Mechanics Transport Communications

Academic journal

ISSN 1312-3823

issue 3, 2007

article № 0177

http://www.mtc-aj.com

MONITORING PRESENCE, USABILITY, AND RESPONSE TIMES OF THE DIFFERENT HOST WORKSTATIONS ON LAN OR WAN NETWORKS USING CROSS – PLATFORM, OPEN SOURCE ICMP INTERFACE LIBRARY

Goran VUJAČIĆ, Ljubomir LUKIĆ

goran.vujacic@vzs.edu.yu,

Mr, Goran Vujačić, Beograd, Railway College of Vocational Belgrade,Serbia,
PhD, Ljubomir Lukić, Beograd, Mechanical facyliti Kraljevo,
SERBIA

Abstract: *Nowadays, when computer networks are growing in their organization and infrastructure complexity, it is essential to monitor presence, network characteristics, usability and data transfers between different network hardware, network servers and end user workstations, all in favor of speed, quality and finally, business.*

Key words: *ICMP, cross-platform, networking, library, lightping library*

1. INTRODUCTION

Client – server revolution has brought many advantages and innovations; mainly it has become the epicenter and backbone of the entire network facility and also the standards implementer. There are many software products present on the market today, starting from the network simulators to the IRC or chat programs, mail clients, private intranet chat servers and various dedicated software applications. In this, so called "jungle" of software, not just server administrators, but more often, users need to know if there are remote hosts present on the network. Users want their software to be easy to use, with friendly interface and simple help files; they want software to be fast, attractive and easy to use. On the other side, server administrators want to have more control; they do not care much about the user interface, they want to tune their software to their needs and personal taste; furthermore, they want their software to be extendable, scriptable or based on the *plug-in* architecture.

So what can one computer scientists or engineer do to help?

There are many demands: portability, interface, modularity, *plug-in* architecture, easiness of use, and finally budget. Open source is the philosophy which is used in this solution; it fulfils most of the demands of this enigma. Pretty modest IT budget and the quality of open source solutions of the large specter, force the software designer to see the big picture. The solution presented here follows the GNU philosophy and standards of the open source community. The entire C – GLIBC – Console – GNOME – KDE – Windows™ framework is based on this idea and RFC documents, providing the simple, yet powerful solution to the presented problems and demands. In this solution we are going to present the most significant parts of software, concerning advantages and disadvantages between architectures, software implementations and platform differences.

2. IMPLEMENTATION

Main programming language used in this project is C, both on Windows™ and Linux platforms. C is very powerful, low level language able to access raw operating system functions, which is especially important on the Linux platform. The core of the software is dynamic (shared) library, written entirely in C and based on the ip-utils package (on Linux) from Mike Muss (US Army ballistics laboratory) and the Berkley University in the United States of America. This package offers many useful utilities, among which the ping program is of the most interest. This program is

Linux console program which does the "standard" pinging and is invoked from the shell with `ping <options> <hostname | IP address>`. The ping utility is useful program and is used very often by network administrators; it can also be useful in shell scripts and pipes. And … that's almost all of its possibilities. This program completely implements the ICMP protocol and uses checksum functions that calculate IP packet data and headers length, defined in RFC 793 and 1791. It was released in the early '80s and it can run on every Linux platform I had opportunity to work with. Windows™ solution is based on the Icmp* functions API provided by Microsoft and with Icmp.h and Icmp.lib files. These functions are IcmpCreateFile, IcmpSendEcho and IcmpCloseHandle. These functions are wrapped in the iphlpapi.dll and icmp.dll dynamic link libraries. This solution provides another library which even more simplifies the usage and implementation. I've mentioned that the entire library is written in ANSI C on both platforms. That is the key to the power of this library. I've wrapped all the low level details in bunch of simple to use library calls including the power and the speed of the C language.

On the Linux platform this library is compiled with the GCC (GNU C Compiler) which offers superior optimization and minimum possible file size. On the Windows™ platform I've tried the Microsoft Native C/C++ compiler and the Bloodshed Dev C++ which is based on the *Mingw* project. Microsoft's products offers more possibilities and platform integration, but the file size of Dev C++ is much smaller (as twice as small, without the shared linking). Even though, I've noticed that the Microsoft's compiler has produced more stable and less memory demanding code, so I've decided to use it (file size does not matter that much these days: 6kb with the Dev C++ and about 10k with the Microsoft's C/C++ compiler) . The source code implementations mentioned above differ very much. It was "easy" on the Windows platform: wrap *Icmp* functions, write a few checksums and thread handlers (because we did not want to block the calling process), and that was almost all about that. On the Linux platform things were different. We had to use RAW sockets, realize and/or implement entire ICMP protocol in our program, realize the standard procedures, parse replies, decode errors, preserve the buffers, install the filter(s) on the socket …

Functional differences forced implementing custom functions (which were primarily needed on Linux) and centralized flow handlers. Based on mentioned *Icmp** Windows™ functions I've implemented *icmp_init*, *icmp_send_echo_echo*, and *icmp_get_last_error* functions in the library. They behave almost like the Microsoft's functions, which was their main purpose. You can see the entire source code in the Appendix. With this functions in a shared library began the GUI part of the problem.

