      /* good */
```

```
    } else {
      fprintf (stderr, "\n%s:%d: S_Dispatchfailed", __FILE__, __LINE__);
      rc = 234;
    }
  } else {
    fprintf (stderr, "\n%s:%d: S_CommandLine failed", __FILE__, __LINE__);
    rc = 234;
  }
  return rc == 0 ? EXIT_SUCCESS : EXIT_FAILURE;
}

/* --- end of file --- */

/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/scan.c,v 1.4 2006/01/18 03:46:37 gene Ex
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */

#include "this.h"

/*
 * Maximum number of octets for the list.
 */
#define MAX_OCTETS_IN_LIST (1 << 16 /* 64 kilobyte */)

/*
 */
#define S_lst_len (MAX_OCTETS_IN_LIST / sizeof (struct in_addr))
static struct in_addr S_lst[S_lst_len];
static size_t S_count = 0;
static size_t S_i = 0;

/*
 * Port where we are looking for live hosts.
```

```
 */
static u_short S_port = 0;

/*
 * Timeout (in seconds) to wait for input on the socket.
 * Set with the "-t" command line option.
 */
static int S_timeout = 10;

/*
 * Pathname of file from which we read the message to send.
 */
static char *S_pathname;

/*
 * The message to send.  Maximum size is 8,000 octets because
 * that's pretty much the maximum payload of a UDP packet.
 */
static char S_message[8000];
static int S_message_len;

/*
 */
static char *S_progname = "";

/*
 */
enum { ACTION_RUN, ACTION_HELP };
static int S_action = ACTION_RUN;

/*
 */
static int
S_Init ()
{
  int rc = 0;
#if IS_WINDERS
  WORD w;
  static WSADATA wsadata;
#endif

  srand (time (NULL) + getpid ());
#if IS_WINDERS
  w = MAKEWORD (2, 0);
  if (WSAStartup (w, &wsadata) == 0) {
    /* good */
  } else {
    fprintf (stderr, "\n%s:%d: WSAStartup failed", __FILE__, __LINE__);
    rc = 56;
  }
```

```
#endif
  return rc;
}

/*
 */
static int
S_CommandLine (int argc, char *argv[])
{
  int rc = 0;
#if IS_UNIX
  int c;
#endif
#if IS_WINDERS
  int optind = 1;
#endif

  S_progname = argv[0];
#if IS_UNIX
  while (rc == 0 && (c = getopt (argc, argv, "t:")) != -1) {
    switch (c) {
    case 't':
      if (sscanf (optarg, "%d", &S_timeout) == 1) {
        /* good */
      } else {
        fprintf (stderr, "\n%s:%d: warning:", __FILE__, __LINE__);
        fprintf (stderr, " Can't parse \"%s\" into an integeral", optarg);
        fprintf (stderr, " timeout.  Using 10 seconds for the timeout.");
        S_timeout = 10;
      }
      break;
    default:
      fprintf (stderr, "\n%s:%d: warning:", __FILE__, __LINE__);
      fprintf (stderr, " Ignoring unexpected command line");
      fprintf (stderr, " option \"%c\".", c);
    }
  }
#endif
#if IS_WINDERS
  /* Winders doesn't have getopt. */
  while (rc == 0 && optind < argc && argv[optind][0] == '-') {
    if (strcmp (argv[optind], "-t") == 0) {
      ++optind;
      if (optind < argc) {
        if (sscanf (argv[optind], "%d", &S_timeout) == 1) {
          /* good */
        } else {
          fprintf (stderr, "\n%s:%d: warning:", __FILE__, __LINE__);
          fprintf (stderr, " Can't parse \"%s\" into an integeral",
                  argv[optind]);
```

```
          fprintf (stderr, " timeout.  Using 10 seconds for the timeout.");
          S_timeout = 10;
        }
      } else {
        fprintf (stderr, "\n%s:%d: warning:", __FILE__, __LINE__);
        fprintf (stderr, " Must have integral number of seconds following");
        fprintf (stderr, " the \"-t\" argument.  Assuming the default,");
        fprintf (stderr, " 10 seconds.");
        S_timeout = 10;
      }
    } else {
      fprintf (stderr, "\n%s:%d: Ignoring unexpected switch, \"%s\".",
               __FILE__, __LINE__, argv[optind]);
    }
    ++optind;
  }
#endif