2.a Windows implementation

On Windows™, Borland Delphi having the reputation of the best RAD tool (which is true), was used to create the GUI. Delphi uses Object Pascal as its base and very powerful VCL framework.

Fig. 1. ***Delphi interface***

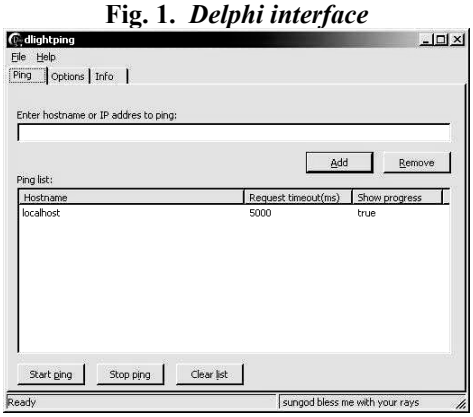

Fig. 2. ***GNOME*** **Interface**

There were no major problems with creating and realizing the GUI (minor problems occurred with callback functions and redefining C structures and types). You can see the Delphi interface below. Few clicks and lines of source were just enough. Every network administrator could have done this, with the language of his choice (or even a script), because he would have the library. Program was tested on the Windows 2000 and Windows XP and worked very stable. Its file size was a little larger, (because of the VCL) but it was compressed with UPX.The situation was much more complicated on Linux. First of all there are several window managers. Most popular managers nowadays are GNOME and KDE so I've decided to write interfaces for

both of them, and, of course, the console version of the program. GNOME and KDE use completely different APIs on top of the X Window Library, or so called Xlib.

2.b GNOME implementation

GNOME was built on top of the GTK+ (or GIMP Toolkit) which is completely written in C. GTK is based on the GLIB library which implements and redefines some GLIBC functions and types and gives easier and friendlier API. Luckily there is a great tool, Anjuta IDE. GNOME implementation was completely written in it. Interface designer Glade was used to produce an UI. Program supports GNU *gettext* package by which it can easily be translated to any language. Main source file contains the callbacks which are triggered when user interacts with the UI. New thread is created per entry, just like in Windows. Maximum number of threads that can be created is 100 per process. This limitation was set because we did not want to create to resource demanding program and to produce bugs that are hard to trace. Lightping library was implemented as a shared library. All functions in the library are called from the thread's core stack. When the thread starts new window opens and shows the progress, as well as the TTL and round trip times. This window can be hidden (useful when there are many hosts in the list) and user can change its background and foreground colors. User can choose to automatically end running threads on exit, save and/or reload last used list on startup and to define how main times to send echo requests. Program is small and versatile and its beta version was very stable on the GNOME 2.6 platform from the SuSE 9.2 Professional distribution, as well as on the Debian Unstable Sid 3.0.1.

2.c KDE implementation

With KDE there is different approach. KDE is based on the Trolltech Qt library and has been built on top of it. Qt library was written in C++ and has wide range of classes; from low level and utility classes to widget and interface parts of various kinds.

KDE application was developed using one of the best IDEs I have ever worked with – KDevelop. The interface part was designed with Qt Designer from Trolltech. It was easier to implement it than with Glade, and SIGNAL → SLOT mechanism is easier to use and understand then the GTK+ callbacks, (one has to write fewer lines of source code thanks to the `uic` compiler). The only thing I do not like is Qt Designer's implementing of slots. One have to subclass the *.ui.h file to get the actual implementation of the signal handlers and the *.cpp file which can be compiled (it's not that annoying once you get used to it). KDE application can be translated easily to (it uses i18n function versus to GTK _ macro). KDE applications look nicer, and can be really an eye candy, because of the superior rendering engine and presence of many window styles and themes. There are two main source files. First one contains KMainWindow class and defines it.