  if (optind < argc) {
    if (sscanf (argv[optind], "%hu", &S_port) == 1) {
      ++optind;
      if (optind < argc) {
        S_pathname = argv[optind];
        ++optind;
        if (optind < argc) {
          fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
          fprintf (stderr, " Ignoring extra command line arguments.");
        }
      } else {
        fprintf (stderr, "\n%s:%d:",  __FILE__, __LINE__);
        fprintf (stderr, " Missing name of message file on the");
        fprintf (stderr, " command line.");
        S_action = ACTION_HELP;
      }
    } else {
      fprintf (stderr, "\n%s:%d:",  __FILE__, __LINE__);
      fprintf (stderr, " Cannot parse \"%s\" into a port", argv[optind]);
      fprintf (stderr, " number.");
      S_action = ACTION_HELP;
    }
  } else {
    fprintf (stderr, "\n%s:%d:",  __FILE__, __LINE__);
    fprintf (stderr, " Missing port number & name of message file");
    fprintf (stderr, " on the command line.");
    S_action = ACTION_HELP;
  }
  return rc;
}

/*
```

```
 */
static int
S_Help ()
{
  printf ("\nUsage: %s [-t SECONDS] PORT FILENAME", S_progname);
  printf ("\nPORT is the UDP port number");
  printf ("\nFILENAME identifies a file which contains the message to");
  printf ("\n    send to each host to see if it's alive.");
  printf ("\n");
  return 0;
}

/*
 */
static int
S_LoadMessage ()
{
  int rc = 0;
  static char mode[] = "rb";
  FILE *fp;

  fp = fopen (S_pathname, mode);
  if (fp != NULL) {
    S_message_len = fread (S_message, 1, sizeof S_message, fp);
    if (S_message_len > 0) {
      /* good */
      fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
      fprintf (stderr, " Message contains %d octets", S_message_len);
    } else {
      DEBUG_PrintLastError (__FILE__, __LINE__, "S_LoadMessage");
      rc = 121;
    }
    fclose (fp);
  } else {
    DEBUG_PrintLastError (__FILE__, __LINE__, "S_LoadMessage");
    rc = 55;
  }
  return rc;
}

/*
 */
static SOCKET
S_CloseSocket (SOCKET s)
{
  if (s != INVALID_SOCKET) {
#if IS_UNIX
    close (s);
#endif
#if IS_WINDERS
```

```
    closesocket (s);
#endif
  }
  return INVALID_SOCKET;
}

/*
 */
static SOCKET
S_MakeSocket ()
{
  SOCKET s;

  fprintf (stderr, "\n%s:%d: S_MakeSocket", __FILE__, __LINE__);
  s = socket (PF_INET, SOCK_DGRAM, 0);
  if (s != INVALID_SOCKET) {
    /* good */
  } else {
    DEBUG_PrintLastError (__FILE__, __LINE__, "S_MakeSocket");
  }
  return s;
}

/*
 */
static void
S_Trim (char x[])
{
  while (strlen (x) > 0 && isspace (x[strlen(x) - 1])) {
    x[strlen(x) - 1] = '\0';
  }
}

/*
 */
static int
S_ReadAddy (struct in_addr *addy)
{
  int rc = 0;
  char line[100];

  if (fgets (line, sizeof line, stdin) == line) {
    S_Trim (line);
    bzero (addy, sizeof *addy);
    /* sin->sin_family = AF_INET; */
    /* sin->sin_port = htons (S_port); */
    /* sin->sin_addr.s_addr = inet_addr (line); */
    addy->s_addr = inet_addr (line);
  } else {
    /*
```

```
     * Probably normal end of file, though I suppose some other
     * problems are possible.
     */
    rc = 3;
  }
  return rc;
}

/*
 */
static void
S_RandomizeLst ()
{
  size_t i, r;
  struct in_addr tmp;

  for (i = 1; i < S_count; ++i) {
    r = rand () % (S_count - i) + i;
    memcpy (&tmp, &S_lst[i], sizeof S_lst[i]);
    memcpy (&S_lst[i], &S_lst[r], sizeof S_lst[i]);
    memcpy (&S_lst[r], &tmp, sizeof S_lst[i]);
  }
}

/*
 */
static int
S_LoadLst ()
{
  struct in_addr addy;

  S_count = 0;
  while (S_count < S_lst_len && S_ReadAddy (&addy) == 0) {
    memcpy (&S_lst[S_count++], &addy, sizeof S_lst[S_count]);
  }
  fprintf (stderr, "\n%s:%d: S_LoadLst:", __FILE__, __LINE__);
  fprintf (stderr, " S_count is %d.", S_count);
  fprintf (stderr, " Last is %s.", inet_ntoa (S_lst[S_count - 1]));
  return S_count > 0 ? 0 : -123;
}

/*
 */
static int
S_CmpInAddr (const void *va, const void *vb)
{
  int cmp = 0;
  struct in_addr *ina, *inb;

  ina = (struct in_addr *) va;
```

```
  inb = (struct in_addr *) vb;
  if (ina->s_addr < inb->s_addr) cmp = -1;
  else if (ina->s_addr == inb->s_addr) cmp = 0;
  else cmp = 1;
  return cmp;
}

/*
 */
static void
S_RemoveAddy (struct in_addr *addy)
{
  struct in_addr *elt;

  elt = lfind (addy, S_lst, &S_count, sizeof S_lst[0], &S_CmpInAddr);
  if (elt != NULL) {
    /*
     * Remove the element from the list by copying the last element
     * on top of it, then decrementing the count.
     */
    if (S_count >= 2 && elt < &S_lst[S_count - 1]) {
      /*
       * List has at least 2 elements, & the one to remove
       * is not the last.
       */
      memcpy (elt, &S_lst[--S_count], sizeof *elt);
    } else if (S_count >= 1) {
      /*
       * List has just one element, or the one to remove is
       * the last one.
       */
      --S_count;
    } else {
      /* List is already empty. */
    }
    S_i = 0; /* reset the write location */
  } else {
    /* fprintf (stderr, ".  Not in list???"); */
  }
}

/*
 */
static SOCKET
S_DoRecv (SOCKET s)
{
  int i;
  char buffer[8000];
  struct sockaddr_in sin;
  struct sockaddr *sa;
```

```
  int sin_len;

  bzero (&sin, sizeof sin);
  sin_len = sizeof sin;
  sa = (struct sockaddr *) &sin;
  i = recvfrom (s, buffer, sizeof buffer, 0, sa, &sin_len);
  if (i > 0) {
    printf ("%s\n", inet_ntoa (sin.sin_addr));
    S_RemoveAddy (&sin.sin_addr);
  } else {
    DEBUG_PrintLastError (__FILE__, __LINE__, "S_DoRecv");
    /* s = S_CloseSocket (s); */
  }
  return s;
}

/*
 */
static SOCKET
S_DoSend (SOCKET s)
{
  int i;
  struct sockaddr_in sin;
  struct sockaddr *sa;

  if (S_count > 0) {
    if (S_i >= S_count) S_i = 0;
    fprintf (stderr, "\n%s:%d: Send %s", __FILE__, __LINE__,
             inet_ntoa (S_lst[S_i]));
    bzero (&sin, sizeof sin);
    sin.sin_family = AF_INET;
    sin.sin_addr = S_lst[S_i];
    sin.sin_port = htons (S_port);
    sa = (struct sockaddr *) &sin;
    i = sendto (s, S_message, S_message_len, 0, sa, sizeof sin);
    if (i > 0) {
      /* good */
    } else {
      DEBUG_PrintLastError (__FILE__, __LINE__, "S_DoSend");
      fprintf (stderr, "%s:%d: Removing %s", __FILE__, __LINE__,
               inet_ntoa (S_lst[S_i]));
      S_RemoveAddy (&S_lst[S_i]);
      s = S_CloseSocket (s);
    }
    ++S_i;
  } else {
    /*
     * The list is empty, so don't send anyting at all.
     */
    fprintf (stderr, "\n%s:%d: S_DoSend:", __FILE__, __LINE__);
```

```
    fprintf (stderr, " list is empty");
  }
  return s;
}

/*
 */
static int
S_Loop ()
{
  int rc = 0;
  SOCKET s = INVALID_SOCKET;
  fd_set rfds, wfds;
  struct timeval stv;
  int i;