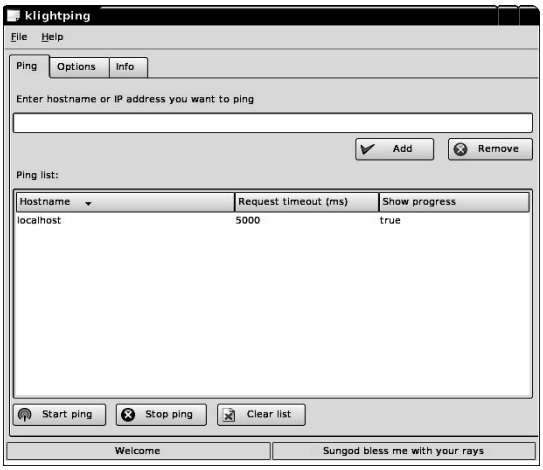

Fig. 3. ***KDE Interface***

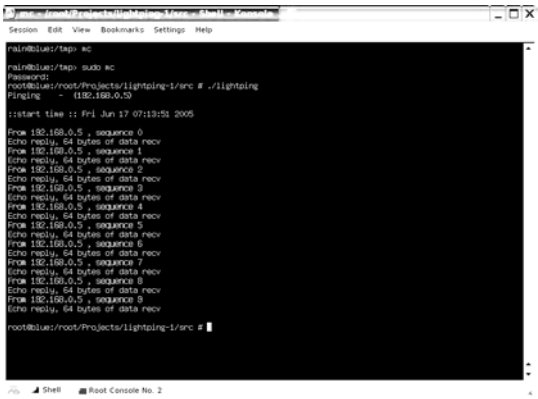

Fig. 4. ***Console implementation***

The second one contains implementation of the central widget (all actual work gets done here). Similar to GNOME application, new thread is created per entry and maximum number of threads per process is 100. User can choose whether to show ping progress and other useful options. When the thread start progress window pops up and shows progress. When entire pinging process is done report is created and displayed showing TTL, and round trip times. Ping list can be saved or retrieved from a file, which is very useful if user wants to check hosts or his local area network, or his other favorite host list. Program is a bit more larger then the GNOME implementation it haven't crashed so far and has been tested on KDE 3.3 and KDE 3.4 from the

SuSE 9.2 Professional distribution, as well as on the Debian Unstable Sid 3.0.1.

2.d Console implementation

Console implementation was the easiest part of the implementation.

Program was written in C in ViM editor and compiled with GCC, linking the ligthping library. Only problem which existed was passing long list of hosts which were going to be checked on the command line. This problem was solved by the means of implementing a pipe (forking the parent process), so one can easily write a command `cat ./myhostlist > lightping`.Of course this program has the smallest file size, because it has no UI, it is very fast and convenient to be used in shell scripts. It worked very stable and had a few minor bugs that were successfully fixed. It was tested on the SuSE 9.2 and Debian Unstable Sid 3.0.1.

3. DEPLOYMENT AND VARIOUS CHARACTERISTICS

Even though this projects is open source it has some limitations depending on the interface implementation used:

On the windows platform it is because of the Delphi, which is not free tool; it is commercial software licensed from Borland Corporation.

On the KDE platform GPL + Qt license apply (basically Qt license is free for the open source programs).

On the GTK/GNOME platform program is completely under the GPL.

Every platform has some advantages and disadvantages. To be able to run a program one must have root permission or have the program installed as suid root on the Linux platform. This is because of the raw sockets. There is no such limitation on Windows; every standard user account can be used to run the program.On Windows Icmp functions are implemented in the icmp.dll library and can not be used for fine-tuning and reimplementation.Furthermore GTK framework does not allow program to have setuid root. If this is the case GTK refuses to initialize and quits. I've solved this using a pipe as a mean of transferring data from the library and backwards.There are installation problems (on Linux platform), too.If you are trying to compile the library in the GNOME environment you need to have libgnome, libkeyring and libgtk2 development packages, as well as other GNOME development files.

For KDE application to compile you need Qt version 3.3 and above and at least KDE 3.3. There are no special requirements on the Windows platform, but I'm not sure how this library works on the Win9x.

Library itself and console application do not have any special requirements, they have been successfully compiled on both platforms and have been thoroughly and heavily tested. This library is just a part of another, larger software project which is currently under development and will have ICMP functions library as a plug-in module.This project is primarily developed for the Linux platform, following the Open source philosophy.

REFERENCES:

[1] http://www.openview.hp.com/
[2] http://www.nagios.org/
[3] http://nsclient.ready2nin.nl/
[4] http://www.net-snmp.org/
[5] http://people.ee.ethz.ch/~oetiker/ webtools/mrtg/
[6] http://www.gnokii.org/
[7] GNU C Compiler
[8] GNOME Interface
[9] Bloodshed Dev C++ which is based on the *Mingw*
[10] Microsoft Native C/C++ compiler Bloodshed Dev C++ which is based on the *Mingw* project
[11] ICMP protocol
[12] RFC 793 and 1791
[13] SuSE 9.2.

LAN ИЛИ WAN МРЕЖИ, ИЗПОЛЗВАЩИ –БИБЛИОТЕКИ НАПРЕЧНИ ПЛАТФОРМИ, ОТВОРЕН ИЗТОЧНИК И ИНТЕРФЕЙС ICMP

Горан Вуячич, Любомир Лукич

Горан Вуячич, Висше железопътно училище, 11000-Белград,
д-р Любомир Лукич, Машинен факултет в Кралево,
СЪРБИЯ

Резюме: *Днес, когато компютърните мрежи се разрастват организационно и като инфраструктурна сложност, е важно да се наблюдава наличието, мрежовите характеристики, използваемостта и пренасянето на данни между различния мрежови хардуеър, сървърите на мрежите и работните устройства на крайните потребители и всичко това да е в полза на скоростта, качеството и накрая бизнеса.*

Ключови думи: *ICMP, напречна платформа, мрежова библиотека.*