  while (rc == 0 && S_count > 0) {
    if (s == INVALID_SOCKET) {
      s = S_MakeSocket (); S_i = 0; S_RandomizeLst ();
    }
    if (s != INVALID_SOCKET) {
      FD_ZERO (&rfds); FD_SET (s, &rfds);
      FD_ZERO (&wfds); FD_SET (s, &wfds);
      stv.tv_sec = S_timeout; stv.tv_usec = 0;
      assert (FD_ISSET (s, &rfds));
      assert (FD_ISSET (s, &wfds));
      i = select (s + 1,
                  /* always want to read */
                  &rfds,
                  /* only want to write if we have not written
                   * everything.  Otherwise, we busy-write */
                  S_i < S_count ? &wfds : NULL,
                  NULL, /* never exceptions */
                  &stv);
      if (i > 0) {
        if (FD_ISSET (s, &rfds)) {
          s = S_DoRecv (s);
        }
        if (FD_ISSET (s, &wfds)) {
          s = S_DoSend (s);
        }
      } else {
        /* Timeout */
        S_count = 0;
      }
    } else {
      fprintf (stderr, "\n%s:%d: S_MakeSocket failed", __FILE__, __LINE__);
      rc = 3;
    }
  }
```

```
  if (s != INVALID_SOCKET) {
    s = S_CloseSocket (s);
  }
  return rc;
}

/*
 */
static int
S_Run ()
{
  int rc = 0;

  if (S_LoadMessage () == 0) {
    while (S_LoadLst () == 0) {
      if (S_Loop () == 0) {
        /* good */
      } else {
        fprintf (stderr, "\n%s:%d: S_Loop failed", __FILE__, __LINE__);
        rc = 134;
      }
    }
  } else {
    fprintf (stderr, "\n%s:%d: S_LoadMessage failed", __FILE__, __LINE__);
    rc = 154;
  }
  return rc;
}

/*
 */
static int
S_Dispatch ()
{
  int rc = 0;

  switch (S_action) {
  case ACTION_RUN:
    rc = S_Run ();
    break;
  case ACTION_HELP:
    rc = S_Help ();
    break;
  default:
    fprintf (stderr, "\n%s:%d:", __FILE__, __LINE__);
    fprintf (stderr, " S_action is %d.", S_action);
    fprintf (stderr, "  This should never happen!");
    rc = 102;
  }
  return rc;
```

```
}

int
main (int argc, char *argv[])
{
  int rc = 0;

  fprintf (stderr, "\nMAX_OCTETS_IN_LIST is %d.", MAX_OCTETS_IN_LIST);
  fprintf (stderr, "\nS_lst_len is %d.", S_lst_len);
  if (S_Init () == 0) {
    if (S_CommandLine (argc, argv) == 0) {
      if (S_Dispatch () == 0) {
        /* good */
      } else {
        fprintf (stderr, "\n%s:%d: S_Dispatchfailed", __FILE__, __LINE__);
        rc = 234;
      }
    } else {
      fprintf (stderr, "\n%s:%d: S_CommandLine failed", __FILE__, __LINE__);
      rc = 234;
    }
  } else {
    fprintf (stderr, "\n%s:%d: S_Init failed", __FILE__, __LINE__);
    rc = 526;
  }
  return rc == 0 ? EXIT_SUCCESS : EXIT_FAILURE;
}

/* --- end of file --- */

/*
 * $Header: /home/gene/library/website/docsrc/sca/src/RCS/this.h,v 1.5 2006/01/18 03:46:41 gene Ex
 *
 *  Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU Lesser General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this program; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 *  USA
 */
```

```
/* Standard C */
#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* POSIX, unix, linux */
#if IS_UNIX
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <search.h>
#include <rpc/rpc.h>
#include <rpc/pmap_prot.h>
#endif

/* Microsloth Winders */
#if IS_WINDERS
#  include <windows.h>
#  define bzero(p,sz) memset(p,0,sz)
  int getpid ();
  void *lfind(const void *key, const void *base, unsigned int *num,
              unsigned int width, int (*cmp)(const void *, const void *));
#  define CloseSocket closesocket
#else
   /*
    * On non-Winders systems, use these symbols (SOCKET,
    * INVALID_SOCKET, & CloseSocket).
    */
   typedef int SOCKET;
#  define INVALID_SOCKET -1
#endif

/* This */
typedef int Boolean;
#include "debug.h"

/* --- end of file --- */
```

B What about nmap?

Yes, `nmap` can scan like this.

I didn't use `nmap` because my client needed code which was unencumbered

by the license agreement on `nmap`.

"Unencumbered by the GPL? But you, yourself, slapped the GPL on your own code?" Yes, I did, & since I'm the license-holder, I can create a custom, non-GPL license for my client.

"If the user installed `nmap` separately, wouldn't that allow your client's software to work with `nmap` without itself being GPLed?" I believe it would.

"So could you do that? Could you require the user to install `nmap` himself?" No.

References